NTNU
Norwegian University of Science and Technology

Practices of Agile Software Product-Line Engineering
A qualitative assessment of empirical studies

Snorre Gylterud

Master of Science in Computer Science
Submission date: August 2009
Supervisor: Torgeir Dingsøyr, IDI
Co-supervisor: Kyo C. Kang, POSTECH

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Software engineering is evolving as we speak. The need to develop software efficiently with high quality is becoming more important for companies and their core competencies in various domains. Software Product-Line Engineering is a software engineering approach where commonalities in the product portfolio are exploited through reuse. Usually, this has a large initial investment. Agile Software Development is a collection of methods and practices that tries to implement products through team-orientation, embracing change and reduced design effort. Lately researchers have studied the theoretical combination between these two approaches to Software Engineering. This assignment is to get an understanding of how companies combine the best of these two approaches to control and streamline the process of making software.

Assignment given: 16. February 2009
Supervisor: Torgeir Dingsøyr, IDI

Abstract

This thesis elaborated on the how Software Product-Line Engineering is combined with Agile Software Development to improve Software Engineering, through investigating published case studies and performing interviews in several companies. This combination are often described as Agile Software Product-Line Engineering and our study aimed to *describe what agility is for software product lines* and find out more on *how this approach could be realized*. Agile Software Product-Line Engineering could reap benefits from the best of the two software engineering approaches combining long term strategic efforts with short term agility.

By following a specified research method that combines qualitative research methods we were able to ensure validity in our analyses and generalize the findings of this study. We used both semi-structured interviews and textual analysis techniques.

The companies under study seem to combine Software Product-Line Engineering and Agile Software Development with success, reducing initial investment and exploiting reuse, and we found several practices that are interesting for further study. Based on these practices we present our view of a top-down approach to Agile Software Product-Line Engineering starting with several characteristics and a proposal for a definition of the field. Further, a framework for implementing the approach based on our research is presented, before we describe our thoughts on how the practice areas of Software Product-Line Engineering can be combined with Agile Software Development practices.

We think that this thesis could be used as a guideline for further study and implementation of Agile Software Product-Lines. We believe that the data we cover is comprehensive based on the small existing research field and covers the general ideas of both the fields included in the combination.

Preface

The Master Thesis you are now holding is the final work of my *Master of Science in Computer Science* degree at the *Department of Computer and Information Science* (IDI) at the *Norwegian University of Science and Technology* (NTNU). The thesis was mostly written at *Pohang University of Science and Technology* (POSTECH) in South Korea in combination with an exchange year. For me it was important to create a win-win situation, and did that by having a problem that spanned over both my supervisors' research fields. An obstacle for this study was my location and being in a environment where Agile Software Development was not commonly used. This made it hard to do the empirical data collection, but I managed to perform a couple of interviews.

I would like to take the opportunity to thank my supervisor at NTNU Adj. Ass. Prof. Torgeir Dingsøyr. He has supervised me through this Master Thesis work giving helpful hints and directions for this study. He was also positive about me working from a distant country and I am very grateful for that.

Prof. Kang at POSTECH and his lab members have also helped me with the progress of my thesis. They provided me with a good infrastructure for research and helped me within the field of Software Product-Line Engineering.

[illegible]

2009.08.24 Trondheim, Norway Snorre Gylterud

Contents

Figures:

Tables

Appendices

1. Introduction

Software systems and applications influence most people's everyday work and tasks. People of developed countries usually use e-mail, computers with specific software, and machines controlled by software to help them in their day to day life. In the high technology market the time it takes from a product is produced and sold until the product is replaced by a newer or better product ,is getting shorter and shorter while the demand for making new innovative products is constantly present (Schilling 2004). Software can be included in the high technology market which changes constantly. Software can be categorized within the complex high-tech market and one of the definitions of software engineering is *"systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"* (IEEE 1990). Software engineering strives to fulfill customer demands by delivering software to both organizations and individuals. Challenges with software engineering have been discussed for a long time with Brooks' *No Silver Bullet* (1987) as one of the most influential works which still characterizes many of these challenges (Fraser & Mancl 2008). Quality, complexity, and changeability are some of the challenges mentioned. To handle these challenges, several approaches or methods for software engineering have emerged.

These methods are often referred to as development paradigms such as Waterfall Development, Spiral Development, Software Product-Line Engineering, and Agile Software Development. Today we experience recession times in the global market, and customers become more conscious of the value of money which creates opportunities for cost innovation by: 1) offering breeding edge technology for mass-market price; 2) offering variations and customizations as a value; and 3) evolving niche markets to mass markets (Williamson & Zeng 2009). Software Product-Line Engineering could be one solution to exploit these cost innovations in the field of software engineering. On the other hand Boehm (2008) describes the future of software engineering mentioning agility, adaptability and learning as some of the principles that could be valuable in the future. Agile Software Development could be the solution to dealing with these principles. An even more interesting thought is to combine the benefits of these two approaches to software engineering. This could lead to increased quality, reduced time to market, and more exploitation of reuse. This master thesis will elaborate on the hybrid paradigm, Agile Software Product-Line Engineering, consisting of the combination of Software Product-Line Engineering and Agile Software Development.

1.1 Problem Areas

Software Product-Line Engineering create software with a long-term strategic effort, building a platform for reuse and standardization in order to fulfill needs in domains that are relatively stable producing products from the platform (Clements & Northrop 2002a). The approach tries to exploit commonalities, best practices, and known working solutions in the domains in order to reduce time to market and increase quality. The scope of Software Product-Line Engineering can be described as the development of several products for various customers with different needs through a product platform or core asset base. Software Product-Line Engineering has its roots from Feature-Oriented Domain Analysis and several research projects both in the US and Europe (Gylterud 2008). A couple of the significant problems with this approach are the time and money necessary to establish a Software Product Line.

Agile Software Development values iterative methods, close cooperation and communication, team-based flat organizational structure, embracing change, and maximizing value to develop software in a changing market. These methods can be described as a short-term value-based effort that is highly dynamic. Agile Software Development also tries to reduce time to market and increase quality. However, the scope of single-system development is often in unstable domains. Agile Software Development emerged from early iterative methods

(Gylterud 2008). Project management and control are mentioned as problem areas when applying Agile Software Development.

We could try to handle the problems in the two fields by combining the best practices of these two approaches. This could reduce initial investment and efforts mentioned in Software Product-Line Engineering, and introduce more controlled agile methods that can benefit from reuse. The introduction of Agile Software Development in Software Product-Line Engineering called Agile Software Product-Line Engineering introduces new challenges and contradictions that need to be addressed. We looked at these in a literature review last year (Gylterud 2008) and several conferences and articles have discussed this emerging field. In this thesis we wanted to further investigate if the theories and suggestions concurred with the current practices in the software industry. In addition to that, a definition or uniform description of *"what agility is in Software Product-Line Engineering"* was not found in the research. Software Product-Line Engineering is also associated with an up-front investment both in cost and time consumption (Frakes & Kang 2005; Sugumaran et al. 2006), which might be one of the reasons it is not so wide spread among software engineering practitioners.

1.2 Research Questions

To cover these problem areas we established two main research questions. These are named Research Question 1 and 2 (RQ-1 & RQ-2). We also established some sub-questions to state the questions needed to be answered before we could discuss the main RQs.

RQ-1: *"How is Agile Software Development combined with Software Product-Line Engineering in software companies today?"*

RQ-1.1 Which studies handle both agility and software product line engineering?

RQ-1.2 Which criteria have to be obtained to determine the level of agility and Software Product-Line Engineering?

RQ-1.3 How should the criteria be weighted to choose the best suited data material in the community?

RQ-1.4 How is agility introduced in Software Product-Line Engineering and how does it work in the industry?

RQ-1.5 Which practices are used in industry to obtain agility in software product lines?

RQ-2: *"What could characterize a method or a framework to describe agility in software product line engineering?"*

RQ-1.1 What does agility mean for software product line engineering?
RQ-1.2 What characterizes agility in software product line engineering?
RQ-1.3 How can software product line engineering companies incorporate agility into their practices?

RQ-1 handles the multi-case study part of this thesis. It asked us how we investigate Software Product-Line Engineering and Agile Software Development and describes which answers we are looking for. RQ-2 aims to explain the main findings of our study and will be answered in our discussion and our conclusion. Until now, no clear answers to RQ-2's sub questions have been found in research and we are hoping to fill this gap with the thesis at hand.

1.3 Objectives

To support the research questions and have some pinpoints for where to go, we established three goals for this Master Thesis. The goals were:

1. Establish concrete indications from cases on how agility and software product lines work together.

2. Provide characteristics and guidelines for agility and software product lines combined.

3. A valuable contribution to the field and thesis that documents the research done.

1.3.1 *Scope*

In order to achieve these goals we looked at Software Product-Line Engineering and Agile Software Development, two large research fields. We made our scope the introduction of agility in Software Product-Line Engineering and started working from that viewpoint. This means that we try to introduce Agile Software Development into Software Product-Line Engineering and not vice versa. We did this to limit our problem and to reduce the span of the thesis to a feasible problem domain. This was also done to provide a significant contribution to the field.

1.3.2 *Contribution*

This study mainly aims to provide a contribution for the Software Product-Line Engineering field. Currently there are no clear characteristics or perceptions of what agility means for Software Product-Line Engineering. Through this study we wanted to investigate this combination by looking at companies that combine Software Product-Line Engineering with Agile Software Development. Concretely we try to establish a set of characteristics and a framework for agility in Software Product-Line Engineering.

1.4 Target Readers

Since the Software Product-Line Engineering field is strongly influenced by practitioners we wanted to create a thesis that could fit research topics in the area and still address the practical introduction of agility in Software Product-Line Engineering. Readers that are not familiar with either Software Product-Line Engineering or Agile Software Development are advised to read other sources like (Gylterud 2008) to get an overall view of the software engineering approaches, before reading this thesis. Practitioners can skip chapter 3, as they might not be interested in the research method.

1.5 Clarifications

In this thesis we will try to use Software Product-Line Engineering for our software development related topics, while Software Product Lines will be used about the result of a Software Product-Line Engineering effort. In other literature, this notation may not be the case, and the notations can be used with the same meaning.

Agile Software Development is used as a terminology for all the agile methods' principles and practices. Agility is used to express the agility of a certain method or approach.

1.6 Organization of Thesis

We start with a brief introduction to the State of the Art when it comes to Software Product-Line Engineering, Agile Software Development, and Agile Software Product-Line Engineering (Chapter 2). Secondly, we describe the research method used in this study (Chapter 3). Then we document our results on Software Product-Line Engineering practice areas and Agile Software Development practices from the analysis (Chapter 4), before a discussion on these results and an Agile Software Product-Line Engineering framework is presented (Chapter 5). Our conclusion ends this thesis (Chapter 6).

2. State of the Art: Product Line and Agile Software Engineering

This chapter will briefly describe the state of the art practices of Software Product-Line Engineering, Agile Software Development and Agile Software Product-Line Engineering. We focus on explaining the practices which are used in the different approaches to software engineering. The first section will concisely, describe Software Product-Line Engineering and cover its 29 practice areas, first presented by Clements and Northorp (2002a). In the second section we briefly present Agile Software Development and introduce the most important practices used in this approach. Thirdly, we cover the emerging field of Agile Software Product-Line Engineering, and some of the key characteristics we found present to this master thesis. Readers with little or no knowledge about the software development approaches handled in this study may find this information deficient and are advised to read our former literature review (Gylterud 2008) and investigate its sources. All three sections will have a short introduction, overview of the definition, and an assessment of the research and industry of the field, before we explain the practices associated with the respective field.

2.1 Software Product-Line Engineering

The Software Product-Line Engineering field incorporates many of the manufacturing and product development thoughts mainly using a product line which produces products for a variety of customers with different needs. A product line is created by combining various products into an assembly line, which exploits commonalities to efficiently produce the products in a similar manner. In software development it means that we produce our final products through a platform or a core asset base where we *"assemble components"* to fulfill a customers' need (Clements & Northrop 2002a; Pohl et al. 2005). The core asset base is described as the reuse center where all the parts and production plans for our products are located. The parts and plans are used as artifacts when a customer engages in a software development effort and need to create a product. An example frequently used in the field is an Electronic Home System for controlling setting in a digital home. In this example, homes can use the same base system, but have variations based on requirements and needs.

2.1.1 *Definition*

Two definitions are present in the field, separating two concepts (a) Software Product Lines and (b) Software Product-Line Engineering:

a. *"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"* (Clements & Northrop 2002a)

b. *"a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customisation"* (Pohl et al. 2005)

Software Product Lines are the description of the product line in place at a company, whereas Software Product-Line Engineering describes the paradigm or software development approach by using platforms and mass customization. Exploiting commonalities through respective domain and platforms is common between these definitions. The Software Product Lines definition also includes market efforts and strategy. The Software Product-Line Engineering definition uses the word 'mass customization' coming from manufacturing industry meaning production of products in a large scale fitted to individual customers (Pohl et al. 2005).

We see the Software Product-Line Engineering definition as slightly more technical than the Software Product Lines definition, and that reflects the content of the books referenced. Establishing an understanding of the main theory available is necessary before we look further into the practices for Software Product-Line Engineering.

2.1.2 *Status*

In Software Product-Line Engineering, a set of practice areas are suggested to explain the approach and how it can be realized in practice. The practice areas explain how to create and maintain a Software Product Line through software engineering theory and are categorized as follows (Clements & Northrop 2002a):

1. Software Engineering;
2. Technical Management;
3. Organizational Management

Further practice areas inside each category need to consider the platform which contains reusable assets, and the applications or products that are developed to the customers. These can be called a) domain engineering and b) application engineering. We also use the research behind the three approaches to Software Product-Line Engineering namely Proactive, Reactive (incremental) and Extractive (Clements 2002; Krueger 2002) in our thesis. Pros and cons with these methods can be found in the articles mentioned, while Frakes & Kang (2005) describe the more overall case for software reuse. The proactive approach establishes the domain model and the reusable assets before products are built from the platform. The reactive approach is the other extreme and products are developed before the platform elements are created based on a need or opportunity with the product. The extractive approach is somewhat in between, meaning we often use an existing platform or set of products and base a new platform on those efforts meaning we are incrementally building our core asset base.

The research of this field is highly influenced by practitioners as described by an early assessment of the practices in the field (Birk et al. 2003). A special edition (Sugumaran et al. 2006) in *Communications of the ACM* presented research papers situated around three aspects: Process; Organizational; and Technical. This thesis works on a higher level of Software Product-Line Engineering and looks at the overall approach instead of going into detail on one of these aspects. Therefore, we will not include in-depth descriptions. Software Product-Line Engineering is also classified as an approach to software reuse and some of its technologies are summarized in (Frakes & Kang 2005).

In software industry, the practitioners have tried to learn from each other, and the annual Software Product Lines Conference (SPLC)[1] draws a crowd of many practitioners and researchers from all over the world who share experiences. The Software Product Line Hall of Fame is usually presented at every conference and several case studies[2] describing successful Software Product-Line Engineering adoptions to serve as proof for the approach.

2.1.3 ***Practice Areas***

There are 29 practice areas in Software Product-Line Engineering which can be found in (Clements & Northrop 2002a). We introduce the reader to the various practice areas within each category and introduce three topics for each category to present the practice areas in a simple way (Table 1). We have chosen to use the framework (Northrop et al. 2007) as our reference on the practice areas since it is more up to date.

Table 1: Practice areas of Software Product-Line Engineering presented in topics divided on categories.

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

[1] More information at: http://splc.net/

[2] More information at: http://www.sei.cmu.edu/productlines/spl_case_studies.html

Software Engineering

Software engineering in a Software Product-Line Engineering approach involves both specific and traditional practices, to produce the software product line or the products we want to create. The fact that they handle both product line creation and products development make the practices two-dimensional, but we focus mostly on what is specific to Software Product Lines here and advise getting further information by reading the source. Design, development and testing are keywords for the practice areas highlighted in italic within the Software Engineering category which we have divided into three main topics, namely ‘Domain and Requirements’, ‘Architecture’, and ‘Development’.

<u>Domain and Requirements</u>

Understanding Relevant Domains is needed for understanding the commonality and variability in a Software Product Line. *“What are the problems and solutions within a domain?”* is an important question to be answered (Northrop et al. 2007). Information about what is a good product in the domain should be assessed continually in a Software Product Line.

Requirements Engineering is well-known in software engineering and describes what to produce and build. Software Product Lines require a set of products based on reuse of commonalities, and should be specified in that way. Requirements are also used for producing tests and leading implementations of the product line

<u>Architecture</u>

Architecture Definition involves creating the architecture of a software product line. Variations and efficient integration are both important for Software Product Lines. Core assets use architecture with variants, and applications employ this architecture with variant mechanisms triggered (Northrop et al. 2007). This practice area focuses on modeling variations and planning how the architecture can withstand the variations.

Architecture Evaluation’s goal is to ensure that the architecture definition is correct, based on the quality goals and requirements of the system (Northrop et al. 2007). Software Product Lines have variations to consider when evaluating the architecture. In single system development the Architecture Tradeoff Analysis Method (ATAM) (Bass et al. 2003) is a common method to evaluate software architecture.

Development

Component Development describes how the parts that constitute the architecture are developed. Software Product Lines need variation support in every component, and the components should be as loosely coupled as possible. When developing products these components should be the building blocks for the product at hand.

Mining Existing Assets involves using legacy code, existing artifacts or documentation as a guide for new development. Software Product Lines need support for reuse in these mined components, meaning that we have to use refactoring for reuse and add variation support.

Software System Integration means combining components as a whole. Continuous integration is recommended as the practice to use in this practice area (Northrop et al. 2007). Integration is described to get better with the increasing amount of products produced. In Software Product Lines, interfaces have a lot of influence on integration and product development is mostly integration work.

Testing the software could mean both validating and finding errors in the code, or performance testing. Software Product Lines need to think about testing for variations, and should try to reuse tests if possible. This means that tests have to be written for reuse.

Using Externally Available Software presents the use of COTS, open source, and freeware software for the core asset base or a component in a product (Northrop et al. 2007). Software Product Lines need support for the variability, and should analyze the fit, advantages, and costs for using these kind of products or code.

Technical Management
In a software development organization there are always technical risks, and the complexity of software makes it hard to manage. The technical management category includes practice areas highlighted in italic which try to cope with these challenges. We have divided the category into three main topics, namely ‘Scope and Technology’, ‘Process’, and ‘Management’.

Scope and Technology

Tool Support describes how development can use CASE[3] tools to support development and manage progress. A variety of tools are usually used, but Software Product Lines have to have the ability to use multiple versions of the

[3] Computer-Aided Software Engineering

same artifacts making non-specific software hard to use. Automation support is also considered here.

Make/Buy/Mine/Commission describes the choices for how to obtain a component for the Software Product Line (Northrop et al. 2007). Software Product Lines have different constraints and strategies connected to this decision as to single-system development.

Scoping is a practice that should describe the characteristics and what is supposed to be ‘in’ and ‘out’ in respect to functionality in a set of systems.

<u>Process</u>

Measurement and Tracking efforts are done in order to see if they meet organizational goals. Informal qualitative and objective quantitative measures are important. Tracking core asset development, product development and management is necessary. Software Product Lines use somewhat different measures because of its multiple customized system nature, compared to single-system development (Northrop et al. 2007).

Process Discipline explains how an organization should define, follow, and improve processes. Software Product Lines need to consider consistency in their core asset base and high interaction between separate organizational entities.

<u>Management</u>

Configuration Management involves identification, maintaining, controlling, and measuring artifacts that can change during the development lifecycle (Northrop et al. 2007). Software Product Lines have a multi-dimensional challenge with configuration management since the core asset base is within all products and all products must be controlled and updated according to changes in this base. However, nowadays this is often done automatically concurrent with the system integration mentioned above.

Technical Planning is the project based planning of certain core assets or developing a product for a customer (Northrop et al. 2007). Using the software development method’s planning for core asset development and following the production plan from the core asset base makes an organization able to produce software products.

Technical Risk Management means identifying risks, what to do with them, and how to deal with them. Software Product Lines involve more products, so a greater effort towards risk assessment is advised (Northrop et al. 2007).

Organizational Management

The last category, Organizational Management, covers the business and organization related topics around Software Product-Line Engineering. Developing a long-term strategy for the organization and being able to direct efforts where it is needed are covered by these practice areas highlighted in italic. We present these divided into three main topics, namely ‘Environment’, ‘Indoctrination’, and ‘Organization’.

<u>Environment</u>

Building a Business Case verifies the business needs and opportunities in a Software Product Line.

Customer Interface Management is about handling the customer, regarding both expectations and requirements (Northrop et al. 2007). Future directions of the products could be created through listening to key customers.

Developing an Acquisition Strategy can be interesting if the organization plan to involve third parties in delivery of parts or components to the product line.

Market Analysis is supposed to handle factors that determine the success of a product line or a product in the marketplace. It serves input to the scoping and the business case practice as well. In addition it could find commonalities in the core asset base.

Technology Forecasting identifies trends and predicts future markets. This could be triggered by own tools, customer needs, requests or emerging technology.

<u>Indoctrination</u>

Launching and Institutionalizing handle the adoption of the Software Product-Line Engineering approach. Being able to exploit the benefits of the Software Product-Line Engineering approach throughout the organization is essential, but poses as an organizational change.

Training provides the skills and knowledge needed to perform software management and technical roles. It is important that developers and business people learn Software Product Line techniques and obtain a multi-view perspective of the organizations development process (Northrop et al. 2007) in order to achieve the advantages the approach capacitates.

Funding is described as a practice since initial development in Software Product-Line Engineering is usually described as a Big Design Up Front (BDUF) effort following the proactive approach (Northrop et al. 2007). Maintenance and product development are secondary costs, where eventually product development enable you to earn money in the long run (with a proactive approach).

Organization

Organizational Planning is planning at the organizational level. The goal of planning is more abstract, involving all projects, compared to technical planning.

Organizational Risk Management provides mechanisms for surfacing and managing risks that transcend, or are shared, across projects (Northrop et al. 2007). Seven principles are mentioned by SEI. In Software Product Lines there are many stakeholders which could result in a broad specter of views.

Structuring the Organization describes how the organization is divided into roles, responsibilities, and authority. Software Product Lines are not product-centric, and need different roles and responsibilities to end up with products and have a functional product line.

Operations involves how the business gets done. Policies and practices for the organization are described here. In Software Product Lines operations is often how a product line is functioning and a concept of operations document.

2.1.4 ***Patterns***

The field of Software Product Lines and Software Product-Line Engineering describe a set of patterns to follow when adopting the Software Product-Line Engineering approach. While the practice areas cover concrete problems and solutions to these, the patterns handle the interactions between several practice areas to solve larger problems with Software Product Lines adoption (Clements & Northrop 2002a). The authors further describe that each pattern has three elements:

1. a context which is the organizational situation;
2. a problem that describes what kind of product line effort required; and
3. a solution which is the grouping and relations of practice areas who solves the problem for that context.

These patterns can be seen as the way to attack a Software Product Line problem or opportunity by combining practice areas, but a pattern does not cover the whole life cycle of the approach and sometimes several patterns should be used together. We have listed the patterns with variations (Table 2), and will briefly describe the patterns we are to use later. Interested readers are advised to read more in (Clements & Northrop 2002a).

Table 2: Patterns for Software Product-Line Engineering adoption. (Clements & Northrop 2002a)

Pattern	Variants
Assembly Line	
Cold Start	Warn Start
Curriculum	
Each Asset	Each Asset Apprentice
	Evolve Each Asset
Essential Coverage	
Factory	
In Motion	
Monitor	
Process	Process Improvement
Product Builder	Product Gen
Product Parts	Green Field
	Barren Field
	Plowed Field
What To Build	Analysis
	Forced March

Each Asset describes the situation when a skillful employee(s) in an area is to develop an asset that is planned through using the practices to increase the quality of the asset (Clements & Northrop 2002a). We mention every practice area involved in this pattern (Figure 1) without explaining the dynamics in detail. The practice areas all work iteratively around the production of a new asset for the platform. Two variations of this pattern are also identified in the source: a) *Each Asset Apprentice*, where the person in charge of developing the asset lacks experience in the area and 'Training' has to be included as a practice area; and b) *Evolve Each Asset*, means enhancing or changing an asset instead of developing.

What to Build is a pattern where the products that should be included in the product line are decided based on the domain(s) belonging to the product line (Clements & Northrop 2002a). 'Market Analysis' and 'Technology Forecasting' gives input to 'Scoping' and 'Building a Business Case' which again works iteratively over the results.

'Understanding Relevant Domains' also interacts with scoping. All together this results in a product line scope and a business case for this scope. Here there are also two variants: a) *Analysis,* which is a broader pattern where implementation specific details like requirements and architecture are involved; and b) *Forced March,* which uses legacy systems as a type of market analysis in order to elaborate the scope.

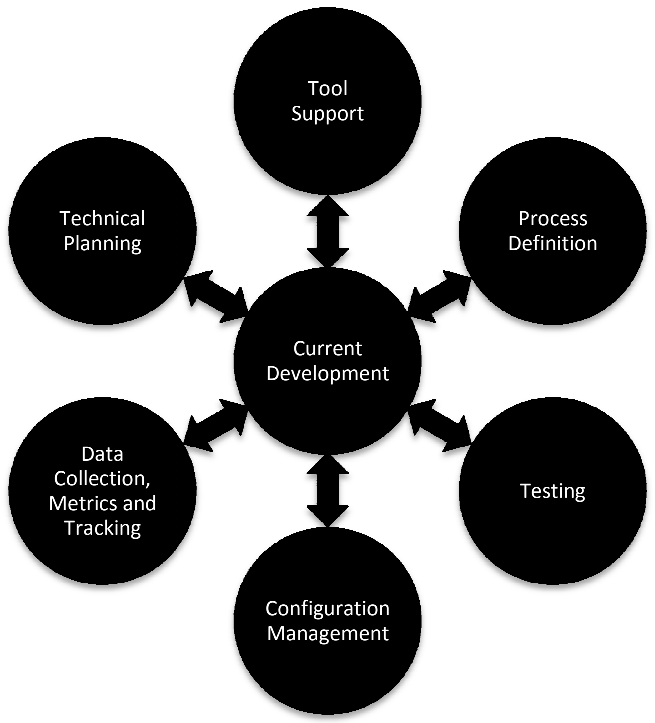

Figure 1: Practice areas involved in the Each Asset pattern of Software Product-Line Engineering. (Clements & Northrop 2002a)

Product Parts combines both practice areas and a pattern to develop core assets that are joined to be used in the products of the product line (Clements & Northrop 2002a). The 'Each Asset' pattern forms an outline for the combination starting with requirements, then looking at architecture and evaluation. After architecture is established, the components have to be realized through several practice areas, before it is tested and integrated. These steps will also provide feedback back to the originator of the data. Three variants for this pattern are described (Clements & Northrop 2002a): a) *Green Field,* where the context changes to no experience and has to develop or buy all assets where both 'Mining Existing Assets' and 'Acquisition Strategy' are removed; b) *Barren Field,* no scope established and 'What to Build Pattern' is combined with 'Green Field'; and c) *Plowed Field,* the opposite of the two other variants, meaning we try to use existing software as much as possible, resulting in a mining based effort to build parts.

Assembly Line describes the practice areas needed to build a capability of production based on an adoption of Software Product-Line Engineering (Clements & Northrop 2002a). The pattern combines 'Configuration Management' and 'Process Definition' with 'Tool Support' to establish a tool, and then let 'Organizational Planning' with input from 'Operations' give the final input to technical planning who realizes the final products.

Monitor has the responsibility of measuring and maintaining the course and operation of an established, running product line. Clements and Northrop (2002a) divides the practice areas for this pattern in two groups with the following practice areas:

- Listen: ‘Data Collection, Metrics, and Tracking’; ‘Technical Risk Management’; ‘Organizational Risk Management’; and ‘Customer Interface Management’.
- Response: ‘Technical Planning’; ‘Organizational Planning’; and ‘Process Definition’.

In this pattern the ‘Listen’ group provides feedback to the ‘Response’ group who enhance the plans and processes in the product line.

Product Builder is the pattern where products are realized through the established product line and the practice areas required (Clements & Northrop 2002a). ‘Requirements Engineering’ lead to ‘Architecture Definition and Evaluation’ in addition to giving input to ‘Testing’. ‘Component Development’ is done to implement the product, while ‘Software System Integration’ assembles components for the end product. A variant of this pattern is *Product Gen* where configurations are used instead of variations in code, and products are built through parameters based on requirements and an automated production process (integration) before it is tested.

Cold Start is the set practice areas that should be used the first time an organization launches a Software Product-Line Engineering effort to communicate the changes in culture and establish the new work practices (ref figure). A variant of this pattern is the *Warm Start* which describes an organization already established one or more product lines and the practice areas ‘Funding’, ‘Organizational Planning’, and ‘Operations’ are menttioned for this variant (Clements & Northrop 2002a).

In Motion is a pattern for a launched product line where the practice areas ensure the progress of Software Product-Line Engineering (Figure 2). ‘Funding’ and ‘Operations’ provide directions to ‘Training’, ‘Customer Interface Management’, and ‘Developing an Acquisition Strategy’. In addition, ‘Structuring the Organization’ interacts with ‘Operations’ to provide feedback on the directed efforts described above.

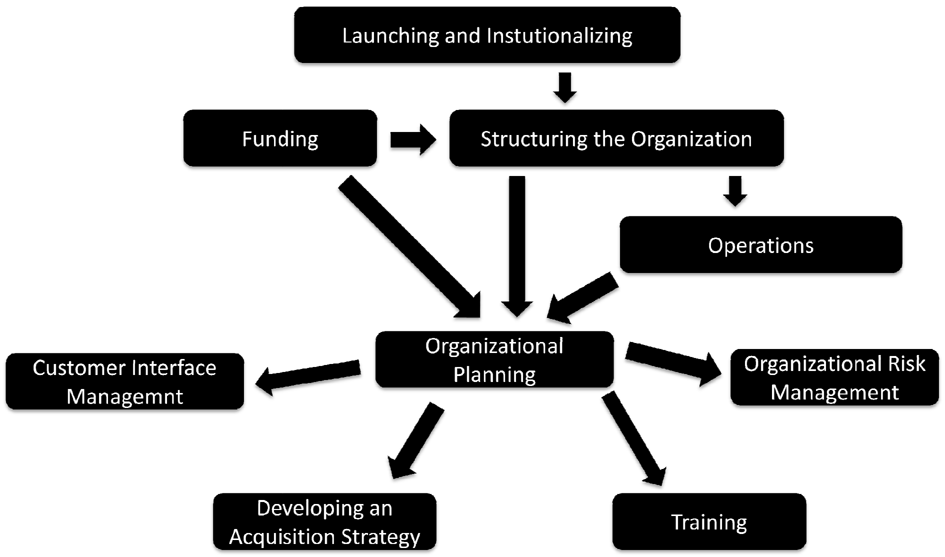
Launching and Instutionalizing
Funding
Structuring the Organization
Operations
Organizational Planning
Customer Interface Managemnt
Organizational Risk Management
Developing an Acquisition Strategy
Training

Figure 2: The In Motion pattern of Software Product-Line Engineering. (Clements & Northrop 2002a)

Process describes the practice areas that could be used for building and maintaining processes in the Software Product-Line Engineering effort (Figure 3).

Process Improvement is a variant of this pattern where the product line is running and the 'Monitor' pattern and 'Launching and Institutionalizing' are included in the effort of the pattern.

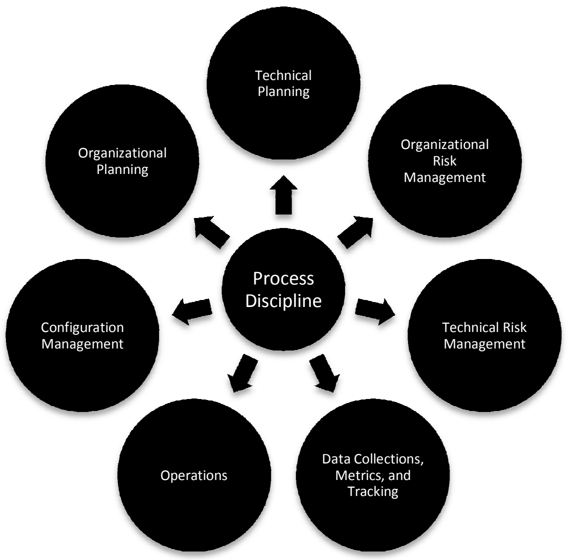
Technical Planning
Organizational Planning
Organizational Risk Management
Process Discipline
Configuration Management
Technical Risk Management
Operations
Data Collections, Metrics, and Tracking

Figure 3: The Process pattern of Software Product-Line Engineering. (Clements & Northrop 2002a)

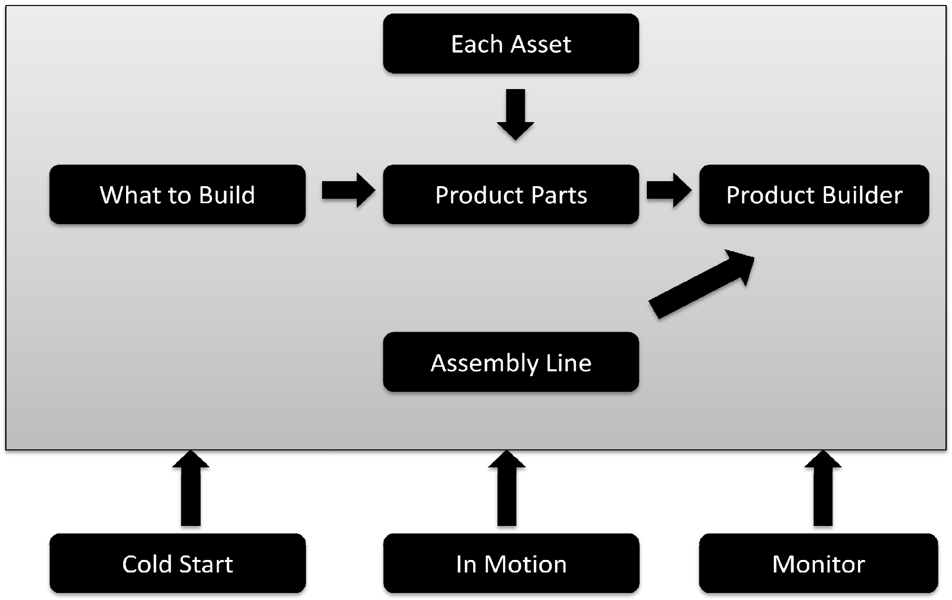

Figure 4: The Factory pattern of Software Product-Line Engineering. (Clements & Northrop 2002a)

Factory is the last pattern established and handles the overview of the product line effort for organizations evaluating a software product line and need to assess the complete effort of a product line (Figure 4) (Clements & Northrop 2002a). Clements et. al. (2006) describes this pattern very well in their article and connects it to the practice areas involved.

2.2 Agile Software Development

Agile software development is an approach to software development that has evolved from rapid application development and early spiral models (Larman & Basili 2003). Today the popular methods[4] of this approach are indicated to be Scrum and eXtreme Programming (XP) including the hybrid of these two as well, and we also choose to include Lean software development.

2.2.1 *Definition*

Since this field is still young, no uniform definition of agile software development exists. Our research found that Conboy's definition of agility is the most complete since this definition is a result of a broad literature review on the meaning of agile, lean, and flexibility not only in software engineering, but also in manufacturing and business.

[4] Based on results from VersionOne's 3rd Annual Survey: 2008 "The State of Agile Development" (found here: http://pm.versionone.com/whitepaper_AgileSurvey2008.html)

The definition states agility as:

> *"the continual readiness of an entity to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while maximising value, through its collective components and its relationships with its environment".* (Conboy 2008)

The early practitioners of Agile Software Development have made their definition based on thoughts and experiences in software engineering, and created the "Manifesto for Agile Software Development" [5] (Manifesto) consisting of four main ideas:

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

The main difference between these two definitions is the environment they are handling. Conboy's definition could be identified as a definition to evaluate agile methods and describe agility, whereas the agile manifesto was created as a guideline for followers of this software engineering approach. Still we find similarities in the way both definitions handle change, and handling the environment or customers. The Manifesto can be seen as a more practical approach, whereas Conboy's definition is a more abstract description.

2.2.2 *Status*

Agile Software Development as a research field still has many questions to be answered, according to the shift from waterfall development to iterative development as discussed in (Rajlich 2006). A couple of studies summarize what we know about the principles and practices that have emerged from the methods used in the industry that apply various views of the comparison (Abrahamsson et al. 2002; Cohen et al. 2004; Erickson et al. 2005). Another study tries to use empirical data to summarize the field's empirical data and generalize a status (Dybå & Dingsøyr 2008). Few generalized conclusions have been reached by the research field, but the industry seems to embrace this new paradigm and is adopting it more and more frequently in their software development projects. Dingsøyr et al. (2008) indicates that the industry is in front of research when it comes to Agile Software Development and therefore empirical studies involving industry could be the right way to learn more about the field. Agile Software Development has been adopted by industry, especially in dynamic environments where the demand constantly changes. The need for

[5] Can be found at http://agilemanifesto.org/

empirical studies of the industry is increasingly important since practitioners are leading the evolution of Agile Software Development (Dingsøyr et al. 2008).

The methods mostly incorporated in the industry, based on a recent survey[6], are Scrum and XP. The research up to 2005 was mostly handling XP (Dybå & Dingsøyr 2008), but Scrum is gaining popularity and a quick search on 'Scrum' in ACM[7] gives us 344 publications. A current trend, looking at the latest Agile Software Development conferences [8] , is Lean Software Development which can be described as a step further towards organizational agility adjusting the Toyota Manufacturing System to software development (Poppendieck & Poppendieck 2006).

2.2.3 ***Practices***

The various practices in Agile Software Development are not presented in a uniform way in the existing research and we found them hard to present in a good way. We try to categorize the practices we found common in the field into the same three categories we used for Software Product-Line Engineering: Software Engineering; Technical Management; and Organizational Management. This will make it easier for the reader to follow our line through this thesis and make it easier for us to reason about our findings in a textual framework. The negative point about this is that some of the practices might be understood to span over two or all three of our categories. We handle this by presenting the practice in the category we find most suitable, sorted in topics (Table 3). We based the collection of practices on especially (Abrahamsson et al. 2002), (Leffingwell 2007) and (Poppendieck & Poppendieck 2006). We also try to present which agile methods they are found in..

Software Engineering

Software engineering in Agile Software Development involves practices to produce the software or the product we want to create, usually a single software development effort. "Just enough" design, development techniques and continuous testing are keywords for the practices within the Software Engineering category.

[6] "The State of Agile Development"
(found here: http://pm.versionone.com/whitepaper_AgileSurvey2008.html)
[7] Search in "The Guide" at http://portal.acm.org
[8] As an example XP2009 (http://www2.xp2009.org/xp2009/)

Table 3: Practices of Agile Software Development presented in topics divided on categories.

Software Engineering	**Technical Management**	**Organizational Management**
Domain and Requirements:	Scope and Technology:	Environment:
Vision	*Automation*	*On-site customer*
Roadmap		
Elaboration		
Planning and prioritization		
Architecture:	Process:	Indoctrination:
Domain-Driven design and initial architecture with architectural runway	*Iterative development*	*Continuous improvement*
	Small, frequent releases	
	Iteration backlog, task board or Kanban	
Development:	Management:	Organization:
Test-Driven Development	*Visible charts and information radiators*	*Cross-functional and self-organizing teams*
Refactoring	*Configuration control*	*Stand-Up meeting*
Code ownership		*Metrics*
Component development teams		
CRC cards and design by contract		
System metaphor and coding standard		
Pair-programming		

Domain and Requirements

Vision involves creating a statement or guideline for the products that are supposed to be developed, including specific needs and requirements. The vision should be clear, broad, simple, and communicated throughout the organization (Leffingwell 2007).

Roadmap describes an approximation of dates within a reasonable timeframe where important information about releases is gathered (Leffingwell 2007).

Elaboration presents a description of the important requirements for the due iteration. It could be use cases, epics and user stories, scenarios, or acceptance test cases to help the creation of valuable software. Usually a product backlog or something similar is used as an artifact to hold these requirements (Leffingwell 2007).

Planning and prioritization can be planning poker or the planning game which plan and prioritize the requirements from a backlog (Abrahamsson et al. 2002). A backlog is commonly a list of requirements a product should obtain. The practice serves as a guideline for developers and as a requirements document for the customer. This can be done on the iteration level to plan and weigh efforts. This practice originates from Scrum.

Architecture

Domain-Driven design (Abrahamsson et al. 2002) and initial architecture with architectural runway (Leffingwell 2007) are practices that make the organization speak the same language between business and developers. There is also a need for a small model or plan for architecture of scale, preferable working examples of code on how the architecture is used. The practice is also mentioned as design spikes or evolutionary design in other sources. This practice originates from Feature-Driven Development and Lean uses it as well (Mehta et al. 2008).

Development

Test-Driven Development presents a practice where you write unit tests before you code, and then run the code until you pass the test. The unit tests also create a test suite which could be run as a whole to ensure component or system quality. This describes the extreme form which is hard to obtain, many companies do a more parallel version meaning that they start coding a little then write a test when it is clear what they need to test (Abrahamsson et al. 2002). This practice is found in XP, Scrum and Lean.

Refactoring is a practice which is usually a part of Test-Driven Development, but could also be used alone. Improving the code for flexibility and sustainability or reducing the complexity can be obtained through this practice (Abrahamsson et al. 2002). This practice originates from XP.

Code ownership involves the team and describes sharing responsibility for source code. In short, everyone can change everything. The team is also accountable for their code. This practice originates from XP and Scrum.

Component development teams practice means that one team defines, builds and tests each story (Leffingwell 2007). By that we mean that teams should have responsibility for the task at hand until it is completely finished. We also discuss teams more under the ‘Organizational Management’ category.

CRC cards and design by contract are practices to discover dependencies between classes and a way of risk reduction and handling risks (Leffingwell 2007). This practice originates from XP.

System metaphor and coding standard (Abrahamsson et al. 2002) is important since code is supposed to work as documentation. Therefore, clear guidelines and rules need to be established for coding. This practice originates from XP.

Pair-programming means programming in pairs to increase quality and creativity (Abrahamsson et al. 2002). Pair programming is also used for code review in Agile Software Development. This practice originates from XP.

Technical Management

In the Technical Management category the practices that try to cope with technical risks and challenges are explained. Agile Software Development handles this through risk reduction practices. These practices are sorted into three topics underneath, namely 'Scope and Technology', 'Process', and 'Management'.

Scope and Technology

Automation as a practice means daily builds, continuous integration, and automatic test procedures. In more detail, to have an automated build and test process that runs as often as possible and does continuous integration can be beneficial to help keep focus on quality at all (Poppendieck & Poppendieck 2006). Integration test and automated system testing keeps the quality focus. Automation is advised especially in Lean.

Process

Iterative development with lightweight up-front planning, focused development, and iteration demo are commonly used in Agile Software Development (Abrahamsson et al. 2002). Teams plan in iterations, further tasks are created and estimations done before an iteration start. Then development is done with time boxing which results in showing the result of the iteration typically to a customer or a stakeholder.

Small, frequent releases are used to see progress with the product and build value to the customer incremental instead of a big bang release (Leffingwell 2007). The practice is also described to reduce risk, and makes the teams more able to change direction according to the customers need at the moment.

Iteration backlog, task board or Kanban serve as an activity schedule for the component teams in an agile project. This practice originates from Scrum and Lean. *Time boxing* is done to be able to facilitate estimation and metrics. The meaning is to work effectively and directed without having to use overtime work because it slows people down (Leffingwell 2007).

Management

Visible charts and information radiators are used to describe the teams' effort and what they do. People that pass by can see this information and obtain the status of the teams without interfering with their work (Poppendieck & Poppendieck 2006). Product backlog and Kanban can be used as artifacts for this and good practices should also be visible. A Kanban is a task control board originating from Toyota Lean Manufacturing. This practice originates from Lean and Scrum.

Configuration control as a practice is somewhat contradictive to code ownership, but could be seen as important when you have several teams and they need to be coordinated. According to Leffingwell (2007) this can be handled by assigning component teams, meaning that each team has their own component to develop trying to separate the development effort to reduce simultaneous code changes.

Organizational Management

The last category, organizational management, covers the organizational topics around Agile Software Development. To create an organization that supports Agile Software Development means changing work processes and shifting focus to short-term planning and controlling. The practices we found viable are sorted into three topics, namely 'Environment', 'Indoctrination', and 'Organization'.

Environment

On-site customer is used to answer questions regarding the product which the team is to deliver, and to solve impediments for the team (Abrahamsson et al. 2002). This practice originates from XP.

Indoctrination

Continuous improvement can be reflections, root cause analysis, and retrospectives to evaluate and redirect the process after each iteration. This practice is found in Scrum and in Lean as Kaizen.

Organization

Cross-functional and self-organizing teams are encouraged in Agile Software Development. The teams should cover the whole component as described above, and be self-organizing to support creativity and effectiveness. The teams can use task switching and should be collocated. This means that the development organization should be split into smaller multifunctional, self-organizing teams that develop on one component at a time (Leffingwell 2007). Team empowerment is also mentioned in some literature and means that the team is trusted and that creativity is supported.

Stand-Up meeting can be preformed every day, and can be scaled up to other meetings for broader control (Leffingwell 2007). This gives the developers an overview of what the team did, will do, and if there are any problems slowing down the work. Stand-up meeting is an easy approach to share information and cooperate in an effective team, tasks can also be assigned and problems shortly discussed or decided what to do with.

Metrics are kept for team velocity, code measurements, test coverage, product progress, and quality (Poppendieck & Poppendieck 2006). Measures could follow code, requirements, and create artifacts such as charts and plans to control and report about progress in an agile project (Abrahamsson et al. 2002). This practice is found in XP, Scrum, and Lean.

2.3 Agile Software Product-Line Engineering

Normally Software Product-Line Engineering is seen as an approach where an organization can use existing market knowledge to grasp a larger share of the market through exploiting commonalities, best practices, and domain knowledge in this market. This assumes that the domain is relatively stable and have developed a share of best practices and a common “know how“-standard among the customers. Agile Software Development on the other hand can be described to cover the opposite of this, being able to respond in changing markets and do development in markets that are not well understood.

2.3.1 ***Definition***

Few definitions of Agile Software Product-Line Engineering exist. During our research I have encountered only one partial definition on the combination:

> *“making product lines more responsive to ever changing customer needs or market developments”* (Noor et al. 2008)

Regarding the definition of agility by Conboy, we clearly see that the definition above has some shortcomings regarding the environments surrounding them and learning from change, which is not clearly stated in the definition. In addition to this we can identify shortcomings when comparing the definition to the agile manifesto as well, because this definition only mentions customer collaboration and response to change. The definition lacks connection to individuals, interactions and working software as stated by the Agile Manifesto.

2.3.2 *Status*

In our previous study (Gylterud 2008) we investigated the motivation, goal, similarities, and benefits. The motivation and goal could be the ability to respond to changing domains with the benefits of reuse and shorter time to market. Market potential in dynamic domains is often bigger than in completely stable domains (Schilling 2004). Aligning Agile Software Development for large scale production with strategy and business needs covered remains an important challenge for the future as described in the FLEXI Newsletter[9]. Kettunen (2008) looked at agile manufacturing and agility for new software development, and found that Agile Software Development could cope with a project's immediate needs and aspects without considering long-term planning and large-scale organizational challenges.

2.3.3 *Practices*

The main difference between these two approaches to software engineering is the situation it is meant for:

- Software Product-Line Engineering tries to mass produce software to many customer through a platform where variants open for customization;
- Agile Software Development tries to maximize the value for a single system through focusing on people, code, changes, and customer interaction.

The practical differences are found to concern documentation, requirements, architectural focus, up-front investment, customer interaction, and quality assurance (Gylterud 2008; Tian & Cooper 2006). Ghanam (2008) also indicates challenges with combining agility and Software Product-Line Engineering, but in the view of Test-Driven Development on amore technical level. Further analysis shows that Software Product-Line Engineering can work on team and creativity challenges, in addition to adopting agile practices to become more agile.

[9] Fourth Issue (No 1, 2009), downloadable here:
http://www.flexi-itea2.org/download/FLEXINewsletter_09a.pdf

3. Research Method

In this thesis we chose to apply a hybrid research method consisting of both a case study analysis and a set of semi-structured interviews. Since our research field has little empirical research we chose to perform the study like this, trying to cover some of the existing research from the industry and combine it with new discoveries from interviews. This chapter will describe the study and its different parts. First, we used a search strategy and summarized the case studies we found. Then we established criteria for which case studies to choose based on our perceptions of what is important in the fields. We also performed interviews with some practitioners in the industry to support and discover new practices. Based on the criteria we chose six case studies and included the three interviews and performed coding. This resulted in sorted data based on themes that made us able to document our results and discuss our problem further. The chapter is divided into two main sections, *3.1 Data Gathering* and *3.2 Data Analysis,* which will elaborate on the method we just explained.

3.1 Data Gathering

In order to answer RQ-1.1 about *which studies handle both agility and software product line engineering,* we gathered data from studies that were combining Software Product-Line Engineering and Agile Software Development in the existing literature, and decided to perform interviews with companies to further state our answer to the research question. This was a difficult task because no uniform description or definition of Agile Software Product-Line Engineering exists.

3.1.1 ***Existing Articles***

We did a comprehensive search in our literature review autumn 2008 and we already had found three of the articles used in this thesis. We also went through various sources with different strategies (Table 4) to investigate whether we could find Agile Software Product-Line Engineering case studies or experience reports. The results were small and the same studies were identified in several sources.

With the result from the search we identified 14 different case studies chosen to be closer investigated, by reading titles and abstract for further investigation. A table describing the different articles was established, and formed a basis for our preliminary choices of which case studies to analyze (Appendix I).

Table 4: Search strategy and results for different sources of information

Where	What	How	Results
SEI DoD Workshop (http://www.sei.cmu.edu/productlines/pub_by_topic.html#dod_workshops)	Summaries from workshops on Software Product Lines in defense systems	Search by browsing and reading abstracts	2
http://softwareproductlines.com/successes/successes.html	Short stories about Software Product Line Engineering implementations	Search by browsing and reading abstracts	2
SEI Hall of Fame (http://www.sei.cmu.edu/productlines/plp_hof.html)	Various company stories from implementing Software Product Line Engineering	Investigated the references listed on the interesting companies.	3
ISI web of knowledge	Published articles	Search terms: software product line and case; software product family and case	107
ACM portal	Published articles	Search term: (software product lines OR software product families) AND case study refined by case studies	468
IEEE	Published articles	Search terms: software product line and case; software product family and case	106

… Continued Table 4

Where	What	How	Results
ESAPS (http://www.esi.es/Projects/Esaps/)	Engineering Software Architectures, Processes and Platforms for System-Families Project	Search by browsing in the 'Public Results' section.	0
CAFÉ (http://www.esi.es/Cafe/)	From Concepts to Application in System-Family Engineering Project	Search by browsing in the 'Public Results' and 'Dissemination' sections.	1
FAMILIES (http://www.esi.es/Projects/Families/)	FAct-based Maturity through Institutionalisation Lessons-learned and Involved Exploration of System-family engineering Project	Search by browsing in the 'Public Results' and 'Dissemination' sections.	2
Agile Conference	Proceedings (IEEE)	Search by browsing for case studies that could fit our criteria.	3

RQ-1.2 and RQ-1.3, respectively *which criteria are needed to determine Software Product-Line Engineering and agility* and *how should the weight of each criteria be to choose the best cases*, were answered by establishing a criteria for the case studies, before our analysis started. For the case analysis we decided to create a set of criteria for both the Software Product-Line Engineering aspects (Table 5) and the Agile Software Development aspects (Table 6) we found important.

Table 5: Criteria with weight for choosing articles in the Software Product-Line Engineering field.

Software Product-Line Engineering

Element of the approach	*Weight*	*Explanation*
Set of software systems Many software products or embedded systems that are delivered to many customer with different specifications	Critical	Economics of scale is the base idea. A line of products with ability to cover the needs of many customers.
Common set of features The features of the software are similar and share commonalities, but have variations and diversity which make components reusable throughout many products.	Important	The feature thinking is not as important as many products and customers, but serves as an enabler to obtain economic benefits so it cannot be neglected in a Software Product Line approach. One has to think in advance on what features to cover and support.
Handling needs in a domain Serves as a solution to many challenges and follow best practices in defined domains to ensure quality to and interest for the products.	Nice	Domain knowledge is the least important criterion, but domain thinking needs to be established to succeed.
Use a core asset base A core asset base or base of components are used as a basis to develop or configuration specific software products to customers	Critical	Important because of the nature of software product lines relies on the core asset base. This is the reuse "center", but does not need to function perfectly. The core asset base is a prerequisite for SPLE.
Systematic, planned reuse Strategic thinking in long scale in addition to product line thinking throughout the organization.	Important	The strategic nature of PLs makes this an important measure, but not as much as core asset base and set of products because the maturity of this effort might not be present yet. Therefore we chose to weigh this criteria lighter.

Table 6: Criteria with weight for choosing articles in Agile Software Development field.

Agile Software Development		
Element of the approach	*Weight*	*Explanation*
Listen to change from environment Internal (team), immediate (organization) and general sources of change (competitor, customer or supplier). How do the companies handle this kind of change and what kind of system is practiced to listen?	Critical	Especially customer change is important, but also other environmental changes affect the agility of a development process.
Allow change in components (refactoring?) Creativity, experimenting, autonomy, refactoring, diversity, observation, and challenging work. How are these ideas handled in the company?	Nice	Not the most important aspect of agile, but could improve to get better after a while. We chose not to put too much weight on this criteria because of its relations to management and the future prospective to improvement.
Proactively work towards change Risk identification and estimation. How are risks assessed by the company?	Nice	Most companies does handle risk some way. For agility it is not the most important aspect, and could also be future improvements. We chose to not weigh it that hard since it is something that could evolve during reaction to and learning from change.
Reactively allow change Identify, resolve, and react to change	Critical	The ability to handle changes is one of the more important aspects of agility and we chose to weigh it accordingly. The thinking and support for the ability to change is critical to agility.
Learning from change Handle information between teams, learning within teams both planned and unplanned, and communication information	Nice	Learning is less prioritized by us. This is also one of the aspects that could improve after a while. It does not need to be fully established in our case studies
Maximize value Deliver the right software to the right time without overhead. Just enough emphasis. Time to market and speed handling.	Important	Maximizing value is also important. We wanted to emphasize this point since it involves some agility principles that companies needs to think about and that we think are important.

Discussions lead to this weighed set of criteria based on our knowledge in the respective fields. This was done to be able to choose between the case studies identified. The criteria are weighted base on what we found to be:

- Critical, the things that have to be in the approach;
- Important, the things that should be in the approach;
- Nice, the things that are nice or could be in the approach.

The criteria helped us evaluate each of the cases in respect of both Software Product-Line Engineering and Agile Software Development. We went through each of the cases from the literature search and validated them against these criteria. That is how we were able to choose which cases to include in our study.

Six case studies in literature were chosen and two case studies based on interviews were used for this thesis (Table 7). Since the research and experience from this field is limited we chose to include two case studies that do not cover the whole Software Product-Line Engineering field, but investigate individual practices for improvement and enhancement as well. We included them to describe some of the emerging practices in the field and to describe the possibility to combine Software Product-Line Engineering and Agile Software Development.

3.1.2 *Interviews*

We also wanted to perform a set of interviews with the case study companies and other interested parties to support our findings, discover other practices and validate our study. The interviews bring an extra dimension to RQ-1.1, and could further increase validity in our qualitative study. The natural contact point was the authors of the case studies and influential researchers in the area. We also decided to actively request for participation on several communities on the Internet (Table 8) (Appendix II). Of the authors we contacted, only one responded and in the communities the response was not as we hoped for, in fact no response was triggered. We might have used the wrong communities or wrong instances, but we felt we did a qualified try. Further, two industry contacts were established through our network:

- Industrial Financial Systems (IFS);
- and Det Norske Veritas (DNV).

These companies were able to give us some introductory information, before they decided to join our interviews to give us new data on their organizational efforts towards software product development.

Table 7: Chosen case studies with the matching criteria they hold.

Case	**Criteria**										
	Software Product Lines					Agile Software Development					
	Set of software systems	Common set of features	Systematic, planned reuse	Use a core asset base	Handle needs in a domain	Listen to change	Allow change	Maximize value	Learn from change	Allow change	Work towards change
CompNN	X			X	X	X	X	X		X	X
Salion	X	X	X	X		X				X	
HomeAway	X	X	X	X	X	X	X	X		X	
IFS	X	X		X	X	X	X	X		X	
DNV Software	X	X	X	X		X	X			X	
Engenio	X		X	X	X					X	
Testo AG	X		X	X						X	
PROSOL	X			X		X		X		X	

Table 8: Communities where request was sent and their size.

Name	Where	Members
agilemodeling	Yahoo groups	1690
leanagile	Yahoo groups	1380
SW-improve	Yahoo groups	252
Agile Alliance	LinkedIn	6896
agile-research	Google groups	62
Agile Product Line Engineering	Google groups	9
all about agile	Google groups	511
Forum smidig.no	Smidig.no	N / A

All together this gave us the opportunity to perform three interviews with various people in the industry. We mean they cover a representative part to be able to support our analysis findings and discover more practices, and help us reason about combining Software Product-Line Engineering and Agile Software Development.

Our interviews were performed as semi-structured interviews, following the guidelines in (Seaman 1999), with a length of around 20 to 30 minutes. Further, we established a situational interview guide (Appendix III), one for each interview, for leading the interviews to collect as much useful information as possible. The essence of the guide was pretty much the same, but we tried to exploit the material we already investigated to avoid overlap and find as much new information as possible.

The interviews were performed using Skype[10], a Peer-2-Peer Voice over IP program that allows people to talk with each other for free. To capture the interviews as audio we used an add-in called Skype Call Recorder[11], which made a good quality mp3-file of the interviews. In order to transcribe these audio files to documents for use in our analysis we used Express Scribe[12] who were able to slow the audio down a little making it easier to write down exactly what was said. As soon as transcription of the interviews was done we assessed the results of the interview quickly and found follow-up questions that could further strengthen our evidence. These were sent to the respective interview candidate and we received answers to those questions as well. These answers were included in the transcription of the respective interview.

3.1.3 *Data Quality*

We also assessed the validity of each case regarding research methods, threats, and measures obtained in the articles to be able to reason about the studies in general. We chose not to take this into account while deciding on the case studies to include in this thesis since the available data in this field is so small. We tried to limit the studies to peer reviewed articles like journals and conferences, but where the material was limited we added non-reviewed articles to make the data more complete. The fields we investigated in this study are by no means highly mature and our options regarding data were very limited.

[10] For more information: http://www.skype.com
[11] For more information: http://www.voipcallrecording.com/Skype_Call_Recorder
[12] For more information: http://www.nch.com.au/scribe/

3.2 Data Analysis

To be able to answer both RQ-1.4 and RQ-1.5, respectively *how agility is introduced and works in the industry* and *which practices are used in industry,* through analysis we had to choose between the identified case studies and assess the level of Software Product-Line Engineering and agility. The analysis was performed with marking themes and comparing the cases, which are further elaborated below.

3.2.1 ***Analysis Technique***

After choosing the appropriate cases we started coding our data following the guidelines of (Seaman 1999). We did this in an iterative fashion meaning that our first iterations went through the text roughly and marked the larger topics. Then we divided the larger topics into themes for which we marked in the following iterations. These themes had to concur with the criteria we made for choosing as well, since these were our most important points for Software Product-Line Engineering and Agile Software Development, and we made a mapping table to ensure this (Table 9). After marking the text through several iterations we used Weft QDA [13], a qualitative text analysis software, to code the text into these themes and make a print sorted on themes. That left us with 50 pages with an average of about seven quotes per page, about 350 quotes. These quotes were cut separate and used in two different ways to document the results.

3.2.2 ***Analysis Result***

First, we sorted by case company and evaluated each case separately based on the criteria (3.3.1) and a template for the case descriptions (Appendix IV). Secondly, we sorted on the themes and crossed the different quotes to find similarities and differences in each theme. We documented what we found in the different practice areas of Software Product-Line Engineering and the practices of Agile Software Development. Using the analysis technique to sort and combine the cases made us able to more clearly see the themes and combine results from several cases, instead of using regular textual combination without coding and grouping (Appendix V). Documenting results were also easier since sorting and grouping quotes from the cases made a draft of result documentation by putting them together in the order we were going to document them.

Further we choose to collect feedback on the case descriptions from the interview candidates' respective cases, to ensure that our interpretation where aligned with their perceptions. In addition we sent the result chapter to the

[13] Can be downloaded from here: http://www.pressure.to/qda/

companies to ensure that the practice areas and practices were correctly documented and for additional ideas that they forgot to mention.

Table 9: Mapping of the analysis themes and the criteria for choosing articles.

Themes	**Subtopics**	**Criteria**										
		Software Product Lines					Agile Software Development					
		Set of software systems	Common set of features	System-atic, planned reuse	Use a core asset base	Handle needs in a domain	Listen to change	Allow change	Maximiz e value	Learn from change	Allow change	Work towards change
Context	Quantitative Information											
	Business aspects		X			X	X					X
Software Product Line	Software Product Line Practices	X			X							
	Approach to Software Product Line		X	X								
Agile	Agile practices						X	X	X		X	
	Action towards change			X	X			X		X	X	X
Software Life Cycle	Project Management Process			X			X		X			X
	Pre-Development Process					X	X					X
	Development Process		X	X	X	X	X	X		X	X	X
	Post-Development Process	X			X				X	X		
	Integral Process			X		X			X	X		
	Validity of study											

4. Practices in Industry: Product Line and Agile Software Engineering

RQ-1 asked *how agile software development principles and practices are used in combination with software product-line engineering in industry today*. To answer the question we chose to analyze several organizations through published articles and personal collected data from interviews. This chapter will present the results of this analysis in four sections. We start by outlining the case studies followed by a description of each case study we investigated (Section 4.1), using a uniform template. The goal of these descriptions is to explain enough details about the cases so that the reader can understand our findings without having to read the data supporting the case descriptions. The second section (Section 4.2) describes our results regarding Software Product-Line Engineering practice areas used in each of the cases which follow the same structure as we explained in (Section 2.1.3). The last section (Section 4.3) presents the practices of Agile Software Development used in the Software Product-Line Engineering approaches in the same structure as described in (Section 2.2.3). This structure allows us to get a good view on the cases and get into detail about the practices used in the software industry. Further, it supports our discussion when we combine the results into Agile Software Product-Line Engineering.

The case studies describe several companies whose domain varied, although they all share an opportunity to reap benefits from a Software Product-Line Engineering approach by individually delivering customized products to several customers (Table 10). The companies' business situations are all described as dynamic, with high demand customers and unstable requirements in their respective domains, and the need for processes that handle change is present. The size of the various companies can be described as small and medium enterprises (SME), with IFS being the biggest company. Five of the six full-scale companies (Salion excluded) mention distributed development efforts. The Software Product-Line Engineering backgrounds of the companies were different:

- four of the studies, CompNN, Testo, DNV Software, and IFS describe an existing approach and introduction of agility;
- the four other studies describe a concurrent effort towards Software Product-Line Engineering and Agile Software Development because of the high cost and large time consumption of a proactive approach to Software Product-Line Engineering.

Five of the eight studies described an intentional Software Product-Line Engineering approach while the CompNN, DNV Software, and IFS studies have clear similarities to Software Product-Line Engineering theory. The case study companies use either the reactive or the extractive approach in Software Product-Line Engineering. An existing platform or assets are used to develop the product line or create assets when a need or opportunity arises in several case studies. A tendency towards an incremental approach when transitioning to Software Product-Line Engineering was found in Engenio, PROSOL, Salion and HomeAway. Further Engenio, Salion and HomeAway used BigLever's Gears (Krueger 2001) and consultant services in order to establish their product line.

Table 10: Outline of the various cases we investigated.

Case	Size (employees)	Products	Reuse platform	Distributed environment	Approach to product line	Level of product line	Level of agile	References
CompNN	90	Market and Customer Survey	Modular	Yes	Reactive	Medium	High	(Hanssen & Fægri 2008)
Salion	21	Proposal and Bid Process	Components based	No	Reactive	High	Medium	(Buhrdorf et al. 2004; Clements & Northrop 2002b)
HomeAway	N / A	Web-based system for vacation homes	Variation based	No	Reactive	Medium	High	(Krueger et al. 2008; Rally Software 2008)
IFS	2600	Enterprise Resource Planning	Component based	Yes	Reactive	Medium	High	
DNV Software	160	Service applications	Modular	Yes	Extractive	Medium	Medium	(Linden et al. 2007)
Engenio	180	Storage	Core assets	Yes	Reactive	Medium	Low	(Hetrick et al. 2006)
Testo AG	N / A	Measurement Devices	N / A	N / A	Reactive	N / A	N / A	(Carbon et al. 2008)
PROSOL	N / A	Supply Chain Management	N / A	N / A	N / A	N / A	N / A	(Noor et al. 2008)

4.1 Presentation of Case Studies

Each case study was described based on a template which can be found in (Appendix IV). The template we followed made a framework for every case description starting with brief description the context of the company. We did this to create a similar structure for every described case to make the thesis become readable and uniform. Later, we researched on how Software Product-Line Engineering was instantiated and in more detail about the Software Product Line development. Next, we describe the agile practices found in the cases and finally we wrap up with the benefits versus disadvantages of the total approach, and assess the level of Software Product-Line Engineering and agility.

4.1.1 *CompNN*

CompNN, from the case study, delivers market and customer survey products in the high-end market with one major variation: standalone server solution, or hosted solution. The market for surveys is described as competitive and the need for changes is present at all times (Hanssen & Fægri 2008). About 90 employees spread over four countries are developing and selling the products of CompNN at the time the case study was written in 2007. The research and development departments of the company are situated in Norway and Vietnam, while the US and UK handles marketing, sales, and support which are in the biggest markets that provides customer feedback to development (Hanssen & Fægri 2008). CompNN is said to have two releases per year and works in small development teams.

The products CompNN deliver are customized surveys or survey tools for their customers which is based on a product platform, or core asset base, which is modular. Reuse is handled through license files that trigger each module and the respective configuration (Hanssen & Fægri 2008). The most common configurations are predefined, whereas variations for customers can be custom developed by CompNN. Custom development efforts should fit and be allowed for inclusion in the core asset base to be realized according to the company's scope. They also do system integration with their customers. As the request for CompNN's products has increased, they have moved more towards Software Product-Line Engineering, even though they have not followed known guidelines like (Northrop et al. 2007) or (Pohl et al. 2005). The case states the similarities with the principles known in this field, and our assessment according to our criteria coincides. An example can be the two different delivery strategies, and the module based platform in which modules are activated by the registration code of the customer. Their approach to Software Product-Line Engineering has evolved based on experience and needs over the last ten years, with weight on the strategic part of the approach (Hanssen & Fægri 2008). The strategic part includes: a Product Management team; a roadmap; a market process; product goals with metrics and business cases (works as requirements); and a Product Advisory Board with key stakeholders. These practices guide the development efforts on the core asset base.

Their software development process was long a waterfall process but the adoption of the EVO method (Gilb 2005), an iterative method with agile principles, was done to become more flexible and respond faster to changes (Hanssen & Fægri 2008). The freedom in this method is described to encourage creative problem solving, and involves short iterations and customer interaction. Impact Estimation Tables (IET) leads development of each project and provides both stakeholders and developers with the necessary overview to control the efforts of the team (Hanssen & Fægri 2008). In addition, stakeholders give feedback to developers after each iteration and product goals are used both as requirements and metrics for the development teams. Test-Driven Development is being adopted in the case, and another factor that increases the agility of the approach is encouragement of continuous learning provided by both EVO and Software Product Line Engineering (Hanssen & Fægri 2008). Continuous integration is another agile practice that is followed by CompNN and their architecture is described as simple, flexible, and open to change with the modular platform.

In addition to these practices, the operational handling is based on customer feedback with support and market analysis as the main practices. For quality assurance, a green week is described for improving sustainability and error correction in the core asset base. Benefits of combining Software Product-Line Engineering and Agile Software Development found in this study are: fast response to changing needs; precise delivery; and a rigid but agile framework for requirement specification. Challenges with the combined approach are mentioned in the case to involve high cost with the strategic, long-term process (Hanssen & Fægri 2008), getting the right customer interaction, and abstracting enough on the higher levels so that lower levels can creatively solve problems.

The ability to combine Software Product Line Engineering with agile methods like EVO is exemplified in this study. A disadvantage for us is that this approach to Software Product-Line Engineering did not directly follow the practices and guidelines in the field. The agility level of this approach can be stated as high based on the EVO approach and agile practices used, while the Software Product-Line Engineering level can be described as medium based on the core asset base and strategic planning efforts of the company.

4.1.2 *Salion*

Our second case study describes a startup company, Salion, focusing on the proposal and bid domain which targeted the automotive, custom manufacturing companies. The goal of Salion is described as being able to provide software to maximize the customers' business value. There were 21 people in the startup, with about 10 people working with product development when these articles were written (2002 and 2004). Salion was not experienced in this domain, but had some former knowledge about similar domains and research material from their owners (Clements & Northrop 2002b). The articles describe this as a new market with no former applications developed. The Software Product Line approach background were established based on the need to quickly deliver customized software, and agility elements was combined in Salion's described effort towards Software Product Line Engineering (Clements & Northrop 2002b).

Salion chose to use a reactive approach towards Software Product Line Engineering. The articles explained benefits like lower initial cost and design up-front with this approach. Salion combined a lightweight Rational Unified Process (RUP), VRAPS [14], and eXtreme Programming (XP) with SEI's Software Product Lines to get an agile Software Product Line Engineering approach. They also chose to use Gears (Krueger 2001) and CloneDR[15] as tool support, respectively for delivering products and refactoring commonalities. As for Software Product-Line Engineering practice areas, they started with what can be called lightweight scoping, described as an informal scope definition that evolved during product initiation for customers (Clements & Northrop 2002b). Salion uses COTS whenever possible, but customization and refactoring are also possibilities described if the need for components arises or enhancement is possible (Clements & Northrop 2002b). Documenting is also done based on components in the core asset base. The products (3 applications) obtained through the core assets is called the RAM platform, but each customers have their own needs which a team in Salion has the responsibility to implement (Buhrdorf et al. 2004). Salion was described to delivered solutions either hosted or installed in two phases with the standard system first and then customized based on individual needs.

In Salion's management effort, a series of meetings is described to support the process. The "Joyous vision" meeting is a high-level stakeholder meeting to choose a strategic direction, the "Joyous chunk" meeting ensures that customer needs are handled according to contract, and last the "Joyous love" meeting

[14] Vision, Rhythm, Anticipation, Partnering, and Simplification (VRAPS) is a model of the organizational aspects of creating and maintaining a software architecture.

[15] more on http://www.semdesigns.com/Products/Clone/

held two times a week discusses impediments for the production cycle (Clements & Northrop 2002b) much like the agile daily stand-ups. These meetings set the direction and handle risks in the company. Metrics are also involved in the management approach, in addition to projecting and planning the company's effort. The management involvement, architectural inspections, and ownership of the chief architect and product line champions are described to ensure that the product line approach evolve in a sustainable way. Continuous focus on quality is also a practice described in the articles. Further, the marketers of the solution provided are in charge of dealing with the customers' needs according to the scope, variability provided, and confirmed customizations.

Regarding change and agility of Salion's approach to Software Product Line Engineering a couple of practices are mentioned in the articles: frequent and automatic builds; test-driven development; contracts among units; and refactoring (Clements & Northrop 2002b). The architecture is also described as agile, because it is not limiting the company in their choices of variants. Customer involvement, a culture for change, and self-awareness are also mentioned in the articles and fit well in the Agile Software Development principles. Component ownership and the unified direction of the employees are also actions towards change and continuous improvement. Release is often obtained through a 30-day release plan to maximize the value. Lastly, a "stop the line" focus on conflicts described in the articles is also an indicator for the agility based on the theory in Lean.

A successful implementation and a running product line is described in this case study. Short introduction time and less up-front investment in both time and cost, are envisioned through metrics in both the articles (Buhrdorf et al. 2004; Clements & Northrop 2002b). Several incremental improvements and efficiency challenges solved underway is also described through metrics in the case. Overall challenges are not mentioned, but might be: overhead from frequent meetings; adjusting the RUP method lightweight enough; and refactoring for reuse.

We believe that the level of agility in this approach is medium, because elements of agile software development and agility are described. The Software Product-Line Engineering level is high because many practice areas are used, the core asset base is clearly defined, and the articles are presented in the research field.

4.1.3 *HomeAway*

Another company that combines Agile Software Development and Software Product Line Engineering, HomeAway, consists of several national vacation home rental companies which has web-based systems for both renting out and obtaining a vacation home. HomeAway's goal is described as *supporting "easy to use" web-based service to travelers and expanding the market with their products* (Krueger et al. 2008). The case studies describe multiple development teams, with distributed locations in the US and the UK.

HomeAway's 'Public Sites' product line (later referred to as 'Public Sites') is described in the interview to deliver basically the same product in every country, with certain language and special feature variations. First they tried to fit their product in a one-size fits all solution (Krueger et al. 2008). As the first solution grew bigger the amount of code and the complexity became uncontrollable, and the company, with support throughout the organization, decided to adopt Software Product-Line Engineering. A successful approach to Software Product-Line Engineering for the company could reap commonality benefits, increase quality and reduce time to market for enhancements (Krueger et al. 2008).

The company chose to use Krueger's 3-Tiered Software Product Line Methodology and the Gears tool (Krueger 2001) to establish their product line. The methodology presents three tiers of Software Product-Line Engineering, focusing in sequence on variations and automation, then core assets and finishing with portfolio evolution. These tiers have dependencies, but were done partly in parallel by the company. In general we could describe their effort with the reactive approach towards Software Product-Line Engineering. HomeAway used their old platform as input to their new core assets in the Gears solution, creating a feature model and setting variation points in the code. Efforts were put in continuous integration and introduction of new practices like Test-Driven Development to the software development (Krueger et al. 2008). Today, 'Public Sites' is the main product line of HomeAway with 20 products, 4 subsystems and a team of 8 developers add new products every second week according to the interview.

Agile Software Development was adopted in HomeAway's development from the start of the product line adoption, and a tool from Rally is used to track features in the core asset base and improve visibility of development efforts. HomeAway uses Scrum with agile practices and release software every other week, with various lengths of the sprints[16] among the teams (Rally Software 2008). Communication is described as good between developers and the rest of the organization, which could mean that they are communicating well with each other, something that is important to both Software Product-Line Engineering and Agile Software Development. Training

[16] A sprint is a period of time within an iteration used in the Scrum method.

in Agile Software Development, certifications, and motivated individuals have also helped the company combine agility with Software Product-Line Engineering. A thoughtful balance of *"thinking ahead, but not too far"* (Krueger et al. 2008) can also describe the "just enough" focus and maximizing value, often associated with the Agile Software Development theory.

Benefits of this combined approach are described as: reduced time to market; better team productivity; and higher quality. Tracking, metrics and portfolio management are some of the management benefits mentioned in the articles. Challenges mentioned in articles are initial difficulties in adoption of agile software development (Rally Software 2008) which was supported by the interview were it was said that developers had to *"learn a new set of skills and a new way of attacking the problem"*. The business perspective of Agile Software Development in Software Product-Line Engineering is also a challenge mentioned in the interview, since business alignment needs to become more agile which means that it can change faster than in normal release development. Testability was the last challenge addressed in the interview and quality assurance had a hard time keeping up with the frequent releases of software development.

We assess the agility level of the articles as high since it is mentioned that the company uses agile principles and practices and the fact that the adoption of Software Product Lines was done in an agile fashion. The level of Software Product Line Engineering effort could be described as medium since there are several indications, but not the full effort towards all the practice areas.

4.1.4 ***IFS***

IFS is described as a Swedish multinational company with about 2600 employees and a Research and Development department (R&D) of about 550 people with about 180 people in Sweden and 370 in Sri Lanka. They develop products for the ERP domain, selling and delivering software to a vast amount of customers worldwide. IFS strive to make other companies more agile and effective through their software and services in many industries, according to their homepage[17]. The need for a Software Product-Line Engineering approach can be described from the variations in the product and support for flexibility, both when delivering standard products and implementing customized solutions to the customers.

IFS were an early adopter of the component-based and modularized software structure. They have not approached software development through the

[17] See http://www.ifsworld.com/

Software Product-Line Engineering theory, but there are many similarities to the incremental approach. One of them is that they use a component-based platform to deliver customized ERP solutions to their customers, much similar to Software Product-Line Engineering’s core asset base. Variations in the solution are among others language and location support, process modeling, and custom automation according to the interview. They also implement specialized functionality based on the best practices in the domains and use this functionality in custom integration to the customers. Recently, they changed approach to the evolution of the platform from planned releases to project-based work, with customer interaction and maximized business value as focus points. These projects are triggered based on feedback from both internal and external environment such as customers and maintenance needs. The feedback process is also described as filtered in the way that customers’ customized solution problems are not sent directly to R&D. Further, when a project is assessed by management, a business case is made and a decision, whether to implement or not is decided before R&D takes over the project.

The application engineering in IFS is done distributed in the respective regions with their own methods. This facilitates a close cooperation with the customer in an implementation project. Sales are also done in the regions and initiate an implementation project through selling standard products and/or customized solutions. The high-level documentation describing functionality of the standard solution is done through view, activity, and process descriptions supported by process models, similar to the common feature model in Software Product-Line Engineering. This documentation is also described to be reused and built upon, when customer-specific needs are handled.

The interesting practices in this case are how R&D works with the incremental adjustment to the core asset base. They use a Scrum inspired, iterative model to implement the requested business cases through distributed development. A project is managed by a project leader, commercial manager and a steering committee, which uses a tollgate decision system to decide whether the project should continue or stop at certain critical moments of the project. A functional designer is in charge of the fit towards customer requirements among main stakeholders. Automatic testing, build and configuration management is supporting this process. The project starts with an inception phase, where some formal challenges are solved and the project is planned “just enough”, meaning that it describes enough details to start implementing as described in the interview. Then, iterations follow where the team design, code, test and document, before the test department take over the project and runs the acceptance test suite to ensure high quality. Last, the release management department took over the finished project and shipped it to customers.

The agility of the approach is based on the described similarities and adoption of Scrum's iterative process and stakeholder interaction. Smaller and shorter iterations and frequent deliveries are used actively. Agile planning and estimation are used to measure progress, and daily stand-up meeting improves communication in the project. User stories are used as the requirement artifacts and are placed in a prioritized backlog. Customers prioritize user stories and evaluate iteration demos in the software development process. Team empowerment, continuous testing and retrospectives are also mentioned as practices in IFS R&D.

The project-based evolution of the platform is described to enable a shorter time-to-market for IFS. Benefits regarding higher customer interaction in the agile product line approach are explained with being able to deliver what the customer want and need, ergo increasing the business value of the products. The challenges of this approach can be described as creating the process. As IFS said: *"a big, distributed organization does not fit to agile out-of-the box methods"*. Other challenges were described in obtaining team empowerment in a distributed environment and exploiting domain knowledge in a project-oriented way of working.

The Software Product-Line Engineering level of IFS' development can be described as medium, provided that they have a core asset platform, they deliver a set of software systems with a common set of features, and they try to reuse software throughout their approach. The agility level is high, since they have adopted many agile practices to their software development approach.

4.1.5 ***DNV Software***

DNV Software, a self-governing business unit of DNV[18], is described as a company delivering software products and customized solutions to the shipping, oil and gas, process, rail, automotive, and food industries. The company is divided into four departments namely 'Sales and Marketing', 'Products', 'Solutions', and 'Software Factory'. As of 2004 they had about 160 employees where about 100 of them were developers (Linden et al. 2007. Chapter 10). DNV Software are described to operate in the service domain, providing services and support with regards to classification, certification, and risk, safety, quality consulting in the presented industries.

The history behind DNV Software's Software Product-Line Engineering is based on Nauticus, a product they started developing in the early 1990s. Nauticus' vision was described to handle changing customer environment, variations in domain, and ability to deliver high quality services while

[18] Det Norske Veritas (DNV) is a Norwegian company.

continuously improving them. In order to realize this vision they developed the BRIX.COM software platform incrementally, in parallel with establishing a domain information model and implementing end-user tools. Today, the Software Factory Department is in charge of the BRIX.NET platform and its evolution, attaining feedback from both ‘Products’ and ‘Solutions’. The BRIX.NET platform was established when DNV Software emerged in DNV. The book chapter (Linden et al. 2007. Chapter 10) describes partial reuse of the existing BRIX.COM platform with new variations, a non-enforced architecture, modularized features, and an open, transparent system. Because of that DNV Software used an extractive approach when adopting the Software Product Line. They also did product development in parallel with the platform development and new functionality to the platform had to be associated with a product development project in order to be developed. This is still applicable for DNV Software as described in the interview and indicates a reactive approach to Software Product-Line Engineering.

The BRIX.NET platform called BRIX Foundation includes among others ‘Model’, ‘Workflow’, and ‘Security’ which are three important parts of the platform. The parts are described to ensure reuse and exploitation of commonalities among the applications (Linden et al. 2007. Chapter 10) while working as *“industry independent modules”*. This platform is mainly changed based on feedback from the two departments using them, and happens based on requirements from these departments.

Since both ‘Products’ and ‘Solutions’ use the product platform this case is somewhat different from regular Software Product-Line Engineering where there is usually one application engineering entity. ‘Products’ develop Commercial Off-The-Shelf (COTS) software to be sold to many customers, whereas Solutions do custom development for one-and-one customer. ‘Software Factory’ which is in charge of the platform described two kinds of reuse from the platform:

1) Release based Integration, which means that the entity does not need to change or influence the platform and can use or not use the existing functionality; or
2) Continuous Integration with BRIX, meaning that the entity work with their project in parallel and cooperation with the BRIX development usually changing, improving or customizing a component of the platform.

Our interviewee was an employee in the ‘Products’ department and their software product development are therefore best explained here. ‘Products’ utilize a waterfall approach to software product development, but are piloting a more agile approach to their development. DNV Software explained that some teams have implemented more than other, but also that the adoption of Agile Software Development takes time because of the cultural change involved. In addition, ‘Products’ is described to develop new products based on observed customer need or from internal ideas in DNV. The ‘Sales and Marketing’ department is described to handle sales and listen to customer needs.

We found little information about the management of the software engineering process in this case, except the waterfall approach explained above. Regarding marketing and sales, the department with the same name is responsible for this. We were told that ‘Solutions’ were often more involved with the customer since they do customized solutions, while ‘Products’ most often listen to ‘Sales and Marketing’ but in some cases obtain direct feedback from the customers.

Agile Software Development at DNV Software is still in the adoption phase and was started about a year ago according to our interviewee. Therefore it is hard to explain the overall efforts regarding agile principles and practices used at the company. They described that they want to use XP practices in the teams with the Scrum framework for project management in the follow up-questions. Our interviewee mentioned among others sprints and stand-up meetings from Scrum which they were to use. It was also stated that ‘Solutions’ also had some projects using Agile Software Development, but detailed information could not be obtained for this study. In addition, we found that DNV Software use own employees as their ‘Product Owners’[19], not customer representatives.

The benefits discussed in the book chapter mention quicker return on investment (ROI), additional inclusiveness and flexibility (Linden et al. 2007. Chapter 10). Challenges mentioned in the interview were maintaining models, overhead on both waterfall approach and design up-front, and release planning. The Software Product-Line Engineering level of this case can be characterized as medium since the company did not explicitly use Software Product-Line Engineering theory, but attacked the challenges in a similar way. The agility of this approach is also assessed as medium, since it is still early in the adoption of Agile Software Development. We chose to include this case study since they are early in the adoption of agile and we could find challenges related to the adoption of Agile Software Development in Software Product-Line Engineering.

[19] A product owner is the person responsible for representing the needs of the customer in the product backlog.

4.1.6 *Engenio*

Engenio, a company situated in the storage technology domain where original equipment manufacturers (OEMs) want to use Engenio’s competence in their own special solutions. They wanted to introduce Software Product-Line Engineering to cope with the increasing demand for their products, and to be able to secure sustainable growth. The case describe 180 developers distributed in four sites, 82 products and about one million lines of code (LOC) with 80 % similar code among all products (Hetrick et al. 2006). After an initial assessment the choice of Software Product-Line Engineering seemed most feasible, but the adoption barrier was too high and they chose to use the incremental or reactive approach to Software Product-Line Engineering.

Their approach was to incrementally address challenges and bottlenecks in the software product line introducing Software Product Line practice areas to solve them. The case claims that the incremental investments are smaller and will earn itself back within the transition time. A pilot project was successfully conducted, and the real transition could start with four incremental steps towards establishing the software product line (Hetrick et al. 2006). The first step was establishing the core asset base, set up a production environment, and training in Software Product-Line Engineering and Gears. The second step involved adjusting the development organization to the new approach working on core assets, while the third step changed their development process towards feature orientation instead of product orientation. The last step described was quality assurance. For a more thorough description see the case study (Hetrick et al. 2006). During these steps various Software Product-Line Engineering practices are probably used, but the article does not elaborate on this issue. We found tendencies towards component development, mining existing assets, process discipline, tool support and organizational planning practice areas described in (Clements & Northrop 2002a).

Customer involvement and core asset ownership in the teams are mentioned, increasing the agility of the approach. Few other direct agile practices were found in this approach, but iterative development and the incremental approach justify a somewhat agile approach. We also wanted to prove the ability to choose from the practice areas in Software Product-Line Engineering (Northrop et al. 2007) and use them in an untraditional way.

The benefits discussed in the article mention: a sustainable core asset base; return on investment (ROI) early in the transition; reduced time to market; small big design up front (BDUF) (Hetrick et al. 2006). Challenges are not mentioned, but specialization on core assets for developers could be one of them. The agility of this approach is rather low, but we are fascinated by the ability to change in an efficient matter, and see the overall approach to fit well with the business related values to agility. Regarding the Software Product-Line Engineering level we characterize this case study as medium since little is known about concrete practices used in the company.

4.1.7 ***Testo AG***

Testo AG is a German company who delivers portable measurement devices for industry and emission business. They have a successful Software Product Line, which was started in 2001, that has delivered 15 products so far according to (Carbon et al. 2008). They followed an incremental approach by following Fraunhofer's PuLSE™[20], which is described as *"architecture-centric"* meaning that the architecture leads development. Testo's development cycle time for new products are described to vary from half a year to one and a half year, and employed 35 developers which could be characterized as a small development organization.

Testo AG had no established feedback practice between their application and domain engineers and they needed a lightweight and informal practice to assess the growing problem of architectural mismatch (Carbon et al. 2008). The article only covers enhancements of some practices in Software Product-Line Engineering, assumedly Architecture Evaluation, Component Development, and Testing, to obtain more agility and maximizing the value of their approach. The researchers' idea was to use the agile practice Planning Game to assess the feedback need identified, switching the roles of customer and developers with application and feedback engineers (Carbon et al. 2008). This was believed to enhance the planning of evolution and optimization of artifacts within the Software Product Line and introduce a new Software Product Line Engineering practice area, the product line planning game.

In short this practice introduces reuse stories (instead of user stories) that the application developers formulate based on their feedback on reuse challenges of artifacts in the core asset base in (Carbon et al. 2008). Then, the stories are estimated by domain developers and refined through the process described. The result is a set of estimated reuse stories that could be input to the scoping process and future requirements. These could further be developed through iterations of core asset base development. The main difference though is that the application developers (same as customers in the agile practice) do not choose which reuse stories to implement; rather the product line manager and architects are in charge of directing the path for the product line (Carbon et al. 2008).

Benefits mentioned in the article are: varied viewpoints on different components; knowledge transfer between members of application and domain engineering;

[20] Fraunhofer PuLSE is a registered trademark.

and high efficiency in feedback process. The reuse stories are also subject to implementation monitoring and traceability through inspection in reusability (Carbon et al. 2008). The goal for this approach could be improving the reusability of the core asset base, but the article does not clearly state this. The article further state that the practice could fit well for small development organizations or departments who need a lightweight feedback practice.

After experimenting with the practice in three workshop handling two aspects of the product line feedback, the authors found that the product line planning game provides feedback and supports scoping in a Software Product Line. The workshops was introduced, moderated and facilitated by the researchers. Participants' creativity and knowledge are described to have strong influence for the outcome of this practice, and was one of the two limitations found. The second limitation was the fit to distributed environments because of the strong emphasis on discussion. We chose to not assess the agility and Software Product-Line Engineering level of the practice studies, since it is hard to reason about the overall level when one only know the details of one practice.

4.1.8 ***PROSOL***

The last company described is PROSOL, a small Austrian firm, which deliver software in the supply chain management domain with customers in European countries. The company aims for expansion and their goal is to make an incremental approach to learn, build and maintain a Software Product Line (Noor et al. 2008). One of the first steps to obtain this is a product roadmap, and the researchers have looked into the practices for creation of this artifact.

The practice is built up by adding agile principles and collaborative engineering facilitated by thinkLets[21], to existing Software Product-Line Engineering practice areas. A set of tasks is defined based on needs from the practice in Software Product Line Engineering, then thinkLets and input is associated with each of the tasks to be able to create the output of each task. In the end, a product map and a development plan are outputs from the new practice. The tasks handle understanding domains, features exploration in domains, product scoping, product features and prioritizing a product map (Noor et al. 2008) which are all practices in the Software Product Lines field (Northrop et al. 2007). The thinkLets vary from task to task, but the main idea is to be able to be creative first, and then analyze the creative results to be able to evaluate them in the end. A discussion about stakeholders that could participate in the practice, describe potential participants in this practice as: customers, management, marketing and sales, architects, developers and maintenance staff (Noor et al. 2008).

[21] thinkLets is described as patterns of group collaboration (Noor et al. 2008)

The agility of the practice concerns flexibility and change. The stakeholders involved are allowed to change the process according to results during the execution of the practice (Noor et al. 2008). The practice was also described as easy to learn and reduced the waste by ruling out less promising ideas. In a Software Product Line setting the approach presents an agile practice that could be used instead of established practices. The benefits of this approach are described to be efficiency and value creation (Noor et al. 2008). Challenges with the practice are not mentioned, but knowledge in product line planning, preparations and cooperation is mentioned as success factors. Again this case study shows that creative, lightweight approaches can be used instead of somewhat comprehensive established practices.

4.2 Practices Found in Software Product-Line Engineering

The second section of this chapter describes the practices used in the industry sorted into the theoretical practices described in (Section 2.2). We elaborated on each case and tried to combine the results in themes based on the practice areas and our initial presentation of them. Our sources was not complete in the sense that they covered all practices, so we choose to combine practices into bigger themes were it seemed reasonable. This section will state our results with regards to Software Product-Line Engineering practice areas used in industry.

4.2.1 *Software Engineering*

The results when it comes to software engineering follow the structure we recognize from Chapter 2. We report our findings in three main topics namely ‘Domain and Requirements’, ‘Architecture’ and ‘Development’.

Domain and Requirements

For the domain topic we found that Salion obtained research in the manual processes in the domain and could be seen as both market analysis and domain knowledge. This knowledge was further developed into business cases used as a domain model. Further, Salion also employed use cases to their business actions as well as the development effort. Engenio were somewhat a leader in their domain based on the high demand for their product. Engenio also staffed their component teams based on their knowledge in the sub domains. HomeAway used localized knowledge together with experienced management for their domain perceptions. CompNN were described to have a proficient product management team with broad domain knowledge leading the development.

DNV Software is described to obtain important information about their domains through DNV[22] and 'Marketing and Sales'.

As for requirements engineering, it is usually divided into domain requirements and application requirements. Domain requirements usually emerge from domain knowledge and scoping, but also variation handling could be included. Salion, Engenio and HomeAway all used a feature model (in Gears) to control and introduce variations in the core asset platform. CompNN used a set of predefined configuration for the most common uses of their platform, while other variants are subject to customization. Custom development efforts in CompNN are usually also included in the platform. Salion had requirements from their early efforts and used this to guide the development of their core asset platform. IFS described a process where domain requirements are developed by management and passed on to development teams through business cases while application requirements are gathered distributed in the regions at IFS. Requirements for domain development are handled by their inception phase were a high-level design is created, and detailed design emerges in the iteration. Further they described the abilities of their standard product through view descriptions, activity descriptions and process descriptions. DNV Software used their application engineering departments to gather requirements to the platform development.

When delivering products CompNN used product goals as high-order requirements while the detailed requirement are made just in time when they are needed. The product goals are influenced by stakeholders' feedback from the last iteration and decided before the next iteration. This way they reduce the up-front design effort and direct the development as they proceed. HomeAway are described to use Scrum and should therefore use some kind of planning and prioritization through user stories. Further, HomeAway were described to not do any other models, because the cost is high and value low, *"no one reads them"*. Salion is described to use the Rational Unified Process (RUP), and includes models in their requirements engineering effort.

Architecture

Salion and CompNN both described flexible architectures that could withstand change and evolution. The case studies emphasize this to be able to both serve customers needs and own component addition. More specific, Salion are described to use a three-leveled architecture with variations through the Façade design pattern. CompNN uses a modular platform. HomeAway's transition describes a modularization and component-based architecture, but sees it more as a necessity with Software Product-Line Engineering and it is not elaborated.

[22] Det Norske Veritas (DNV) is the mother company of DNV Software.

Engenio also describe a core-asset based separation of the architecture of the two programs the product line was built from. IFS also had a component-based architecture of their solution and DNV used a module-based platform called BRIX Foundation.

Development

The development effort is described by CompNN's platform is defined by the regular use cases, but supports variations through license files. HomeAway, Salion and Engenio used Gears to handle variations at implementation level. CompNN also provides customer with the opportunity to have custom needs implemented if the need could be included in the core asset base and fits the strategic direction of CompNN. DNV are described to deliver both customized solutions to single customers and package software to multiple customers, choosing if they use the BRIX platform or not. The BRIX platform is further explained to be enhanced based on feedback from 'Products' and 'Solutions' in DNV Software. IFS had a component-based product where the components are loosely coupled and can also be used in customized solutions for customers: *"Components are big, and can be used either horizontal for standard implementation or modules can be taken out for vertical integration"*. IFS described that they have a devoted department for maintaining and enhancing this platform, while the installation and application engineering is done distributed.

Salion used OTS[23] components for unimportant and standard aspects of the system; the rest is subject to in-house developed. CompNN used latest technology for their component-base, but it is not described closer. IFS are also described to use OTS components and the latest technology in their technical framework.

Engenio, HomeAway and Salion used existing software to establish their core assets. Engenio used two programs as a basis, but did not alter much of the code base. The case describes only introduction of variations in the new core assets and incremental improvement of these assets. The incremental improvement involved refactoring, abstracting and balancing tradeoffs in the component base. HomeAway produced a "one size fits all" platform from one of their sites before using this platform in their product line approach. Salion had planned and started a platform solution before they decided to try a product line approach. DNV were described to use an existing platform to build their BRIX.NET platform (Linden et al. 2007. Chapter 10), here called BRIX Foundation.

[23] Off the shelf (OTS) is associated with "ready- made" components that could be used as parts of a development effort. A framework or library is an example of this.

All the full-case studies are described to use continuous integration in their development. Engenio, Salion and HomeAway used the Gears software combined with continuous integration. Engenio described their “configurator” which built and run tests on deployed code automatically. Salion had similar automation with build and testing running together and frequent. HomeAway describe that their testing got improved through their new software development approach (Krueger et al. 2008), and they have own quality assurance people taking over the code after an iteration for ensuring high quality as described in the interview. IFS described going from a regime where testing was done in the end of a development effort, towards automation of tests and testing up-front instead of waiting to the end.

4.2.2 *Technical Management*

The results when it comes to software engineering follow the structure we recognize from Chapter 2. We report our findings in three main topics namely ‘Scope and Technology, ‘Process’ and ‘Management’.

Scope and Technology
To scope the Software Product-Line Engineering effort Salion mentioned that they started with an informal scope and let it evolve as customer products were delivered. After a temporary standard emerged they could further use this scope for not only development efforts but also in marketing and sales. CompNN and HomeAway were found to use a roadmap with opportunities for new products and enhancements. CompNN also had a close cooperation between operations and developers with a feedback system to determine the scope and technological future of the products they deliver. IFS also had a similar solution based on business cases which are evaluated according to business value.

Feedback planning game also realized an agile approach to this practice. The Testo case stated a practice improving the feedback between application and domain engineers in an established product line. It also means that this approach is efficient and easy to learn.

Gears are used by Salion, HomeAway and Engenio. As HomeAway puts it:

> *“Gears is used to: model features across the sites; manage variations in terms of language and ‘look and feel’; and also to decide where to generate configuration files”*

Salion also uses an analytical code cloning detection tool called CloneDR. HomeAway described the use of Rally, an agile management and tracking tool. IFS described a technical framework responsible for efficiency and productivity in the development mentioning Oracle, PLSQL, and Visual Studio among others as supporting tools. They also focus on keeping this technical framework up to date.

Process

With regards to the process CompNN and Salion both described a strategic focus and an operational focus. CompNN realized this through scoping and creating a roadmap on higher level every year, Salion on the other hand used VRAPS and vision meetings for strategic focus. On the operational level they are respectively guided by the use of EVO and a lightweight RUP. Salion also used a set of meetings to guide their product line effort with regards to: business opportunities; customization and configuration; and implementation challenges. HomeAway used the Scrum method *"out of the box"*[24], with a various degree of adoption among teams in the organization. In the 'Products' department of DNV Software it was described that they use a waterfall approach with two paths. However they are introducing more agile practices and trying to adopt new techniques as described in the interview. IFS had their own process to fit their distributed work situation and formal requirements, built from Scrum.

CompNN, Salion, and HomeAway all described different approaches to measurement and tracking. CompNN uses their product goals, weigh and estimate them according to both complexity and value then tracks the development effort. The Salion case mentioned many measures to assist in both capacity predictions and risk management on the organizational level, while they are also used in production to track and self-diagnose. HomeAway mentions the tracking more implicit through its rally software tool, and also relies on statistics from their market analysis efforts. None of the cases mention tracking of reuse, even though Engenio and Salion states the improvement and reuse rate at that moment.

Management

In the management topic we found that in addition to the continuous integration and automatic builds described, we found some configuration management practices. CompNN separated the components to the teams and restricts teams to work on their component. HomeAway protects their live-systems and has a notification solution when a variation point is changed. Engenio relied on change request from component team to component team. Salion used contracts between components, especially front-en to back-end to reduce misunderstandings.

CompNN and Salion applied some kind of a maintenance or enhancement week. CompNN had a "green week" where they correct errors and enhance the quality

[24] "Out of the box" means that they are following the original theory or description to adopt the method.

of their core assets, and a chief technical officer (CTO) who is in charge of the platform with its business alignment and the products. Salion has the same kind of "chief architect" and also a "quiet week". HomeAway followed agile and relied on the individual and teams creating sustainable software. A technical lead meeting every week is also mentioned in the interview. DNV Software described to use a strict update routine where the installation of an updated version has to go through the "main branch".

4.2.3 *Organizational Management*

The results when it comes to software engineering follow the structure we recognize from Chapter 2. We report our findings in three main topics namely 'Environment, 'Indoctrination' and 'Organization'.

Environment
The business case of the companies was briefly described in each case's description and reflects the environment of the studied companies. CompNN and Salion described their market analysis efforts and were not found to be uniform. CompNN did this through a roadmap, while Salion had a set of business use cases where their potential markets are included. Further, Salion obtained analysis results from its former company and related domains. HomeAway was described to have a feature-based evolution on their product line (Krueger et al. 2008), in addition they have business functions performing metrics, statistics, and usability tests in order to direct the product lines which they described in the interview. DNV and IFS also had departments or entities handling market and sales. IFS also described driving implementation of new functionality to the platform through business cases evolved from market analysis: *"We try to have one business case for one platform development project"*. The cases also mentioned the restrictive scoping of CompNN only allowing custom development for customers if the efforts are included in the platform, and Salion's efforts in reducing customizations.

When it comes to customer interface management CompNN used an Advisory Board to collect feedback, ideas and future requirements from important customers. Salion used a two-phased delivery strategy which includes a standard delivery within 60 days and then the customer can add specific needs. In this way Salion and the customer could speak at the same level and use the existing installation as a guide for customization. HomeAway had internal product owners serving as customers for their development, much of the same that were found in DNV Software. DNV Software described that they do this because they are producing for many customers and mean that they know their market best, but see the need for directed knowledge to the developers. In IFS the customers were distributed in the various countries they deliver software, and have regional user groups evaluating the software. They explained that the big regional markets have more influence on the direction of the software because they have a representative in high-level management meetings where scoping is done. IFS relied on a support process for feedback, including feedback from the distributed implementation departments and customer feedback through a support system. Customer specific problems stay in the region, while overall problems are sent back to R&D.

Indoctrination

Salion, HomeAway and Engenio used a step wise approach to indoctrinate their Software Product-Line Engineering effort. Salion emphasize support and communication through their company to be able to succeed. HomeAway had experts supporting the transition, and innovators that lead the adoption of software product-Line engineering in the right direction. Engenio identified bottlenecks and implemented the engineering approach as a phased transition. For CompNN the big transition was to EVO and handling more loose plans in the strategic level (Hanssen & Fægri 2008). DNV Software has an open approach to reuse of the core asset base, projects can choose more freely to use platform components or not. IFS are also reliant on their component–based structure and use that as a competitive advantage.

HomeAway described to train and teach employees about Software Product-Line Engineering and Agile Software Development. It was also mentioned that they used a wiki for development related information. We also found weak indications that Salion used knowledge management and captured experiences in the core asset base. Another commonality is that four of the full scale cases were introducing Agile Software Development in parallel with their software product-line engineering and had to learn the principles of their combined approach individually and through consulting. DNV Software tries to attract individuals who have high knowledge in Agile Software Development and could work as champions to adoption of the new practices from this field.

Organization

The process disciplines mentioned (Section 4.2) should lead the planning efforts of the cases, but little concrete information about this is found. Risk management is to some extent corporate in the process as well and not explicitly described. The only similarity we found is that is seems like the companies use iterative development and frequent builds to reduce risk. User stories were used in various degrees at HomeAway, IFS, and DNV Software.

CompNN, Engenio, and HomeAway structured their teams and organization according to modules or components and want to make experts in the various components. We found indicators that most of the organizations used small teams for developing easing communication and frequent delivery. IFS were more project-oriented with their resources, but tried to take advantage of employees' skills and knowledge.

In the day to day operations Salion was described to use three different product cycles at one time with 30 day release cycles. Model-driven development was also present in Salion through the Rational Unified Process (RUP). HomeAway uses their Scrum method for operations. CompNN had Impact Estimation Tables as their day-to-day operating tool. DNV Software has product owners on each product in 'Products' department and uses a waterfall approach.

4.3 Practices Found in Agile Software Development

The third section of this chapter describes the practices of Agile Software Development used in the industry sorted by the categorization of practices described earlier (Section 2.3) describing each practice separate. The sources did not cover all practices, so we choose to combine some practices into bigger themes were it seemed reasonable. Since some the studies were directed for the Software Product-Line Engineering field the results here might not have been as complete as we wanted.

4.3.1 *Software Engineering*

The results when it comes to software engineering follow the structure we recognize from Chapter 2. We report our findings in three main topics namely 'Domain and Requirements', 'Architecture' and 'Development'.

Domain and Requirements
The Domain-Driven design practice and Software Product-Line Engineering work well together since it is one of the ideas behind the approach and are explained earlier (Sec 4.2.1). HomeAway described to use vision, roadmap, elaboration planning. The vision is driven by market and business efforts, while the roadmap is done to assign features or functionality to a time frame, while elaboration is done through a backlog. IFS had a similar adoption of this practice were management work with a roadmap consisting of business cases, while development elaborates on business cases of this roadmap when a project is initiated. The PROSOL case described an agile way of doing planning and prioritization for software product line engineering. Further HomeAway's Scrum and CompNN's EVO methods also did planning and prioritization. Scrum are described to have a iteration planning meeting before an iteration starts where user stories are planned in tasks and prioritized often with the customer (Abrahamsson et al. 2002). EVO uses the Impact Estimation Table to do the same thing and prioritize in regards to risk and business value (Hanssen & Fægri 2008). IFS did planning and prioritization in their project

inception phase, and user stories were estimated and prioritized for use in the iterations.

Architecture
With regards to the initial architecture no information was found since the cases built upon existing platforms, but this could be a kind of initial architecture even though the agile community might see this practice as a more lightweight approach (Leffingwell 2007).

Development
In the development topic HomeAway were the only company who described to use coding standards as a tool in their quality control, but we would think it is normal to implement in software development organizations. CompNN, HomeAway, IFS and Salion were described to use Test-Driven Development, but the actual effort is not elaborated. Other test efforts are described in *Testing* (Section 4.2.1). Salion used a tool and refactoring to extract commonality in their software product-line. They found similar code blocks and refactored it to introduce more commonality in the core asset platform. HomeAway described efforts towards software evolution with *"discipline and good people maintain the integrity of the software system and design"* ref which are hard to do without refactoring.

The Salion case also described code ownership or component ownership of developers as a helpful way to ensure reusable core asset base. The Engenio case mentioned ownership implicit through their confidence in the product line and belief in own abilities to respond to opportunities. HomeAway was described to rely on their developers to cope with software evolution as mentioned above.

4.3.2 *Technical management*

The results when it comes to software engineering follow the structure we recognize from Chapter 2. We report our findings in three main topics namely 'Scope and Technology, 'Process' and 'Management'.

Scope and Technology
We found little information about scoping in the agile practices. When it comes to technology Salion, HomeAway, IFS, CompNN and Engenio mentioned frequent builds, automation, and continuous integration as ways to obtain better quality. The Salion case described a comprehensive test suite that run automatically after coding on a release is done. HomeAway presented an automated configuration management, build and deployment infrastructure (Krueger et al. 2008):

"when a piece of code is checked in after code review, it is run on the build server with all product and unit tests before a successful build is automatically deployed to a cluster of servers"

IFS were found to use automatic build and configuration management. CompNN was described to have projects and sprints within each project. Engenio used their 'configurator' and Gears for this automation.

Process

As for the process topic HomeAway was described to use Scrum, while IFS had their own iterative process and they both use a backlog to guide their development. CompNN used their IET table to guide development. CompNN's EVO method emphasized this practice. HomeAway describe varying sprint lengths, but always a day for planning the next sprint. HomeAway was described to use the Scrum method which involves these qualities. Salion used the RUP method which is also an incremental, iterative approach to software development, but it is model-driven and are usually not characterized as an agile method because of that. DNV Software described to use sprints as well, without elaborating how they do it. IFS used iterations as one practice to produce results in a process where brief planning are done before and quality assurance with deployment are done after.

Management

To manage the effort Salion, HomeAway, CompNN and Engenio all mentioned small, frequent releases. IFS described that they use shorter iterations, while DNV Software described to use Scrum which implicit means using sprints for frequent releases. We also found that HomeAway used a Wiki[25] for information about the technological environment and development with "how to create a branch" and "how to do a code review" as examples. IFS did this implicit by sending material from their Wiki.

4.3.3 *Organizational management*

The results when it comes to software engineering follow the structure we recognize from Chapter 2. We report our findings in three main topics namely 'Environment, 'Indoctrination' and 'Organization'.

Environment

To handle the environment two of the cases described customer interaction in a more comprehensive way than the Software Product-Line Engineering practices intends it to be. CompNN used both an Advisory Board to reveal and discuss future needs, and a stakeholder representative that could be a customer for their

[25] A wiki is a website for easy creation of multiple web pages often used for documentation in software companies.

projects. This stakeholder representative is responsible for feedback on the iterations and guides the coming iterations (Hanssen & Fægri 2008). Engenio mentioned changing customer requirements as one of the drivers for changing their process, but does not describe closer how they improve their predictions of customer needs. Salion used customers' needs to drive their own scope and create core assets from. DNV Software and IFS were described to use internal resources as customers to platform development, while both try to use customers in application development As IFS described in the interview:

> *"Every project has a commercial manager who is in charge of directing development according to the business case and customer interaction. This ensures high value and quality since development fits the customer need"*

Besides this we can also call feedback from application engineering to domain engineering a kind of on-site customer. The planning game approach in application to domain engineering feedback from the Testo case is an example of a working agile way of obtaining important feedback throughout the organization in dynamic work environments. Salion also had their meetings to ensure that communication was frequent in every project and the HomeAway case indicated a healthy communication between developers and the organization. IFS get feedback from customers and distributed departments. The support process was filtered, meaning that the R&D department only got the feedback that is applicable to the platform. The customers were usually organized in communities and try to speak everyone's case towards enhancements to the platform.

Indoctrination

To indoctrinate Agile Software Development HomeAway learned more about agile practices through coaching from experts (Krueger et al. 2008), and they used retrospectives after each iteration to improve according to the interviewee. IFS also did retrospectives to assess the iteration and propose changes to the iterations. The Salion cases indicated that the organization reflected and tried to improve through trying out new things. The planning approach in PROSOL was also able to change according to how the process evolved.

Organization

As for the organization HomeAway, CompNN, Salion describes the use of metrics to track and evaluate development. HomeAway experienced better visibility and expense control through certain measures, while CompNN and Salion used it as a planning tool and measures team velocity through estimates on the work packages. IFS estimated user stories and used that as a metric.

Salion has such a small group that everyone is doing everything, except from that no indications of this were found. HomeAway was described to have only one team on the 'public sites' product line and they do all functions. IFS also had self-organizing teams, but experienced some challenges regarding team empowerment and domain competencies.

Salion has a different approach called "joyous love" meeting twice a week to solve challenges and track projects, we also heard about daily stand-up meetings in the interview. HomeAway use stand-ups in their Scrum approach. DNV Software is also described to use stand-ups in some projects.

5. Discussion

The overall combination between Software Product-Line Engineering and Agile Software Development are discussed in this chapter. This introduction answers RQ-1.4, where we asked about introducing agility to Software Product-Line Engineering and how it works in the industry. The following discussion illustrates the main findings and are also summarized (Table 11). Our case study companies introduced agility in Software Product-Line Engineering in different ways. Six of the eight companies did a full scale combination between Agile Software Development and Software Product-Line Engineering, while the two last ones changed a practice area within Software Product-Line Engineering to become more agile. HomeAway, Salion, and Engenio were described to combine Agile Software Development and Software Product-Line Engineering concurrently from the start of their Software Product-Line Engineering efforts. The second variant of combining the two approaches were employed by CompNN, IFS, and DNV Software who had the Software Product-Line Engineering effort running, but wanted to become more agile to serve a dynamic market. DNV Software is still in the phase of establishing their combination, while IFS have done pilot testing and are ready to adopt the resulting method they have tested.

Further, we found that the case studies adopt Agile Software Development in various levels in their Software Product-Line Engineering approach to software product development. The domains of the companies varied, but can be described as customer-driven and fairly stable. The size of the companies can be described as small and medium, with team-based efforts in software engineering. The problems the companies have, like most other companies in the industry, are delivering high quality software to a set of customer that has different needs. Overall we can say that they try to combine long-term strategic winnings with short-term flexibility. We can also say that the companies feared the big design up-front effort which are associated with Software Product-Line Engineering, and wanted to reap the benefits from the approach without spending lots of money and a big amount of time in advance.

The context of the companies we investigated was somewhat similar. They deliver a set of products that shares commonality to a set of customers. They were all using small teams and many companies did distributed development. The domains varied, but all supported the product line capability. The complexity could be considered high and the products can be described as unmanageable through regular single system maintaining.

After analyzing and investigating the companies' approach to Agile Software Product-Line Engineering we believe that Agile Software Development can be both a mindset and concrete practices that could further enhance Software Product-Line Engineering. We see that the companies often reduce time-to-market and are able to exploit reuse in a much higher grade than possible with other reuse approaches using a hybrid approach between Software Product-Line Engineering and Agile Software Development. Agile Software Development practices introduce both principles and practices that fit well towards the product lines in the companies we studied. The methods in Agile Software Development fit well on the technical level and separate practices could increase efficiency, but on an organizational level agile lacks the comprehensiveness of Software Product-Line Engineering. The ability to change while not being directed by models and "signed"[26] requirements as agile proclaims could be a benefit for the Software Product-Line Engineering effort. Another benefit of combining the two approaches could be that the agile methods are lightweight and easy to learn which again could foster incremental learning of Software Product-Line Engineering. The findings also showed us that the combination could reduce the big design up front. These are some of the genral benefits Agile Software Development provides to Software Product-Line Engineeering.

[26] By signed requirements we mean contracted requirements that are not able to change after they have been agreed on.

Table 11: Summarized findings of the analysis of the case studies.

What	Similarity	Varied results
Customized products to several customers	X	
Reduce BDUF	X	
Small or medium sized company	X	
Distributed environment	X	
Combining Product Line and Agile Software Engineering	X	
Using special tools	X	
Domain stable		X
Introduction of agile in a product-line		X
Using practices and principles from Software Product Line Engineering		X
Using practices and principles from Agile Software Development		X
Knowledge about Software Product-Line Engineering		X
Knowledge about Agile Software Development		X
Approach and introduction to product line		X

- 71 -

More specific, the similar benefits between Salion, Engenio, and HomeAway were that they all reduced the big design up-front, and incrementally adopted a Software Product-Line Engineering approach in an agile fashion. They were also able to reduce their code base because they could 'refactor' branched code into commonalities and exploit reuse benefits. Further indications about reduced time to market were also presented, and higher quality through maintainable source code was mentioned in these cases. In CompNN's case the combination of agile software development and software product-line engineering fulfilled long-term strategic needs with short-term instability and changing customer needs (Hanssen & Fægri 2008). IFS can be described to obtain the similar benefits as CompNN.

Regarding the two approaches of Agile Software Product-Line practices, we have illustrated that it is possible to obtain agility in the practice areas of software product-line engineering through combining the practice area with thoughts from the Agile Software Development field. In addition, we showed that practices in both fields can be combined to make processes more agile. We elaborate on this based on our results in the next sections of this chapter.

In the following sections we present our discussion in a top down fashion starting with a discussion of the characteristics of agility in Software Product-Line Engineering. Then, a framework for Agile Software Product-Line Engineering is presented following the patterns of Software Product Lines described in (Section 2.1.4), before we elaborate on the practice areas and practices we gathered results about. In the discussion we try to combine our results to describe where the Agile Software Development practices could fit in the Software Product-Line Engineering practice areas following the same structure as earlier in the thesis. Last, we discuss the validity of our approach.

5.1 Characteristics of Agility in Software Product-Line Engineering

After discussing the overall results, we believe that it is possible to combine Software Product-Line Engineering and Agile Software Development, similar to what Ghanam (2008) and Tian & Cooper (2006) have reasoned before. To add another dimension to our thesis we try to establish a meaning (RQ-2.1) and some characteristics (RQ-2.2) for agility in Software Product-Line Engineering. First, we try to combine the definitions of Software Product Lines and Software Product-Line Engineering with the definition of agility. If we follow Conboy's definition as a basis we could easily rename the entity making a proposed definition of agility in Software Product-Line:

> *"the continual readiness of software product line engineering to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while maximizing value, through its collective components and its relationships with its environment"*
>
> (modified from Conboy 2008)

Or we could combine the definition of Software product line and create this proposal:

> *"the continual readiness of a set of software-intensive systems sharing a common, managed set of features to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while maximizing value, through satisfying the specific needs of a particular market segment or mission with a common set of core assets reused in a prescribed way and its relationships with its environment"*
>
> (modified from Conboy 2008; Northrop et al. 2007)

A third option would be to combine Software Product-Line Engineering and the agility definition:

> *"the continual readiness of software product line engineering to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while maximizing value, through its collective components, platforms, mass customization and its relationships with its environment"*
>
> (modified from Conboy 2008; Pohl et al. 2005)

The first proposal inherits the agility definition to Software Product-Line Engineering. The influence it would have on Software Product-Line Engineering is that the approached has to be more able to change than it might be today. We think that the maximizing value statement could support new lightweight practices in Software Product-Line Engineering while the change directions would modify common practice areas. This also supports the results and discussions performed in this thesis earlier. Putting both definitions together resulted in a long definition of Agile Software Product Lines. This definition describes how Software Product Lines are supposed to be open to change and maximize value in a certain market segment with reuse. We think it covers the meaning of agility in Software Product Lines well, but it might sound easier in words than it is in practice. Our third proposed definition is similar to the first, but adds platform and mass customization as a tool for change and maximizing value.

In our opinion the agile manifesto tries to encourage people, code, customer, and changes whereas Software Product-Line Engineering clearly use processes, tools, documentation and plans. Contract negotiation in Software Product-Line Engineering is not a big challenge because the choices or variations usually constrain what is possible or not and are decided by the producer, except for custom development which is usually avoided. Software Product-Line Engineering has to embrace the values of the manifesto to obtain more agility. Especially reduce the big design up-front efforts in the proactive approach, find a good way to embrace the customer, and have the ability to change according to the need.

Elaborating on RQ-2.2 we divide the problem in the three different approaches to Software Product-Line Engineering namely proactive, reactive and extractive. In proactive Software Product-Line Engineering agility could be earlier development of code, test-driven development, emphasis on code not documents, customer interaction through business people, self-organizing autonomous teams, component-based development with automatic integration, allow agility from the start of the effort. Problems that could evolve are control, decision structure, creating “just enough” plans and documentation, and variations. An agile approach to the proactive approach could import many practices from the agile methods. The long-term strategic objective still needs to be in tact to be able to exploit the product line in a beneficial way. Further Software Product Line practice areas supports many of the agile elements, but documentation and design are probably the worst bottlenecks. A way of concurrent development and cooperation between business and development could be beneficial.

For the reactive approach, agility could fit even better since we build parts as they are needed or an opportunity arise, and could make projects from this to develop on the platform. Agile Software Product-Line Engineering could experiment with practices and find out which could work and not. Customer is a little more present since we deliver products in parallel with developing on the platform. Automation, component self-organizing teams, and test-driven development in addition to reduction of documentation and design up-front are things to emphasis when trying to make this approach more agile. This approach does not need that much control since we are adding pieces bit by bit to the core asset. Refactoring could evolve to be a big challenge with an approach like this because of the variations. Iterative development fits well with this kind of approach.

Last, the extractive approach fits the refactoring nature of agile. Building a core asset base from products could implement agile teams as well. Here the teams could think about change when building and use agile methods and practices for the building. The need for documentation and design up-front is not as important here as in the proactive approach. However, we still need the strategic focus and direction by the business part of Software Product-Line Engineering. Architecture is also a question, since we need variation.

Based on the results in this thesis and the above discussion some characteristics for agility in Software Product-Line Engineering could be:

- reduced documentation and models;
- lightweight requirements and architecture;
- test-driven development;
- iterative processes;
- automation; and
- component teams.

We think the reactive, incremental approach to Software Product-Line Engineering could be the best match for Agile Software Product-Line Engineering.

5.2 Framework for Agile Software Product-Line Engineering

In order to answer RQ-2.3 about incorporating agility to Software Product-Line Engineering companies, we try to use the results found in this study, combine them with the patterns of Software Product-Line Engineering to establish a framework for Agile Software Product-Line Engineering. We saw that companies were able to both include agile practices in their Software Product-Line Engineering effort and change Software Product-Line Engineering practice areas to become more agile, but also that some practice areas in Software Product-Line Engineering are not covered by Agile Software Development and remains similar to what they were. This section will try to look at the patterns suggested for Software Product-Line Engineering, discuss how they could be made more agile. We start with the 'Factory' pattern which describes introduction of a Software Product Line. From that we work ourselves down in the patterns that make up this composite pattern. Bear in mind that we look at the introduction of agility to Software Product-Line Engineering not covering introducing product lines to Agile Software Development organizations.

5.2.1 *The Agile Software Product Line*

In (Gylterud 2008), we stated that we think Software Product-Line Engineering and Agile Software Development could be combined to further increase efficiency and quality while reducing time to market using Agile Software Development in a Software Product-Line Engineering organization. This thesis have showed how companies tend to approach such a combination with success being able to reduce big design up front, increase quality and exploit reuse. To add a contribution to the field we combined our findings with the patterns of Software Product-Line Engineering, and look at Agile Software Product-Line Engineering in a top-down fashion starting with a variant of the 'Factory' pattern namely the 'Adoption Factory' pattern.

The 'Adoption Factory' pattern describes the whole product line effort with putting together several patterns. We think that Software Product-Line Engineering can become more agile when the sub-patterns included in this pattern have achieved agility. We imagine two scenarios when constructing the Factory pattern for an Agile Software Product Line. The first scenario (S-1) is an organization that is looking into adoption of Software Product-Line Engineering and creating one or more product lines based on domain knowledge and a product base. The second scenario (S-2) is an organization where Software Product-Line Engineering is established and one or more product lines already exist. Here the need to cope with new market demands and evolution of the core asset base is present. To document the effort in these two scenarios we used the 'Adoption Factory' pattern briefly described in (Clements et al. 2006). This pattern originates from (Northrop 2004), where the author describes Software Product-Line Engineering adoption needs, and abstracts it with a similar figure as we have modified. In S-1 we have made changes in the different sub patterns and introduce a new pattern 'Parts Factory' to reflect Agile Software Product-Line Engineering based on the discussion on practice areas (Section 5.3) and the contradictions of the 'Each Asset' pattern towards agility since it separates work. S-2 handles Software Product-Line Engineering organizations who introduce agility and we changed the original pattern with the 'Warm Start' pattern since the organization has already implemented product line(s) before and we also adopted the 'Parts Factory' discussed in S-1.

5.2.2 *Scenario 1 (S-1)*

In this sub section we describe how an organization could adopt Agile Software Product-Line Engineering without having implemented product lines before. We start by discussing the establishment of the context, then explaining how the production capability could be built before we handle the operating of the software product-line (Figure 5).

Context
On the organizational level we use the 'Cold Start' pattern. This pattern introduces additional challenges when trying to combine Software Product-Line Engineering and Agile Software Development because the organization probably has to introduce both Software Product-Line Engineering and Agile Software Development concurrently.

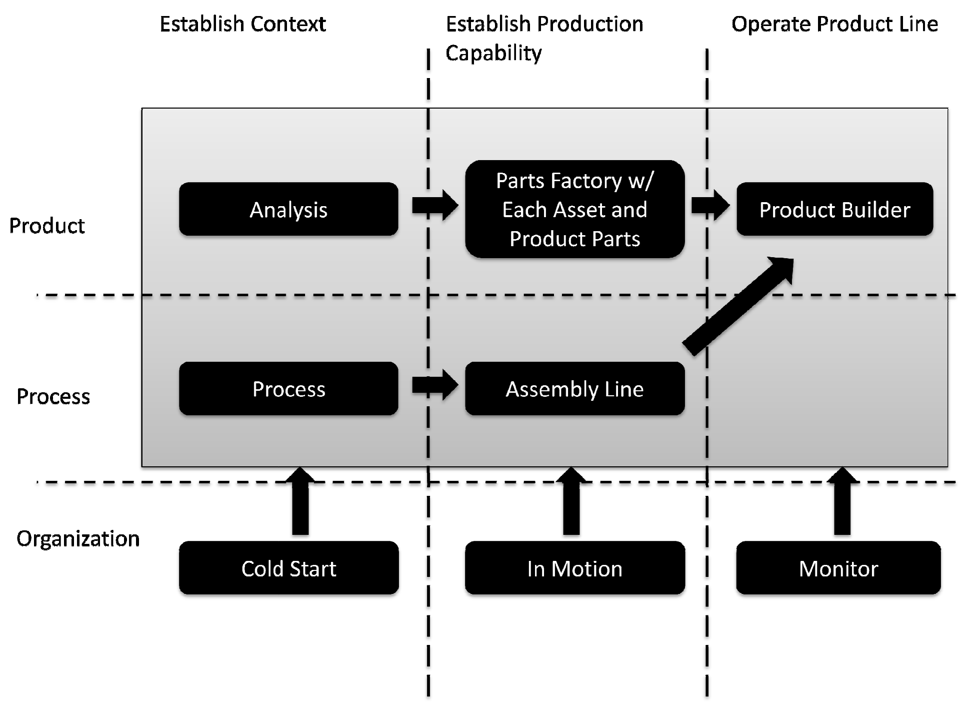

Figure 5: The Adoption Factory pattern for scenario 1 modified to fit an Agile Software Product Line. (modified from Clements et al. 2006)

The pattern communicates the new product line effort and sets up the new environment handling the launch, funding, structure and operations. From Agile Software Development we make changes to ‘Organizational Planning’ and ‘Operations’ practice areas as described in (Section 5.3.3), making the ‘Organizational Risk Management’ and ‘Training’ practice areas incorporated in the method. ‘Customer Interface Management’ practice area will also change according to what method we adopt. Challenges with this pattern could be the launch, changing the organizational planning and operation effort, introducing small releases to reduce risk, and incorporating the agile way of thinking. The launch is critical since we need to introduce two new ways of working, with minimizing reduced productivity and communication the change to the whole organization.

The ‘Process’ pattern would mainly do the ‘Process Discipline’ practice area which we have discussed (Section 5.3.2), with input from the involved practice area as described in (Section 2.1.3). The main points from these discussions describe a process as: iterative; lightweight; easy to learn; open to change; and listening to the environment. The cases described in this study elaborate on these process and we found that they are individual based on the need for the companies, but they are iterative and handles change as it arise. Challenges with the process are: to come up with good processes that fit the organization; handle the practice areas in lightweight processes; and being able to combine long-term strategic planning with short-term agility. A compromise with regards to Agile Software Development and its principles would be to involve the higher levels of product development like market analysis and domain knowledge because Software Product-Line Engineering is too complex to handle everything on code level. This concurs with the main challenge of 'Process Discipline' as well, namely maintaining the long term strategic focus of the Software Product-Line Engineering while benefiting from short term flexibility. We have seen from the results of this study that the companies were able to combine long term focus with short term agility.

The 'What to Build' pattern would have influence from Agile Software Development in what it is supposed to have as output. Depending on the Agile Software Development method and practices this pattern would have to run iterative in longer time boxes meaning that the output from this pattern should not be static, but change according to the environment and results obtained. Challenges here would be to abstract enough so that entities taking over the output are not limited because of high details in their input. The 'Analysis' variant is a heavier version of the original pattern, and to make that more agile we have to try to include the agile requirements through epics, features or user stories and architectural efforts with architectural runway to reduce the design up-front ergo making the pattern more agile. Also this would be going on in longer time-boxes. 'Forced March' is the more lightweight variant of the pattern which takes advantage of legacy systems to find the scope and is not influenced much by Agile Software Development. In this pattern we think that the pattern could combine the pattern and its variants in a lightweight way to establish the scope like we saw in the PROSOL case described in this study.

On the product level of the context it is hard to choose whether to use the 'What to Build' pattern, one of its variants, or the planning technique used in the PROSOL case. If there are legacy systems to support creation of the product plan of the software product line this could be a natural choice. If not the 'Analysis' variant of the pattern or the PROSOL technique could be the best choices. We base that on the need to start developing early and deliver value. In the original pattern the effort is towards marketing challenges more than development and we think that contradicts *"working software over comprehensive documents"* from the Agile Manifesto. Instead, starting early with requirements and architecture could fit Agile Software Product-Line Engineering well and could prove as a beneficial compromise between Software Product-Line Engineering and Agile Software Development. Other challenges here are to balance the detail level of early requirements and architecture so that the developers can use their creativity later and have enough business value to move forward with the Agile Software Product-Line Engineering adoption.

In the figure we chose to put the ‘Analysis’ variant as a guideline to contextual product efforts.

Production Capability

The ‘In Motion’ pattern is a managerial pattern on the organizational level to ensure that the progress of a product line is satisfactory (Section 2.1.4) and we found little evidence on how to do this more agile except that the manager has to manage the agile method instead of a more regular approach to product lines. Challenges in this pattern are related to the separate practice areas (Section 5.3): ‘Operations’; ‘Training’; ‘Customer Interface Management’; ‘Acquisition Strategy’; and ‘Organizational Structure’. An Agile Software Product-Line Engineering approach should also be able to see results earlier and continuous in addition to getting feedback from iteration retrospectives like HomeAway does in their Scrum method. In addition we would introduce Agile Software Development measurements on development efforts to assess team and organizational velocity.

On the process level the ‘Assembly Line’ pattern could become more agile by setting up for continuous build, test, and integration described in many of the cases in this study. HomeAway described that they used some time to establish their assembly line. IFS used a different approach in addition to the practices mentioned above, where they developed product parts then had quality assurance in a separate team, before distribution was done by another separate team. Another approach is to let tools automate the assembly line, which was described in our cases through the Gears software. This meant that the developers only did domain engineering on the core asset base, with establishing variation points and a feature model. These could be to suggestions to handle this pattern and its agility. A compromise between Software Product-Line Engineering and Agile Software Development here is that we have to minimize work towards the core assets that does not make any value, meaning that we have to do “just enough” so that the application engineers are able to put together products from the platform. Challenges here might be choosing the tools to use and reducing the details so that we increase creativity while being open to change.

On the product level of establishing production capability we need to do two patterns where the ‘Each Asset’ pattern gives the ‘Product Parts’ pattern its components. For the agility of these patterns we introduce a new hybrid pattern ‘Parts Factory’. This will be explained after we look at the two included patterns. Looking at the ‘Each Asset’ pattern Agile Software Development will introduce a new ‘Technical Planning’ practice area through the method the organization chooses to use. The ‘Testing’ practice area will be done concurrent or before actual development on the asset, while the ‘Process Discipline’ practice area could take a lighter role, maybe some short documentation in a wiki or similar. The ‘Configuration Management’ practice area would be automatically handled, while ‘Data Collection, Metrics, and Tracking’ and ‘Tool Support’ practice areas would inherit agile practices in addition to Software Product-Line Engineering practices as described in their respective practice areas above. The challenge for agility here is to perform Test-Driven Development in all asset-related realizations and have a “just enough” emphasis on the process discipline. The agile ‘Technical Planning’ practice area would introduce new ways of doing requirements, architecture and components, but we think it is feasible and we saw that the companies used user stories and features or epics as requirements. The variations would have the same comments and change according to Software Product-Line Engineering and the main pattern. We saw that asset development was done differently in our case companies, but both IFS and HomeAway had an iterative way of developing an asset with agile focus and fits well into this pattern. A compromise between our two approaches for this pattern would be that we need to handle to variations in some way like we described many times earlier. Challenges here would be to follow an agile method, “just enough” emphasis on non value creating tasks and handling variations.

The composite pattern ‘Product Parts’ handle putting together separate assets so that they can become parts within the platform to form a product. To make this more agile we introduced the hybrid approach since Agile Software Development is about delivering working software and avoiding silo work on a product. We have discussed using multifunctional teams for building complete assets for the platform instead of having silo-based development of requirements, architecture, components and test artifacts. Taking input from the ‘Organizational Structure’ practice area and adjusting this new hybrid pattern to the team-based structure could be beneficial for the agility and cooperation in the creation of core assets or product parts. A compromise between Software Product-Line Engineering and Agile Software Development with the hybrid approach is the decision between making in-house, buying or use existing software or legacy systems. This should be done in order to maximize the value of the core assets. Challenges with the new hybrid pattern could be validation towards analysis, variation handling and leading the teams to efficient work.

Operate the Product Line

The ‘Monitor’ pattern fits well with Agile Software Development, but has a broader scope since we deliver to several customers in a larger environment than in single-system development. Further, we see that the output of this pattern is plans and processes which contradicts with the Agile Manifesto. Here we could imagine a more direct feedback process directly to the concerned stakeholders instead of going into planning and process building again. We saw a variant of this pattern in the Testo AG case described in this study which was more agile than the regular approach. To make this pattern more agile we would introduce the agile metrics, assign product owners to each component to enhance them according to need, have some feedback from the products developed and do iterative work on evolving the product lines platform. The pattern would change to do more direct feedback instead of going through organizational hierarchies. The compromise here could be to document enough so that we have control over changes with regards to the product line and its variations. In our results we saw that DNV Software collected feedback from its product development departments to trigger changes in the platform. IFS used feedback from the distributed development departments who were close to the customer in addition to user groups who provided feedback on the products they used. These could be good examples for monitoring. A challenge here is to filter feedback so that we follow the scope of the product line, we could probably not allow every change request.

When delivering product the ‘Product Builder’ pattern can be made more agile with automation like the use of Gears by HomeAway, Salion and Engenio. A fully automated production process or tool which builds and test the software can be the mantra for an agile approach here. Basing that on a feature model was shown to be very powerful for the case companies already mentioned. IFS built their products distributed, closer to their customer, which is also positive because of the customer interaction and delivering value. We think it can depend on the situation and type of software products you are building. IFS follow the pattern but with more agility and are close to their customers. The first case companies mentioned, had predefined variations and used a variant of this pattern ‘Product Gen’ which can more easily be automated. Challenges here are being able to deliver value to the customer through the core asset base and adopt automation or an agile method to make the pattern more agile.

5.2.3 ***Scenario 2 (S-2)***

In this sub section we describe how an organization could adopt Agile Software Product-Line Engineering with existing experience from building product lines. We start with the establishment of the context, and then explaining how the production capability could be built before we handle the operating of the software product-line. In the patterns where this scenario handling is similar to S-1 we refer the reader to the already explained pattern above.

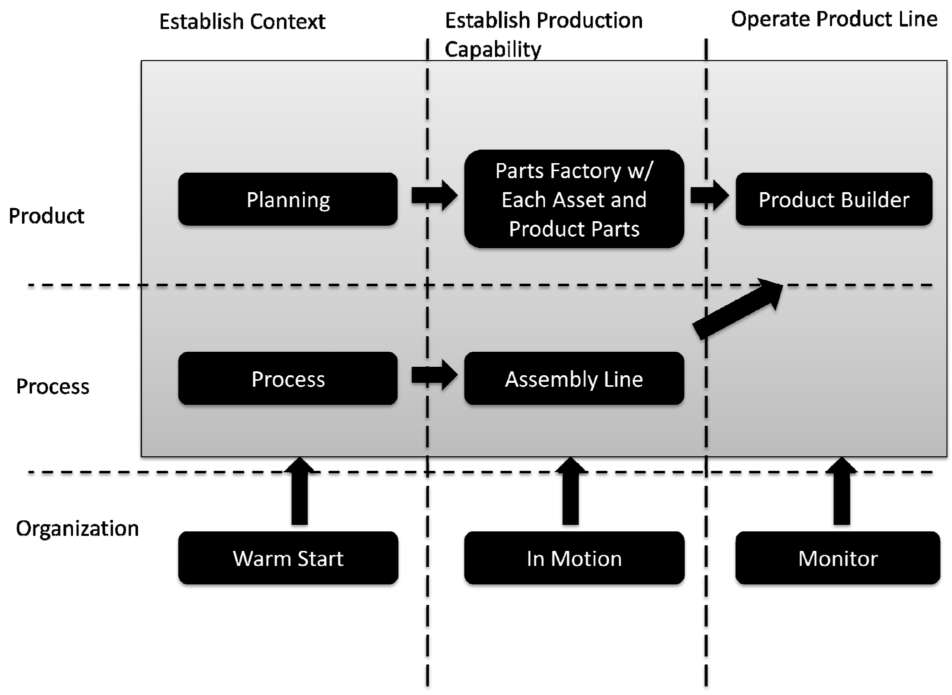

Figure 6: The Adoption Factory pattern for scenario 2 modified to fit an Agile Software Product Line. (modified from Clements et al. 2006)

Context

The 'Warm Start' variant pattern for experienced companies which would need to redefine 'Organizational Planning' and 'Operations' practice areas if they are going from a regular approach to an agile approach. Even though the variant does not include the 'Training' practice area it would be beneficial to introduce and train on Agile Software Development since it would be a new concept to the organization. The challenges in this scenario would be to communicate and learn the organization about change in addition to plan the new product line.

The 'Process' pattern would have a different perspective since we need to change from the Software Product-Line Engineering processes already defined to more agile processes. In this change we could use points discussed in the 'Cold Start' pattern and the 'Process Discipline' practice area. Challenges are the same as in these entities, but this could meet stronger resistance within the organization because people could be reluctant to change something that is already working.

To find out 'What to Build' for the new product line we think that the lightweight version described in PROSOL could be enough since there is already experience and artifacts from product line(s) existing that could be used to set the product visions.

Production Capability
The ‘In Motion’ pattern for S-2 is different because we need to adjust to the agile method introduced. Extra effort should be put in training and ensuring that people develop using Agile Software Product-Line Engineering instead of holding on to the old style of development. The points discussed for S-1 on this pattern is also viable here. To establish the ‘Assembly Line’ pattern for the new product line we could build on what already exists, built we need to consider the test and build focus of Agile Software Development as described for S-1. ‘Parts Factory’ would evolve on existing assets in the core asset base and add or adjust these to reflect the new scope or needs of the new software product line. In addition points from S-1 should be taken into account when adopting Agile Software Product-Line Engineering.

Operate Product Line
In the ‘Monitor’ pattern we need to introduce agile metrics and feedback like described in S-1. There are no specific needs for S-2 in this pattern, but as with other patterns there might be something to build upon or change in order to establish the pattern faster. ‘Product Builder’ would also stay similar to S-1. We should probably discard the existing practice here and introduce new practices that are more agile to reflect to the changes in other parts of the organization.

5.3 Combining Software Product-Line Engineering Practice Areas and Agile Software Development Practices

To describe practices that were used to obtain agility in Software Product-Line Engineering (RQ-1.5), we investigated our results and crossed the practice areas with the agile practices in a one-to-many relation (Table 12). Further we elaborate on the results in each practice area, and describe how the practice area can become more agile according to the results of this study. We cover the general thoughts for the practice area, but also mention special applicability towards the Software Product-Line Engineering adoption approaches.

Table 12: Product Line practice areas with the Agile practices that could be used in combination.

Software Product-Line Engineering practice areas	Agile Software Development practice
Software Engineering	
Architecture Definition and Evaluation	Domain-Driven design and Initial Architecture
	Refactoring
Component Development	Vision, Roadmap, Elaboration
	CRC cards and design by contract
	System metaphor and coding standard
	Refactoring
	Automation; daily builds; continuous integration
	Code ownership
Requirements Engineering	Domain-Driven design and Initial Architecture
	Vision, Roadmap, Elaboration
Testing and Software System Integration	Test-Driven development
	Pair programming
Understanding Relevant Domains	Vision, Roadmap, Elaboration
	Planning and prioritization
Technical Management	
Configuration Management	Automation; daily builds; continuous integration
Measurement and Tracking	Visible charts and information radiators
	Metrics
Process Discipline	Iterative development
Scoping and technological forecasting	On-site customer
Technical Planning and Risk Management	Small, frequent releases
	Code ownership
Tool Support	
Organizational management	
Building a Business Case and Market Analysis	Vision, Roadmap, Elaboration
Customer Interface Management	On-site customer
Launching and Institutionalizing	
Operations	Iteration backlog or Kanban
	Continuous improvement, reflection and retrospective, root cause, and learning
	Stand-Up meeting
Organizational Planning, Risk Management and Structure	Small, frequent releases
	Iterative development
	Cross-functional teams and self-organizing teams
Training	Continuous improvement, reflection and retrospective, root cause, and learning

5.3.1 *Software Engineering*

This section goes through the practice areas of Software Product Lines described in (Section 2.1.3) and discusses the agile practices and how to use Agile Software Development in combination with the respective practice area based on our research results. The structure follows the same pattern as we presented both the theory (Section 2.1.3 & 2.2.3) and the results (Section 4.2.1 & 4.3.1).

Domain and Requirements
Domain knowledge supports the main idea behind Software Product Lines. The companies are described to have an overall knowledge in the field they are delivering products. No agile practices is similar to this practice area, but the *Planning and Prioritization* practice could support this practice in the way that customer feedback and domain experts play as a customer to an agile method and helps plan and prioritize the efforts put in development. This could make the product line more responding to the current needs in the domain, and making the product line more able to change. It also concurs proactively work towards change. Further, Agile Software Development handles domain related challenges stating that the team has to handle them, which we saw could be a problem as in IFS. The challenges we can see here is balancing the practice area with agility meaning to spend “just enough” effort on this issue and incorporate it into specific specialized teams for the domains. Benefits from domain knowledge could be better software and less rework because of wrong domain knowledge.

With regards to requirements Agile Software Development often does requirements concurrent with implementation, while Software Product-Line Engineering put great effort in modeling and ensuring that the requirements are correct and valid. Our results show that most of the companies have a way of handling a high-level model of the system. We believe that it is important to be able to see the direction and future of the products produced in the product line. We also see that business cases seem popular to support the business side of Software Product-Line Engineering, and requirements through user stories and further elaborating to task are possible here and were used in the companies of this study. Further an agile method would support more lightweight requirement engineering based on the customer need, which could be beneficial in dynamic markets where needs change fast. We think a choice of process would define the agility of an approach here. This challenge is currently being researched by Cooper[27]. We could also adopt techniques such as combining user stories into features or minimal marketable features, and further features into epics or vice

[27] http://www.utdallas.edu/~kcooper/research/research.html#Agile

versa with epics first. This could be controlled in different levels of the organization which will be more closely described later (Section 5.3.3).

Challenges in combining the two SE approaches can be control over variations and dependencies among products, getting feedback from application engineering to domain engineering and customer interaction. There is a need for variation control to be able to describe which product the product line is able to produce. Collecting feedback from application engineering is important for the evolution of the platform and customer interaction has to be redefined because we are trying to produce for many customers. Benefits could be reduced design up-front, less documentation to maintain and better communication between business and development. Generally, we saw less design up-front from the companies in our case study and focus on just needed documentation were indicated.

Architecture

When it comes to the architecture in a Software Product Line the need for a sustainable, flexible architecture is high. This could be hard to obtain following true agile lightweight architectural efforts where design is done more ad hoc in extreme cases. From our multi-case study we can say that the companies put extra effort in getting the architecture ready before producing products, but in a lightweight manner. As an example, IFS do some changes in higher-level development because it is more beneficial to do it before actual coding starts. Leffingwell (2007) mentions the architectural runaway as an important practice towards a scaled approach to Agile Software Development, and we mean that these thoughts fit the combination between Agile Software Development and Software Product-Line Engineering. We mean this because the extreme agile approach would be hard to do in practice for Software Product-Line Engineering, who again might be too detailed on architectural design. We say this based on the generalized results from our analysis and this concurs somewhat with (McGregor 2008) who state that we could replace the existing practice with a new one.

To support the architectural runway domain knowledge usually existing in Software Product-Line Engineering, and could further support the creation or evolution of the runway. We also found that the companies we studied mostly used existing products or platform to establish new functionality or versions. This could be further combined with the refactoring and team empowerment from Agile Software Development. The component-based architecture used by the companies fit well to both Software Product-Line Engineering and Agile Software Development. In Software Product-Line Engineering it means we can use different components in different products, while in Agile Software Development we can separate the development effort to component teams which are also advised in (Leffingwell 2007). This is also supported by Lean theory were it is stated that design decides about 70 % of the cost in manufacturing, therefore appropriate architectural effort is needed for software engineering as well (Mehta et al. 2008).

Challenges with the combination in this practice area could be the amount of architectural effort spent before the teams can develop. In addition, we cannot make generalizations about a new product line's architectural definition since all of our case studies handled an existing architecture. Benefits that could be obtained by combining agile architectural thoughts are less design up-front because the implementation can start earlier.

Development

In development we need to handle variations to cope with Software Product-Line Engineering. We found that the way a company does component development vary according to size and domain. It is hard to uniformly generalize any of our results in this area, but we try to imply some ideas in this practice area. We saw that Gears helped three of the companies by easing the production of components and putting them together. The overall theme is that components need to be developed separate with as little cohesion as possible. When it comes to agility we mean that coding practices from Agile Software Development such as coding standards, refactoring, code ownership and continuous integration can make component development more agile. The process discipline and test-driven development also act towards the agility of this practice area. There are also two levels of development, domain and application. In our study, three companies used Gears and mostly avoided the application engineering with automatic production of products from the core asset base and Agile Software Development in development of this base. IFS and DNV Software used separate domain and application teams, where IFS tried to do domain engineering in an agile manner. DNV Software on the other hand was currently experimenting with agility in their application departments.

Challenges with the combination in the component development practice area would be changing the culture to agility, getting feedback communicated across the agile teams and new type of management (encouraging instead of directive). Benefits of combining Agile Software Development and Software Product-Line Engineering in component development could be more efficient development and higher quality based on the agile benefits. Test-Driven development is also keyword in for this topic. It seems like our companies is up to date with Test-Driven development and Agile Software Development relies a lot on testing early instead of testing after coding is complete. Variations in Software Product-Line Engineering do introduce some challenges in test coverage and traceability, but seems to be solved in the companies' approaches and research on that topic also exists (Ghanam et al. 2008). One of the latest contributions (Ghanam & Maurer 2009) to the field is an idea and case study explaining how to combine feature models and acceptance testing. Automatic build with continuous integration running test suites are a natural companion to Test-Driven Development. Pair programming and code review are also practices that could be emphasized here to improve quality. Benefits with Lean related to automatic and small builds like reduction of complexity, reduced bugs and increased efficiency is also argued for (Mehta et al. 2008) and support our findings.

Challenges that arise with introducing test-driven development is a change of focus towards testing, handling variations of products and automation of core asset testing. The cultural change can often be an issue when adopting agile practices. Variations make trouble both with reusing test assets for products and automation of tests in core asset base. Component-based development should make this easier. Benefits with the test and continuous integration could be higher quality, fewer defects, and reduced problems with integration.

5.3.2 *Technical Management*

This section goes through the practice areas of Software Product Lines described in (Section 2.1.3) and discusses the agile practices and how to use Agile Software Development in combination with the respective practice area based on our research results. The structure follows the same pattern as we presented both the theory (Section 2.1.3 & 2.2.3) and the results (Section 4.2.2 & 4.3.2).

Scope and Technology

Scoping and technology forecasting are practice areas that decide which products we should include in our software product line, and forecasting the technology changes in front of us. In Agile Software Development this is done at a high-level through user stories or other planning efforts, and is done repetitively in a project. We found that the companies use roadmaps of future implementations to project the scope of the product line, and their technological platforms were module-based and they could switch parts of technology to cope with new technology. Some of the companies had own sales and marketing departments that were responsible for creation and maintenance of the roadmap. Feedback from customer can also be important here, and a more agile approach could combine the On-site customer practice with scoping. We saw that user groups, stakeholder groups, and similar practices were used to get this feedback in the case study companies. We think that a practice involving this feedback and technical investigation has to be merged from the two approaches.

The challenges in our combination for scoping and technology forecasting could be assigning the right ‘customer’ for feedback towards scoping. In addition the companies have to be updated on the latest technology. In Agile Software Development scoping is done by the customer and organizations should trust its people on technological challenges. In a product line effort we probably have to use a higher level scoping based on business needs and technological challenges combined. The benefits of combining Agile Software Development and Software Product-Line Engineering for scoping and technology forecasting could be that the customer focus could enhance the correctness of scoping, while motivated, learning developers could deal easier with technological advances in the industry.

In our analysis results we saw that the Gears tool supplied many of the companies with an efficient way of handling variations. From this we can say that it could be wise to have a tool to handle variations, in order to reduce work and raise quality. HomeAway also used an agile management tool with success. Besides that best of breed software for the respective domains were shown to support the companies and increase effectiveness of development.

Challenges for this practice area would be to combine the tools of Software Product-Line Engineering and Agile Software Development. Using lightweight tools as Agile Software Development suggests could also pose as a challenge since Software Product-Line Engineering is a comprehensive approach and tools could be heavyweight. Benefits using tools for both approaches could be more efficiency and automatic creation of information about the progress.

Process

The choice of process has dependencies throughout the software development effort. The practice area in Software Product-Line Engineering does not prescribe a default process to use, but describe that it needs to handle how the core assets should be reused in addition to the regular responsibilities of a process. In Agile Software Development the process is lightweight, and should be easy to learn. We think that iterative development is the way to go, and we found that in most of our case studies. An agile process should be able to change and improve based on the environment it is situated in. This is possible in Software Product Lines, but might need a little more structural explanation. We found that some companies have developed their own process to cope with this like IFS and CompNN. They are both able to combine a long-term strategy with a short-term agility the way we see it. HomeAway also has their process with parts of the process as a Scrum method done out-of-the-box. A hurdle for the adoption of Software Product-Line Engineering is the big design up-front cost and effort. This hurdle can be jumped by employing an iterative process which builds core assets and products more incremental and use knowledge in parallel. It could also lead to less cost, since working software should be developed and can create money back from early products before the whole product line is up and running. Challenges with the process could be to incorporate Software Product-Line Engineering and long-term strategic goal with the short term development efficiency of Agile Software Development, including marketing and business efforts in the process, and cope with the multi-dimensional nature of Software Product-Line Engineering. We base these challenges on what we found in our multi-case study. Benefits of combining the two approaches for the process could be reduce design and cost up-front, more flexibility, and shorter time to market.

Measuring and ensuring progress are important in both Software Product-Line Engineering and Agile Software Development. In product lines we want to measure how much we reuse, efficiency of the core assets and time spent on product application, while Agile Software Development focus more on team speed, productivity and estimation of effort to measure and track a project. As an example of how this is often done in Agile Software Development we could mention estimating user stories, calculating team velocity, and creating burn-down charts based on the team or several teams. Here a natural practice would be to combine both fields. Challenges here are to evaluate which measurements we use and how we can get the most out of what the process we are using gives us from before. Benefits could be easier management and statistical improvement opportunities.

Management
It is common sense, in both Software Product-Line Engineering and Agile Software Development, to try to separate development projects so that they interfere as little as possible. This could the initial thought within configuration management. Software Product-Line Engineering has some additional concerns regarding variations and might need more strict rules than single-system development including Agile Software Development. A true component-based development might solve this without too much intervention, but also some sort of cooperative solutions among the teams have to exist and changes to important points in the software might interfere with other teams or individual works. We saw that some of the companies had efforts towards this, but no generalizations could be found. We also think that introducing agile practices like automation and daily builds with test suites and integrations could cope with the configuration management. These practices were implemented at most of the companies we investigated. Challenges we can think of here is maintaining a uniform code base without people stepping on each other’s toes and having rules that apply, but not reduce the freedom of the teams. Benefits with combining Software Product-Line Engineering and Agile Software Development could be more automation and less manual work.

The technical planning involves how we plan a project and risk management is how to deal with risk in the technical parts of the Software Product Line. In Agile Software Development the technical planning is done before an iteration, where the user stories are elaborated into tasks and estimated to measure and track a project. Risk management in Agile Software Development is usually reduced by small, frequent releases with customer interaction so that we know we build the right thing with few errors in a timely manner. Code ownership could also be described here and cross functional team reduces overhead in planning. Stand-up meetings could also be introduced in this practice area, and the companies used this agile practice in their approaches with success. Challenges regarding technical planning will be following the process discipline and its planning effort, while risk management is incorporated in the iterative approach to some extent. Benefits of the combination would be less up –front planning and reduced risk management based on the Agile Software Development practices that could be combined with the practice areas.

5.3.3 *Organizational Management*

This section goes through the practice areas of Software Product Lines described in (Section 2.1.3) and discusses the agile practices and how to use Agile Software Development in combination with the respective practice area based on our research results. The structure follows the same pattern as we presented both the theory (Section 2.1.3 & 2.2.3) and the results (Section 4.2.3 & 4.3.3).

Environment

Market analysis and building a business case is important for both Software Product-Line Engineering and Agile Software Development, but in product lines the effort might be more planned and larger since we are supposed to handle a set of products not only a single-system. We found that the companies can have their own departments for marketing efforts, and the importance of connecting business cases to software development is present. Connecting development to business cases is vital, and Agile Software Development can be described to handle this by user stories applied in both business and development. In Software Product-Line Engineering user stories might not be enough because it might be hard to present variations in user stories and a need for more high-level business cases or feature models are present. The vision, roadmap and elaboration practice described could be included here. The vision could be producing business cases and the roadmap includes the business cases as a plan for what functionality we get when. Last, the elaboration breaks the business cases down into development projects. CompNN, HomeAway and IFS had similar practices in place for this. Challenges here are to cover all the necessity tasks of Software Product-Line Engineering in an agile way and connecting agile software development to the business part of the development effort. Benefits of this combination could be easier communication across functions, more flexibility and embracing change.

When it comes to customer involvement and interface the fields of Software Product-Line Engineering and Agile Software Development are different. Software Product-Line Engineering needs to listen to their specter of customers and adjust their core assets and finished products with regards to their need. Agile Software Development usually handles one customer and does single system development. Here a need to adjust the agile practices to Software Product-Line Engineering is needed. We found that two companies use internal resources as customers in projects, and management decided high-level choices. Efforts were also done to listen to the main customers and adjust the product line based on their need and future predications. We think this is important in a combination between the two software engineering fields. The "On-site customer" practice of agile could be modified to fit Software Product-Line Engineering here. We also found that application engineers in some companies provided feedback to domain engineers and could be seen as a customer to domain engineering efforts because they are reusing the core assets. Challenges combining Software Product-Line Engineering and Agile Software Development in this practice area could be using a representative selection for the customer interaction and introducing the customer role inside the company. Benefits of this combination might be more accurate aligning with regards to the customer base and more direct communication inside the company.

Indoctrination

Both Software Product-Line Engineering and Agile Software Development usually involve big organizational changes. Software Product-Line Engineering need to be established throughout the organization and a culture for planned reuse and development based on a set of products are different from single-system thinking. In Agile Software Development we experienced from the case studies that HomeAway had difficulties in establishing the Scrum method since people were used to working differently. We believe that an incremental approach to Software Product-Line Engineering could be beneficial in order to prepare, communicate, and be supported in the new approach. This type of approach also fit better with many of today's domains who are dynamic and constantly changing. Agile Software Development have been indicated to be popular amongst software developers, but still challenges regarding management and the business side of software development is present. HomeAway and DNV Software both experience these kinds of challenges. We also think that a combination between Software Product-Line Engineering and Agile Software Development could be more beneficial for organizational support and motivation since it could earlier show the results of the product line effort with using agile methods. Challenges in this practice area are connected to organizational change. One would have to change people's practices and way of doing things, into the new process and thoughts. Another challenge would be incorporating the reuse thoughts to Agile Software Development and its creativity. We have to put some kind of boundaries on creativity in order to be able to produce many products of high quality. Benefits in this practice area could be ability to change and respond to a changing market, and exploiting reuse efficiently.

When it comes to training both the fields would need to be thought and adopted by the employees in the organization. Introducing Software Product-Line Engineering means building a core asset base with support for variations and delivering a set of software products from one platform. Agile Software Development should also be incorporated in the whole organization and management will have new challenges in measuring and controlling progress, business have to make looser plans that supports change, while developers should master Agile Software Development and its practices. We saw from our case studies that the companies use time to adopt to new ways of doing work and both Software Product-Line Engineering and Agile Software Development need to train employees to make them master the new work practices. Challenges when it comes to training could be training the right people for the right jobs, training experts in Agile Software Development and Software Product-Line Engineering, and continuous learning in the new process. Benefits of combining Software Product-Line Engineering with Agile Software Development in this practice area could be positive effects from the continuous learning in agile, more focus on learning organization, and more knowledgeable employees.

Organization
The practice area involves how we plan organizational efforts, manage risk and structure the organization. The organizational structure of a combined approach would have to change according to the multifunctional, self organizing, component teams we want to have in Agile Software Development. The planning would also change to coordination instead of directing since we need to manage several teams. A traditional Software Product-Line Engineering approach separates works in silos having specialized teams doing the traditional elements of software engineering. Agile Software Development tries to create teams who cooperate and deliver working software in the end of each iteration. Risk management is again handled by small, frequent releases and iterative development. The team practice of Agile Software Development is also participating in reducing risk and should be self organized meaning that the team decides how they want to develop when they have the prioritized plans. Scrum would be a natural starting point in this practice area and we saw that

HomeAway, IFS, and DNV Software did a variant of this method to control projects. Challenges at this level are situated around scaling agility and risk handling. Agile Software Development are mostly proven beneficial for small teams and projects, while Software Product-Line Engineering usually involve a bigger effort and more employees. Risk handling challenges could also evolve since agile is proposed for single-system development. Benefits of combining Software Product-Line Engineering and Agile Software Development in these practice areas could be reducing risks and supporting component-based architectures with small teams.

Regarding day-to-day operations the process and its methods decide much of this. There are some tools from Agile Software Development that could be tried out here. An iteration backlog or Kanban is a nice way to structure the work that should be done and the agile methods include this. Our case study also showed that companies used kind of a backlog. Stand-up meetings are also a practice which are used every day and improves communication and task handling from day to day. Last from Agile Software Development we can introduce continuous improvement, retrospectives, and learning as practices for day to day operation. Feedback cycles and cooperation between business and development should also be considered here. How software is reused or produced should also be described as part of operations and wiki could serve as a mediator here, and were used by industry as well. Challenges in operations could evolve around changing from a documented and stable process to a dynamic, changing process. Instead of following a plan we would respond to changes. Benefits of this combination could be motivated people, continuous improvement and efficient communication.

5.4 Threats to Validity

In this section we try to address the threats to validity of our research. Seaman (1999) introduces several strategies to reduce the threats to validity. In this section we explain how we have utilized these strategies to address the validity of our thesis. We have divided the reasoning in two parts, namely ‘Data Gathering’ and ‘Data Analysis’, reflecting how the research method was explained.

5.4.1 *Data Gathering*

In order to collect data material for our study we used a search strategy to find articles and performed interviews with people from the industry. The interview setup worked well for our work, but we had some minor problems with the quality of the audio in shorter periods during the interviews. Unfortunately, minor parts of the statements were lost, but asking again and ensuring we got everything prevented us from making assumptions that might not have been true. Further we tried to have an objective role as listener in the interviews, and not being religious towards one or the other approach while coding the cases. This threat is characterized as coming to close to the setting, and called *objectivity* (Seaman 1999), but we think we managed to avoid this threat by having an open mind towards both approaches to software engineering. We also included data from different sources including the interviews much like *triangulation,* described in (Seaman 1999), to confirm the validity.

In addition to this we also described our perceptions about the quality of each the published articles used for the case studies, and found it natural to mention it here since it is a threat to our overall validity. We based our rating of quality on the data material used in the articles and how the study of each article was performed. **CompNN** was a single-case study based on semi-structured interviews of the main roles in the company (Hanssen & Fægri 2008), we see the quality of this study as high since the research method is clearly stated and agility is discussed. **Salion**'s case studies analyzed are performed by researchers and practitioners. The most scientific case study (of the two), (Clements & Northrop 2002b), base their description on two sets of interviews performed with nine months separation to be able to reason about the success of the approach. The second case study (Buhrdorf et al. 2004) is more towards an experience report from the company and the researcher helping the company out. Nothing is mentioned about how the study was performed and little discussion about weaknesses is mentioned. Based on the explanation above the two case studies provide satisfactory quality for us.

In the **HomeAway** case we used two articles and interview. The articles analyzed also promote a tool, and could not be classified as objective. We tried to broaden the scope by looking at two articles instead of one, and the description above provides the combined description of them. We also conducted an interview with the company to further increase the quality and the combination provided support towards the findings from the articles. All together we mean the quality is high. The **Engenio** article does not discuss the validity of the study and we only used one case study. It is subjective because of its practitioner nature, but again as our scientific, empirical research on this field is spare we choose to use the case study (Hetrick et al. 2006) and it provides satisfactory quality for us. The quality of the **IFS** case can be described as satisfactory. We did an interview and got several documents from the company that made it easier for us to obtain an overall view of the combination of Agile Software Development and Software Product-Line Engineering in this case. The threats are interviewing only one person and the interviewers little experience with scientific interviews. Still we believe that we came in contact with the right person for our research and he was able to fluently explain their software engineering efforts. The quality of the **DNV Software** case study can be described as high. The book chapter was written with a defined method and the data collection followed a certain pattern. In addition, we strengthen our case with an interview and follow-up questions to renew the knowledge and include agile thoughts that are starting to emerge in the company. Overall, we could classify the **Testo AG** case as a design science study according to (Hevner et al. 2004). The researchers want to establish and test a new practice in the Software Product Line field. The practice builds upon an agile practice, but little discussion around the actual agility of the approach is discussed. The efficiency and ease of use is mentioned, but not compared to agile theory to a significant extent. This reduces the quality, but we use it because of the lack of data in the field. The **PROSOL** study was conducted as a kind of action research approach with the researchers participating and observing the workshop, trying to answer two research questions. All the stakeholders' roles were covered, but it is a single-case study that does not compared its results to the regular practice. Actually the research is similar to the Testo AG case study described, in the sense that it could be characterized as design science study with a case study to back up the need. Based on the discussion above the quality of this study is satisfactory.

5.4.2 *Data Analysis*

After collecting the data material, we performed a thorough analysis using several techniques to reduce the threats to the validity of our research. Our research design was a combination of research methods, and we combined established articles in form of case studies with interviews. The *representativeness* of the data material and interview candidates can be discussed. Choosing cases as the study evolves and picking knowledgeable interview candidates are mentioned as strategies to support representativeness (Seaman 1999). We believe that our approach is valid based on the lack of material in the field and by following guidelines for empirical studies. We established a framework for our study including criteria and templates before we started our work. The process of choosing cases was probably not unbiased, but we did not have many options. Based on earlier research we knew about several of the articles used for the case studies. The industry contacts were obtained through our network and the described efforts towards relevant communities. An alternative to our research would have been to do a more comprehensive study about interesting cases from the industry and develop an even more generalized framework, but because of limitations in time and geographical distances we were unable to pursue this idea.

The textual coding and analysis of written articles was performed subjectively by one person following a research method making our research qualitative. Threats towards documenting what is right and perception of the interviews are biased in the way that other people might understand the articles in a different way if they try to *replicate* our study. We are aware about this threat to the validity. *Member checking* (Seaman 1999) is another strategy to validate the data and we used it in this study to support and validate our findings and reduces the subjectivity threat. Our discussion qualitatively elaborates what we found in the study and documented these findings. According to Seaman (1999) our study can be described in the *blocked subject-project study* category because we analyzed several projects in several companies. We tried to compare the efforts of the companies and find similarities to establish a framework for introducing Agile Software Product-Line Engineering. To improve our research we could have used quantitative methods as well as the qualitative method we have described (Seaman 1999), but unfortunately we did not have the opportunity or time to do this.

6. Conclusion

This thesis has elaborated on Agile Software Product-Line Engineering, the combination of Software Product-Line Engineering and Agile Software Development. We started with a brief introduction to the two fields and the hybrid approach. Then our analysis results were presented before we discussed possible solutions to our problem areas. This discussion presented some characteristics and a framework for Agile Software Product-Line Engineering, in addition to illustrating how the practices of both fields could be combined. This chapter holds a precise conclusion of this master thesis summarizing the important points and findings of our work.

Recalling our two main research questions RQ-1: *"How is Agile Software Development combined with Software Product-Line Engineering in industry today?"* and RQ-2: *"What could characterize a method or a framework to describe agility in software product line engineering?"* we had several sub-questions to be able to answer the main questions. In this thesis we have answered all sub questions in chapter 4 and 5, and are able to give an overall answer to our main questions in this conclusion.

We analyzed eight studies were six were based on published work while two were own empirical work. Based on the results of this analysis we presented how agility was introduced in the industry and discussed how each practice area of Software Product-Line Engineering could become more agile. Our answer to RQ-1 is therefore documented in chapter 5, section 5.1 and 5.4. The main findings here were that companies reduce time to market, improve quality and exploit reuse through combining Software Product-Line Engineering and Agile Software Development.

RQ-2 was answered through some characteristics of agility in Software Product-Line Engineering and a top-down description on how we could use the patterns of Software Product-Line Engineering to establish Agile Software Product-Line Engineering from no experience in Software Product-Line Engineering or an existing Software Product-Line Engineering effort.

The main points from our discussion indicate that:

- We can reduce unnecessary documentation and up-front design.
- Multifunctional, self organizing, component teams could be adopted.
- An iterative method for elaborating design, code, test and review can be established.
- Manage, coordinate and feed the teams with new abstract requirements to increase changeability and creativity.

Our main contributions to the research field are the characteristics and the proposed framework which also uses discussion in each practice area since the patterns consist of various practice areas. We think our research indicates that the practices of both software engineering approaches could be combined and described a possible combination. However, other people might have other opinions and perceptions of this combination.

The framework is not supposed to be a method or standard, but work as a guideline and points to consider when combining Software Product-Line Engineering and Agile Software Development into Agile Software Product-Line Engineering. We hope this contribution can be further discussed and built upon by both researchers and industry since it is in no way finished and we only touched upon some important points. Research on specific methods or practices of Agile Software Product-Line Engineering could be included in the framework when they are established.

We think our results in this thesis could help further discussion about Agile Software Product-Line Engineering and display some of the ways the companies do the combination as of today. In research, no evaluations of agility among companies using Software Product-Line Engineering have been done before and this thesis could help us to better understand what problems and solutions we are facing. In practice this study can help companies evaluate their own approaches and combine elements from parts they like in this study and make their own hybrid method between the two software engineering fields. We should note that the results of this study are by no means complete since we study only a limited set of articles and companies. We tried to get the most out of the studies investigated. Separate companies will have separate solutions which make it hard for researchers to generalize and combine findings when it comes to the way software is developed.

We hope this thesis could support further research into the following areas:

- economic study of Agile Software Product-Line Engineering;
- additional research on requirements and architectural challenges;
- find the context where Agile Software Product-Line Engineering is beneficial towards a regular Software Product-Line Engineering approach; and
- combine thoughts from Lean Software Development to Agile Software Product-Line Engineering.

Bibliography

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2002). *Agile software development methods*. Review and analysis. Espoo: VTT Publications 478. p. 107.

Bass, L., Clements, P. & Kazman, R. (2003). *Software Architecture in Practice, Second Edition*: Addison-Wesley Professional.

Birk, A., Heller, G., John, I., Schmid, K., von der Massen, T. & Muller, K. (2003). Product line engineering, the state of the practice. *Software, IEEE*, 20 (6): 52-60.

Boehm, B. (2008). Making a Difference in the Software Century. *Computer*, 41 (3): 32-38.

Brooks, F. P. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *Computer*, 20 (4): 10-19.

Buhrdorf, R., Churchett, D. & Krueger, C. W. (2004). Salion's Experience with a Reactive Software Product Line Approach. In Linden, F. v. d. (ed.) Lecture Notes in Computer Science, vol. 30014 *Software Product-Family Engineering: 5th InternationalWorkshop, PFE 2003 Siena, Italy, November 4-6, 2003*
Revised Papers, pp. 317-322: Springer Berlin / Heidelberg.

Carbon, R., Knodel, J., Muthig, D. & Meier, G. (2008). *Providing Feedback from Application to Family Engineering - The Product Line Planning Game at the Testo AG*. Proceedings of the 2008 12th International Software Product Line Conference: IEEE Computer Society.

Clements, P. (2002). Being proactive pays off. *Software, IEEE*, 19 (4): 28, 30.

Clements, P. & Northrop, L. (2002a). *Software Product Lines: Practices and Patterns*: Addison-Wesley. 608 p.

Clements, P. C. & Northrop, L. M. (2002b). Salion, Inc.: A Software Product Line Case Study. Pittsburg: Software Engineering Institute, Carnegie Mellon University. 50 p.

Clements, P. C., Jones, L. G., McGregor, J. D. & Northrop, L. M. (2006). Getting there from here: a roadmap for software product line adoption. *Commun. ACM*, 49 (12): 33-36.

Cohen, D., Lindvall, M. & Costa, P. (2004). An introduction to agile methods. In Advances in Computers, vol. 62 *Advances in Computers*, pp. 1-66. San Diego: Elsevier Academic Press Inc.

Conboy, K. (2008). *A Framework of Method Agility in Information Systems Development*. Ph.D. Galway, Ireland: University of Limerick, Department of Computer Science and Information Systems. 306 p.

Dingsøyr, T., Dybå, T. & Abrahamsson, P. (2008). *A Preliminary Roadmap for Empirical Research on Agile Software Development*. Agile, 2008. AGILE '08. Conference, Toronto, ON. 83-94 p.

Dybå, T. & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50 (9-10): 833-859.

Erickson, J., Lyytinen, K. & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. *Journal of Database Management*, 16 (4): 88-100.

Frakes, W. B. & Kang, K. C. (2005). Software reuse research: status and future. *Software Engineering, IEEE Transactions on*, 31 (7): 529-536.

Fraser, S. & Mancl, D. (2008). No Silver Bullet: Software Engineering Reloaded. *Software, IEEE*, 25 (1): 91-94.

Ghanam, Y. (2008). An Iterative Model for Agile Product Line Engineering. *The SPLC Doctoral Symposium, 2008 - in conjunction with the 12th International Software Product Line Conference (SPLC 2008).* Available at: http://ase.cpsc.ucalgary.ca/uploads/APLE/splc2008.pdf (accessed: November 16th, 2008).

Ghanam, Y., Park, S. & Maurer, F. (2008). A Test-Driven Approach to Establishing & Managing Agile Product Lines. *The 5th Software Product Lines Testing Workshop (SPLiT 2008) - in conjunction with the 12th International Software Product Line Conference (SPLC 2008).* Available at: http://ase.cpsc.ucalgary.ca/uploads/APLE/split_workshop.pdf (accessed: November 16th, 2008).

Ghanam, Y. & Maurer, F. (2009). Extreme Product Line Engineering: Managing Variability & Traceability via Executable Specifications. *Agile 2009*. Available at: http://ase.cpsc.ucalgary.ca/uploads/APLE/agile09.pdf (accessed: 3rd of June, 2009).

Gilb, T. (2005). *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*: Elseiver Butterworth-Heinemann.

Gylterud, S. (2008). Constructing a Silver Bullet? Combining Software Product Line Engineering and Agile Software Development. *A thematic literature review*. Available at: http://tinyurl.com/dhr6ly.

Hanssen, G. K. & Fægri, T. E. (2008). Process fusion: An industrial case study on agile software product line engineering. *Journal of Systems and Software*, 81 (6): 843-854.

Hetrick, W. A., Krueger, C. W. & Moore, J. G. (2006). *Incremental return on incremental investment: Engenio's transition to software product line practice*. Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, Portland, Oregon, USA: ACM.

Hevner, A., March, S., Park, J. & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, 28 (1): 75-105.

HomeAway. (2009). *Interview with HomeAway employee on 21st of May 2009*. [Transcription from the interview].

IEEE. (1990). *IEEE standard glossary of software engineering terminology*. IEEE Std 610.12-1990.

Kettunen, P. (2008). Adopting key lessons from agile manufacturing to agile software product development--A comparative study. *Technovation*, In Press, Corrected Proof.

Krueger, C. (2002). Eliminating the adoption barrier. *Software, IEEE*, 19 (4): 29-31.

Krueger, C. W. (2001). Easing the Transition to Software Mass Customization. In Linden, F. v. d. (ed.) Lecture Notes in Computer Science, vol. 2290 *Software Product-Family Engineering: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, pp. 282-293. Bilbao, Spain: Springer Berlin / Heidelberg.

Krueger, C. W., Churchett, D. & Buhrdorf, R. (2008). *HomeAway's Transition to Software Product Line Practice: Engineering and Business Results in 60 Days*. Software Product Line Conference, 2008. SPLC '08. 12th International. 297-306 p.

Larman, C. & Basili, V. R. (2003). Iterative and incremental developments. a brief history. *Computer*, 36 (6): 47-56.

Leffingwell, D. (2007). *Scaling Software Agility: Best Practices for Large Enterprises (The Agile Software Development Series)*: Addison-Wesley Professional.

Linden, F. J. v. d., Schmid, K. & Rommes, E. (2007). *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*: Springer-Verlag New York, Inc.

McGregor, J. (2008). Mix and Match. *Journal of Object Technology*, vol. 7 no. 4 (July-August): pp 7-16. Available at: http://www.jot.fm/issues/issue_2008_07/column1/.

Mehta, M., Anderson, D. & Raffo, D. (2008). Providing value to customers in software development through lean principles. *Software Process: Improvement and Practice*, 13 (1): 101-109.

Noor, M. A., Rabiser, R. & Grünbacher, P. (2008). Agile product line planning: A collaborative approach and a case study. *Journal of Systems and Software*, 81 (6): 868-882.

Northrop, L. (2004). Software Product Line Adoption Roadmap, Technical Report CMU/SEI-2004-TR-22. Pittsburgh, PA, : Software Engineering Institute, Carnegie Mellon University

Northrop, L. M., Clements, P. C., Bachmann, F., Bergey, J., Chastek, G., Cohen, S., Donohoe, P., Jones, L., Krut, R., Little, R., McGregor, J. & O'Brien, L. (2007). *A Framework for Software Product Line Practice, Version 5.0.* Available at: http://www.sei.cmu.edu/productlines/framework.html (accessed: December 15, 2008).

Pohl, K., Böckle, G. & Linden, F. J. v. d. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. New York: Springer-Verlag.

Poppendieck, M. & Poppendieck, T. (2006). *Implementing Lean Software Development: From Concept to Cash (The Addison-Wesley Signature Series)*: Addison-Wesley Professional.

Rajlich, V. (2006). Changing the paradigm of software engineering. *Communications of the ACM*, 49 (8): 67-70.

Rally Software. (2008). HomeAway Creates Success at Home and Abroad with Agile and Rally. Available at: http://www.rallydev.com/downloads/download/84.html (accessed: 20090428).

Schilling, M. (2004). *Strategic Management of Technological Innovation*: McGraw-Hill/Irwin.

Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *Software Engineering, IEEE Transactions on*, 25 (4): 557-572.

Sugumaran, V., Park, S. & Kang, K. C. (2006). Introduction. *Communications of the ACM*, 49 (12): 28-32.

Tian, K. & Cooper, K. (2006). Agile and software product line methods: are they so different? *1st International Workshop on Agile Product Line Engineering (APLE), collocated with the 10th International Software Product Line Conference (SPLC)*. Available at: www.lsi.upc.edu/events/aple/TianCooper.pdf (accessed: December 15, 2008).

Williamson, P. J. & Zeng, M. (2009). Value-For-Money Strategies For Recessionary Times. *Harvard Business Review*, 87 (3): 66-74.

Appendix I: Table of the Cases We Investigated for This Thesis

Name	Description	AM	SPLE	Approach	Advantages	Disadvantages	Extra data possibilities
Confirmit. The case from Hanssen & Fægri, Process Fusion.	A company that combines strategic planning with the EVO (agile) method. Involves a three-part plan for developing software	EVO, company has an agile focus with product line thoughts	Strategic planning of products and variations.	Extractive, had products and then started reusing	Data material access; Handles the transition to agile and Software Product Lines; Contact with company and researcher responsible for data collection.	Already published material on data. SPLE effort still a little unclear. Not much research on EVO.	Yes, in progress. Clearance got, but waiting for a NDA. NDA not obtained. No data available, except case.
HomeAway's Transition to Software Product Line Practice. Krueger, Churchett, Buhrdorf	Homeaway's transition to a Software Product Line in 60 days. Merging several existing product variations into a Software Product Line that uses Krueger's Gears and mass customization	Scrum is mentioned	3-Tiered Software Product Line (Software Product Line) Methodology and Gears Unified Software Product Line Framework.	Extractive approach.	Using Scrum for core asset development	Not mentioned in detail how Scrum is used and how it affect their product line	In progress
IFS - Industrial Financial Systems	Has a component based product base and configure product based on customers' requirements. Claims to be agile	Claim to be agile	Reuse base, configuration to each customer	Proactive ?	Vast amount of data. No published research on material	Quality of data, strategic advantage misrepresentati on (might not get "all" data)	Yes, in progress

Name	**Description**	**AM**	**SPLE**	**Approach**	**Advantages**	**Disadvantages**	**Extra data possibilities**
Salion	Built a Software Product Line on the reactive approach with help of COTS software. Easier to change the product line according to needs of customers. Refactoring parts of software to reduce entropy. Quicker and less expensive approach to SPLE.	Refactoring, customer interaction	Core assets and configuring products to customer.	Reactive with COTS software used	Many different case studies. Follows practice areas in SEI literature.	Focused on the Software Product Line approach and not the agile practices	Maybe. Existing cases detailed. In progress
Providing Feedback Product Line Planning Game at the Testo AG. Carbon, Knodel, Muthig.	After establishing the product line agile practices are introduced to improve the agility in the product line.	Planning game practice	PuLSE. Established and working. Small development organization	incremental and architecture -centric strategy	Product line organization introducing agile practices. The modification of the agile practice explained. Valuable feedback. Other Testo cases also available.	Too specific. Little about process change and organizational effects of that. Also business need not handled	?
Agile prodcut line planning: Muhammad Noor, Rabiser, Grunbacher	Combination of agile and collaboration theory to plan a product line	To some extend	??	??	?	?	?

Name	Description	AM	SPLE	Approach	Advantages	Disadvantages	Extra data possibilities
Software Product Lines in Action. Book by Linden, Rommes and Scmid.	Going through 10 cases of software product line organizations in a BAPO (Business, Architecture, Process and Organization). model with emphasis on transition.	Mentioned in some cases.	Cases investigate difficulties and measures taken to reduce them in a transition to Software Product Lines.	Varies	All cases follow same structure. Full scale case studies.	Little about agility in the cases. Not very detailed about each company. A lot of emphasis in architecture.	
DNV Software	One of the cases from Software Product Lines in Action. Have some employees that are in the agile community in Norway and are in the process of introducing agile.	Not mentioned, will ask and find out	Case describe the change from a platform to a new one and the challenges with that. Organization is also described	Reactive seems like	Norwegian company, have some contacts we can use	Little about agility	Yes
Siemens.– Challenges and Best Practices. Kircher, Schwanninger, Groher	Challenges when introducing a Software Product Line. Request a cookbook, evaluates challenges and best practices.	Mention agility as a challenge regarding the ability to change in a Software Product Line	Not any detailed information about their process, only experiences, guesses and research	?	Thinking about agility.	Process not mentioned in detail. Not how, missing the approach	

Name	**Description**	**AM**	**SPLE**	**Approach**	**Advantages**	**Disadvantages**	**Extra data possibilities**
Engenio's Transition to Software Product Line Practice. Hetrick Krueger Moore	Software Product Line transistion through gears using existing artifacts to build base		Gears software	Incremental	Little BDUF	No agile indications	
Software Product Lines and Configurable Product Bases Paul D. Witman	Building and structure of ERP systems in order-system and supermarkets	Changing environment, responding to change	Reuse with focus on both functional and non-functional requirements	Proactive	Detailed	Too technical. Not much process information.	
A Case Study in Software Product Lines Nascimento, Santana de Almeida, Romero de Lemos Meira	Mobile devices domain. Experiences establishing and evolution of a Software Product Line	Training in the start included	The process partly outlined and measurements explained	Extractive, incremental approach		Little information about process and which practices are used in the development	
Establishing Product Lines in the Automotive Domain.Tischer, Müller, Ketterer,	Case study describing the transition and maintenance of a Software Product Line for Overwatch. A lot of emphasis on technology platform and architecture.	No	Domain and product application split. Difficulties with variation and support for variations in the start	Proactive	SOA used	Not much details	

Name	Description	AM	SPLE	Approach	Advantages	Disadvantages	Extra data possibilities
Software Product Lines Approach in Enterprise System Development. Yuzo Ishida	Describing a product lines with questions like: how Software Product Line methods are applied, how does the org. manage domain and application engineering.	No	Using variations in both components and configurations in run-time	Extractive and proactive		No information about how the process is done and what kind of practices they use and not use	
Introducing Software Product Line Engineering in SME. Sellier, Benguria Urchegui	Software Product Line within mobile games domain.	No	Waterfall phases in a proactive approach	proactive	Detailed with measures	No agile thoughts at all	
Experiences with Product Line Development Software at Overwatch Textron Systems. l Jensen	6 years of experience in the domain. Establishing a working Software Product Line takes time	No	Developing several product lines to satisfy different markets.		Long time span	No details about process.	

Appendix II: Requests for Interviews

Community Short version

Dear all,

I'm looking into agility and software product lines in industry, and would like to perform a short (duration about 20 min), semi-structural interview through Skype or even e-mail about software industry experiences with agility (agile software development) and software product lines. If you are interested please read further.

My name is Snorre Gylterud, master student in computer science at Norwegian University of Science and Technology, currently on exchange at Pohang University of Science and Technology (South Korea), writing my master thesis. I'm contacting you because my thesis is about agile software product lines.

After a literature review on the topic last semester, I've started to investigate industry practices in this research area. I'm now analyzing (using constant comparison method) the articles that could be characterized as an agile software product lines approach.

Therefore I want to perform a set of semi-structural interviews to be able to support my existing assumption and to discover more industry practices which the cases do not cover. I'm looking for companies that have experience in combining agility with software product lines. It would be extremely valuable for me to have the chance to perform a short Skype interview with you, or even just send you my questions on an e-mail for you to reply? The duration of an interview like this would be about 20-30 minutes. The interviews can be anonymized. If you are interested please let me know.

Hope to hear from you!

Community long version

Dear all,

I'm looking into agility and software product lines in industry, and would like to perform a short (duration about 20 min), semi-structural interview through Skype or even e-mail about software industry experiences with agility and software product lines. If you are interested please read further:

My name is Snorre Gylterud, master student in computer science at Norwegian University of Science and Technology, currently on exchange at Pohang University of Science and

Technology (South Korea), writing my master thesis. I'm contacting you because my thesis is about agile software product lines.

After a literature review on the topic last semester, I've started to investigate industry practice on this research area. I'm now analyzing (using constant comparison method) the articles that could be characterized as an agile software product lines approach. Further I will try to find similarities between several cases, but some challenges in the combination would probably be open even after the analysis.

Therefore I wanted to perform a set of semi-structural interviews to be able to support my existing assumption and to discover more industry practices which the cases do not cover. It would be extremely valuable for me to have the chance to perform a short Skype interview with you, or even just send you my questions on an e-mail for you to reply? The duration of an interview like this would be about 20-30 minutes. The interviews can be anonymized. If you are interested please let me know.

The goal of this study is to characterize and discover practitioner practices to be able to generalize a method or a framework for agile software product lines. If I'm able to fulfill this goal it would motivate further research on the topic and create a common communication platform for agile software product lines.

My research questions are:

1. How are agile development principles and practices used in combination with software product line engineering in industry today?

2. In what way could we generalize a method or a framework to describe agility in software product line engineering?

The first will be answered through analysis of case studies and interviews. The second through findings from the first.

Best Regards,

Snorre Gylterud

Community Norwegian version

Hei

I forbindelse med min masteroppgave (NTNU) jobber jeg med industri praksiser innen smidige programvare-produktlinjer. Jeg ønsker å gjennomføre et sett av korte intervjuer (ca 20-30 min) med bedrifter som har eksperimentert eller gjennomfører smidig systemutvikling på større produktlinjer eller sett av produkter innenfor samme domene. Hvis du er interessert les videre.

Jeg er sisteårs masterstudent på NTNU, men er på utveksling i Sør-Korea. Lab-en her er spesialister på programvare-produktlinjer, mens veileder på NTNU forsker på smidig. Etter å ha skrevet et litteratur studie på temaet forrige semester sitter jeg nå og analyserer case studier på temaet. Jeg ønsker å finne ut hva industrien gjør i forhold til dette temaet, da man kan anta at de er foran forskningen på dette området. Derfor ønsker jeg og utføre et sett med korte semi-strukturerte intervjuer for å støtte funnene i analysen og avdekke flere praksiser i industrien.

Jeg ser etter bedrifter som har erfaringer med både smidig utvikling og produktlinjer. Det ville vært utrolig hjelpsomt for meg helst å kunne utføre et kort intervju, men også eventuelt sende en mail med spørsmålene mine til besvarelse. Tidsaspektet på et slikt intervju vil være ca 20-30 min og gjøres anonymt i forhold til ønske. Ta kontakt om dette kan være noe for deg.

Med vennlig hilsen,

Contact Case Companies short

Dear XXX,

I read your XXX case study and would like to perform a short (duration 20 min), semi-structural interview through Skype or even e-mail about agility and software product lines. If you are interested please read further.

My name is Snorre Gylterud, master student in computer science at Norwegian University of Science and Technology, currently on exchange at Pohang University of Science and Technology (South Korea), writing my master thesis. I'm contacting you because my thesis is about agile software product lines.

After a literature review on the topic last semester, I've started to investigate industry practices in this research area. I'm now analyzing (using constant comparison method) the articles that could be characterized as an agile software product lines approach, and your XXX is one of them.

When finishing that I want to perform a set of semi-structural interviews to be able to support my existing assumptions and to discover more industry practices which the cases do not cover. It would be extremely valuable for me to have the chance to perform a short Skype interview with you, or even just send you my questions on an e-mail for you to reply? The timescale of an interview like this would be about 20-30 minutes. The interviews can be anonymized. If you are interested please let me know

Hope to hear from you! Feel free to ask any questions as well.

Contact Case Companies Long

Dear XXX,

I read your XXX case study and would like to perform a short (duration 20 min), semi-structural interview through Skype or even e-mail about agility and software product lines. If you are interested please read further:

My name is Snorre Gylterud, master student in computer science at Norwegian University of Science and Technology, currently on exchange at Pohang University of Science and Technology (South Korea), writing my master thesis. I'm contacting you because my thesis is about agile software product lines.

After a literature review on the topic last semester, I've started to investigate industry practice on this research area. I'm now analyzing (using constant comparison method) the articles that could be characterized as an agile software product lines approach, and your XXX is one of them. Further I will try to find similarities between several cases, but some challenges in the combination would probably be open even after the analysis.

Therefore I wanted to perform a set of semi-structural interviews to be able to support my existing assumption and to discover more industry practices which the cases do not cover. It would be extremely valuable for me to have the chance to perform a short Skype interview with you, or even just send you my questions on an e-mail for you to reply? The timescale of an interview like this would be about 20-30 minutes. The interviews can be anonymized. If you are interested please let me know.

The goal of this study is to characterize and discover practitioner practices to be able to generalize a method or a framework for agile software product lines. If I'm able to fulfill this goal it would motivate further research on the topic and create a common communication platform for agile software product lines.

My research questions are:

1. How are agile development principles and practices used in combination with software product line engineering in industry today?

2. In what way could we generalize a method or a framework to describe agility in software product line engineering?

The first will be answered through analysis of case studies and interviews. The second through findings from the first.

Hope to hear from you!

Best Regards

Appendix III: Interview Guides

Organization and personal experience: Size, Domain, Customers; Roles obtained, # of Years

HomeAway / Salion	*DNV Software*	*IFS*
Present yourself, software engineering background and current role?	Present yourself, software engineering background and current role?	Present yourself, software engineering background and current role?
Number of years in the industry?	Number of years in the industry?	Number of years in the industry?
Working with SPLE?	Working with SPLE?	Working with SPLE?
When started looking at agile?	When started looking at agile?	When started looking at agile?
Read the case study from SPLC 2008, regarding the organizational information such as size, domain etc what have changed?	I read Software Product Lines in Action by Linden, Schmid and Rommes (2007), is this information still correct? Regarding size, domain and customers?	Obtained information from you and wrote a short summary, did it reflect your software development efforts? Or differences?

Software Product Development: Commonalities and Variabilities; Business and Marketing

HomeAway / Salion	*DNV Software*	*IFS*
How is your company exploiting commonalities and variability in the products delivered? And for how long has the company done it?	How is your company exploiting commonalities and variability in the products delivered? And for how long has the company done it?	How is your company exploiting commonalities and variability in the products delivered? And for how long has the company done it?
What are the reasons for developing software products in this way?	What are the reasons for developing software products in this way?	What are the reasons for developing software products in this way?
What are the benefits and challenges of a software development approach like this?	What are the benefits and challenges of a software development approach like this?	What are the benefits and challenges of a software development approach like this?
What efforts are made to define the scope and market for your products? Does it belong to a certain domain?	What efforts are made to define the scope and market for your products? Does it belong to a certain domain?	What efforts are made to define the scope and market for your products? Does it belong to a certain domain?

How is the product delivered to the customers? Own department for that? How is customers included in the development process?	How is the product delivered to the customers? Own department for that? How is customers included in the development process?	How is the product delivered to the customers? Own department for that? How is customers included in the development process?
How is support and feedback gathered from customers?	How is support and feedback gathered from customers?	How is support and feedback gathered from customers?

Software Development Process: Requirements, Architecture, Testing, Documenting, Management

HomeAway / Salion	*DNV Software*	*IFS*
How is the variabilities documented and handled throughout development?	How is the variabilities documented and handled throughout development?	How is the variabilities documented and handled throughout development?
How do the business people and technological people cooperate?	How do the business people and technological people cooperate?	How do the business people and technological people cooperate?
How is the teams organized and managed? Which grade of autonomy do the teams have?	How is the teams organized and managed? Which grade of autonomy do the teams have?	How is the teams organized and managed? Which grade of autonomy do the teams have?
How does your process differ from other software development methods like waterfall, iterative processes, etc?	How does your process differ from other software development methods like waterfall, iterative processes, etc?	How does your process differ from other software development methods like waterfall, iterative processes, etc?
How is the product tested and quality assured during your development process?	How is the product tested and quality assured during your development process?	How is the product tested and quality assured during your development process?
Which practices and principles do you follow during a product development process?	Which practices and principles do you follow during a product development process?	Which practices and principles do you follow during a product development process?
Which measures do you use for your process? And how is that tracked throughout the stakeholders?	Which measures do you use for your process? And how is that tracked throughout the stakeholders?	Which measures do you use for your process? And how is that tracked throughout the stakeholders?
In your view - what makes your approach agile?	In your view - what makes your approach agile?	In your view - what makes your approach agile?
Which challenges have you experienced with agility?	Which challenges have you experienced with agility?	Which challenges have you experienced with agility?

Continued ***Software Development Process***

HomeAway / Salion	*DNV Software*	*IFS*
In your view - how is the fit on combining strategic product development and agile practices?	In your view - how is the fit on combining strategic product development and agile practices?	In your view - how is the fit on combining strategic product development and agile practices?
How would you describe your documentation throughout the development?	How would you describe your documentation throughout the development?	How would you describe your documentation throughout the development?
How was the adoption of agility into your product development handled? Which obstacles and quick-wins did you experience?	How was the adoption of agility into your product development handled? Which obstacles and quick-wins did you experience?	How was the adoption of agility into your product development handled? Which obstacles and quick-wins did you experience?
In what extend to you change your processes to improve? And what actions are usually done within both teams and management?	In what extend to you change your processes to improve? And what actions are usually done within both teams and management?	In what extend to you change your processes to improve? And what actions are usually done within both teams and management?

Appendix IV: Case Description Template

Context:

- what kind of studies used?
- what kind of company?
- size of development team/org?
- domain and why software product lines and agility

Software Product Line Engineering:

- Approach towards Software Product Lines; Type? Effort?
- SPLE practices; Which practices are used? What changes are made to those?

Software Product Line Development:

- Core Asset Development; How is the reuse platform used, changes made to this one agile? Requirement engineering on this?
- Product Development; How are products realized? What makes this agile?
- Management; What kind of principles and methods are used? Metrics for agility?
- Marketing and Sales; How is customer involved? How do they sell the product and configure it?

Agile:

- Action towards change; In what way is changes handled? Proactively or reactively? Learning?
- Maximize value; What characterizes the efficiency of the approach or method?
- Agile practices; Which agile practices are used? Adoptions or changes made to them?

Benefits/ Disadvantages

SPLE level / Agility level

Appendix V: Photo from Analysis Process

V2