Geir Kjetil Hanssen

From Agile Software Product Line Engineering Towards Software Ecosystems

Thesis for the degree of Philosophiae Doctor

Trondheim, November 2010

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

© Geir Kjetil Hanssen

ISBN 978-82-471-2427-7 (printed ver.)
ISBN 978-82-471-2426-0 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2010:219

Printed by NTNU-trykk

Alla dessa dagar som kom och gick,
inte visste jag att det var livet

- Stig Johansson

Abstract

Development and evolution of software products is a challenging endeavor and a significant sub-field of software engineering. One of the commonly applied approaches to control and manage this process is software product line engineering (SPLE). There exist a few process frameworks where the development of lines of related software products is basically a sum of two processes: the development of reusable assets and the rapid construction of software applications using pre-developed assets. Agile software development (ASD) is another major paradigm, which also has been widely adopted by the industry over the past decade.

SPLE and ASD seek to achieve the same goal i.e. rapid and efficient construction of software. However, the former emphasizes extensive up-front investment in the development of assets for later re-use in contrast to ASD, which emphasizes a reactive approach, avoiding up-front planning and development

Even though these two approaches may seem to oppose each other, the industry has lately showed great interest in combining them both, aiming to cover the weaknesses of the one with the strengths of the other. In combination with the overall shift in the software industry from closed systems mindset towards open systems, the uptake of some ASD principles (for example active customer engagement, incremental and iterative development, and open information flows) in product line organizations may contribute to the emergence of more flexible software ecosystems.

This thesis presents a longitudinal study of a software product line organization, which has adopted an adapted ASD methodology in an SPLE context and to a large extent has successful in gaining benefits from both approaches, developing towards more open yet controlled processes. Data have been collected over a period of approximately five years following the progression from a strictly waterfall oriented approach, via the adoption of the agile method Evo, up to the current combined agile software product line engineering approach.

The following research questions have been addressed in this thesis:

RQ1: How can software product line engineering and agile software development be combined?

RQ2: How does a software ecosystem shape?

The main contributions of this work are:

C 1. Through a longitudinal study of a software product line organization we provide detailed insight into an industrial case and how they have changed over time.

C 2. We have illustrated some of the details of how SPLE and ASD can be combined in practical terms. We describe the current organization, their product line and their agile software product line engineering process.

C 3. We have illustrated how the incorporation of some of the central agile principles has enabled a closer cooperation with external actors.

C 4. We have explained the emergence and mode of operation of a software ecosystem, and provided a conceptual model of software ecosystems based on our findings.

C 5. We have proposed a theory of software ecosystems, rooted in socio-technical theory and the concept of organizational ecology.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfillment of the requirements for the degree of Philosophiae Doctor.

The work leading to this thesis has been performed at the Department of Computer and Information Science, NTNU, Trondheim, with Professor Reidar Conradi (NTNU) as the main supervisor, and Chief Scientist Tore Dybå (SINTEF ICT) and Professor Eric Monteiro (NTNU) as co-supervisors.

This work was conducted as part of the EVISOFT research project, supported by the Research Council of Norway under grant 174390/I40.

Acknowledgements

There are many persons that have encouraged, helped, and supported my work on this thesis in various ways over the last four years. First, I would like to thank my supervisors Reidar Conradi (NTNU), Tore Dybå (SINTEF), and Eric Monteiro (NTNU) for their timely advice and kind pressure whenever needed. During my studies I have continued to work at SINTEF ICT but with a part-time position. It has been of great importance to me to have this contact with my colleagues throughout the study, both for professional and social reasons. Thanks to Tore Dybå, Tor Erlend Fægri, Torgeir Dingsøyr, Nils Brede Moe, Børge Haugset, and all the other great people at SINTEF ICT. I would also like to thank my manager, Eldfrid Øfsti Øvstedal for giving me this opportunity; I hope your investment will pay off. I am also grateful to Pekka Abrahamsson that has provided valuable advice through burning discussions during his time as my colleague at SINTEF. I am also grateful to the Research Council of Norway that have supported my work through grant number 174390/I40, funding the EVISOFT research project which was managed by Tor Ulsund. Thank you Tor! I also need to thank the good people at my case company CSoft for their support over many years. My non-disclosure agreement with them prohibits me from mentioning names, but you know who you are – thank you.

During the time of this study I have been a member of a group of PhD students and the like, that – despite the slightly humorous name *"The Scientist Factory"* – provided a highly valuable arena for discussing good and bad ideas, learning from failures and successes of others, getting nice and nasty feedback, and the sharing of frustrations. Thanks to Eric Monteiro and Torgeir Dingsøyr for managing our meetings, and thanks to my peers; Børge Haugset, Gasparas Jarulaitis, Gro Alice Hamre, Kirsti Elisabeth Berntsen, Ole Andreas Alsos, Øyvind Hauge, Thomas Østerlie, Tor Erlend Fægri, Torstein Hjelle, and Vigdis Heimly. I have also benefited immensely from numerous conversations with a lot of other great people at the university, with industrial partners, at SINTEF, and other places. Some of them are: Anita Gupta, Sven Ziemer, Knut Harald Hassel Nielsen, Tor Stålhane, Maria Letizia Jaccheri, Anders Kofod-Petersen, Sverre Tinnen, Haakon Fougner Spilde, Hans Westerheim, Øyvind Pettersen, Frank Bertheussen, Darja Šmite, mom, and dad. I also need to thank my co-authors at the University of Oslo and the SIMULA research laboratory, Aiko Fallas Yamashita and Leon Moonen. Finally I want to thank my wife Randi for her unfaltering support, and my children Mikael, Silje, and Marius for accepting that at evenings, weekends, and holidays their dad put priority on work instead of spending time with them. Well guys – *now* I'm done!

NTNU, September 10, 2010
Geir Kjetil Hanssen

Contents

List of Figures

List of Tables

Abbreviations

ASD	Agile Software Development
ASP	Application Service Provider
ASPLE	Agile Software Product Line Engineering
CEO	Chief Executive Officer
CI	Continuous Integration
CSO	Chief Strategy Officer
CTO	Chief Technical Officer
EPG	Electronic Process Guide
ICT	Information and Communication Technology
IET	Impact Estimation Table
IPR	Intellectual Property Rights
KLOC	K (1000) Lines of Code
NTNU	Norwegian University of Science and Technology
OSS	Open Source Software
PDCA	Plan Do Check Act
PMA	Post-mortem analysis
PMT	Product Management Team
PSG	Product Strategy Group (succeeding the PMT from approx. 2006)
QA	Quality Assurance
R&D	Research & development (also the name for the development department)
SE	Software Engineering
SECI	Socialization Externalization Combination Internalization
SEI	Software Engineering Institute
SOAP	Simple Object Access Protocol
SPLE	Software Product Line Engineering
TAM	Technical Account Manager

PART I - Summary of studies

1 Introduction

1.1 Problem Outline

Development of software products as well as the maintenance and evolution of these products over time is a complex and challenging task for most software product organizations (Adler, 2005). To support this effort, a large section of the product oriented software industry applies variants of software product line engineering (SPLE) (Clements and Northrop, 2002; Pohl, Böckle et al., 2005). SPLE is an extensive approach to organizing the continuous development of software products, where the main property is the planned, prepared, and anticipated reuse of a set of domain artifacts for later fast and efficient composition of applications. This is in principle a better approach than developing each product from scratch each time (Böckle, Clements et al., 2004), a principle which have been supported by industrial case studies, for example (Buhrdorf, Churchett et al., 2004; Kiesgen and Verlage, 2005), and more recently (Babar, Ihme et al., 2009). Several rather comprehensive SPLE methods or frameworks exist, and have been adopted and practiced in the industry for more than a decade. These frameworks are based on the development of assets for later (anticipated) use, an approach, which may seem to conflict with another major recent methodological development in the software industry: agile software development (ASD). Recently, there has been a lively interest in combining these two approaches as they both promise to improve the efficiency of the software product development process and the responsiveness towards an increasingly volatile market (Tian and Cooper, 2006).

The first challenge this creates is how to practically combine the two methods: SPLE imposing up-front investments and a pro-active approach, and ASD emphasizing flexibility and a fundamentally re-active approach. The second challenge, more relevant to this thesis, is to understand the rationale of such a combination and how it affects the product line organizations ability to produce quality software solutions.

This combination of processes is related to a change in the industry and its market, from traditional closed processes towards open development processes crossing organizational borders (Moore, 1993), a trend that may lead towards a new entity – software ecosystems (Bosch, 2009; Messerschmitt and Szyperski, 2003). This change seems inevitable, and it is important to understand its development and how to respond to it.

1.2 Research Context

This thesis summarizes about five years of studies of a software product line organization called CSoft (a pseudonym[1]), and shows how they have progressed from a plan-based and closed process approach towards a more open process, where feedback and collaboration with various external actors are fundamental drivers. This change in process has affected how the organization relates to its external environment – which we describe as a software ecosystem. The adoption of Evo (Gilb, 2005), an agile method, in 2004 has been an important enabler for this change. During the period from 2004 to 2010, four studies have been conducted, starting with an investigation of the early adoption of Evo and its preliminary effects, and then the later studies looking into how the organization collaborates with its external environment in both planning and development. Thus, the object of study in this thesis is twofold. Firstly, we describe and investigate the present organization, its processes, and its product line. This is the result of fourteen years of development since the establishment of CSoft, and is an example of an agile software product line engineering process. Secondly, we study this change that has taken place and how the situation has progressed towards a software ecosystem.

1.3 Research Questions

What are we studying?

- We have conducted four successive studies of a medium-sized software product line organization. The first study started in 2004 and the last one was completed in 2010, with completion of data collection in 2009 – thus covering almost five years of development. The focus of study has been 1) the longitudinal development of this organization, and 2) the resulting (present) organizational set-up, its planning and development processes, its product line, and its relations with its organizational environment.

Why are we interested in it?

- The investigated organization has adopted an agile development method in a product line environment, and matured this combination over a period of five years in order to improve software process performance in general and the relationship with actors external to the organization in particular.
- The organizational development being described in this thesis – a movement towards an open product line engineering process – is strongly related to an

[1] Due to a non-disclosure agreement with CSoft; company, product, and domain-specific details are omitted to maintain the anonymity of the case organization.

increasing focus on the concept of software ecosystems. This concept is still poorly understood within software engineering, and no explicit coupling to relevant and well-established theories on organizational ecology has been developed.

Why should this be interesting to others?

- The company's position as the market leader is a strong indication of a well functioning organization (including its processes and products). Generalized descriptions of their agile software product line engineering approach, founded on data and related to theory, have a great potential of revealing valuable implications for practice. Likewise, we also propose implications for theory as a contribution to the research community and continued research on software ecosystems.
- Based on the four studies and the resulting six publications, this thesis seeks to answer the following two research questions:

RQ1:	How can software product line engineering and agile software development be combined?
RQ2:	How does a software ecosystem shape?

1.4 Research Design

The complete study of how CSoft has developed over time and their resulting approach to evolving their product line builds on four recent studies, reported in six publications. Studies 1-3 are used to answer research question 1, and study 4 is used to answer research question 2. In sum, these studies constitute a longitudinal study and each of the underlying studies focuses on topics related to the research questions of this thesis. This longitudinal study has been exploratory, and the overall objective has been to collect data to describe the ongoing development of CSoft as well as the resulting process and organization. This means that the study did not start out with a hypothesis or theory to be tested but rather that it is a theory building type of study. For example implications for theory being discussed in section 5.4 have *emerged* throughout the study.

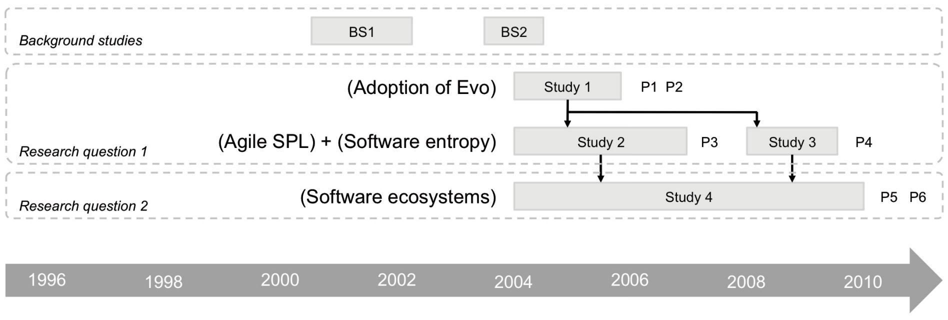


Figure 1 - Study design

The four studies covered the following topics:

Table 1 - Focus of the studies

Study	Focus	Paper
Study 1	The initial adoption of the agile method Evo in a waterfall-oriented product line organization, with an emphasis on how developers relate to customers and the fragility of the agile approach.	P1, P2
Study 2	The combination of ASD and SPLE and the coordination between the strategic, tactical, and operational processes.	P3
Study 3	The emerging problem of software entropy, the potential reinforcing effect from the agile development process, and potential solutions to improve the situation.	P4
Study 4	The increasing openness of information sharing and cross-organizational collaboration.	P5, P6

In addition to the six publications we have also made use of two earlier background studies (by other authors) for details related to the early development of the case organization (Johansen, 2005; Moe, Dingsøyr et al., 2002).

All these studies, conducted sequentially in time, build on the results from previous studies. The initial focus was the adoption of agile software development in a software product line engineering context (discussed in section 5.1). This is followed by a discussion of the emergence of a software ecosystem (section 5.2).

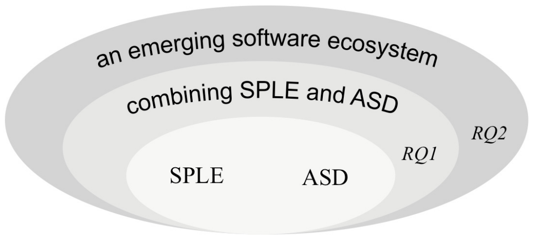


Figure 2 - Evolution of research

1.5 Publications

P1: **G.K. Hanssen**, T.E. Fægri, *Agile Customer Engagement: a Longitudinal Qualitative Case Study*, in: Proceedings of the 5th International Symposium on Empirical Software Engineering (ISESE'06), IEEE Computer Society, Rio de Janeiro, Brazil, 2006, pp. 164-173.
Relevance to this thesis: This paper presents the results of a study of the change from a waterfall-like approach to the agile Evo development process, and how it has affected the way the development organization relates to the customers in their release projects. This study was conducted shortly after the adoption of Evo, and presents the initial effects.
My contribution: As the principal author I initiated and conducted the study with the case organization. The analysis of data was done collaboratively.

P2: T.E. Fægri, **G.K. Hanssen**, *Collaboration and process fragility in evolutionarily product development*, IEEE Software 24 (3) (2007) 96-104.
Relevance to this thesis: This paper reports on a study of the transition to Evo, and explains both positive and negative effects that were experienced shortly after the process adoption.
My contribution: I participated in the whole process, from planning of the study to data collection, analysis, and reporting. The first author had the overall responsibility, and made the final decisions on form and language.

P3: **G.K. Hanssen**, T.E. Fægri, *Process Fusion - Agile Product Line Engineering: an Industrial Case Study*, Journal of Systems and Software 81(2008) 843-854.
Relevance to this thesis: This paper reports on the study of the same case, but builds on new data collected. The paper has a holistic focus on the development and evolution of the product line, and explains how the long-term strategic planning process relates to the tactical processes of developing software and the day-to-day operational processes. The data in this study were collected some years after the initial adoption of Evo (reported in P1 and P2), and thus represent a more mature organization.
My contribution: As the principal author I had the overall responsibility, and made the final decisions on form and language. The whole process, from planning of the study to data collection, analysis, and reporting was done collaboratively.

P4: **G.K. Hanssen**, A.F. Yamashita, R. Conradi, L. Moonen, *Software entropy in agile product evolution*, in: Proceedings of the 43rd Hawaiian International Conference on System Sciences (HICSS'10), IEEE Computer Society, Hawaii, USA, 2010.
Relevance to this thesis: This paper goes deeply into one of the most severe problems in the case organization and one of the main impediments in the agile software development process, namely software entropy. Based on a thorough overview of relevant empirical studies on code-smell[2] detection and the resulting code refactoring as well as new data collected, this paper gives an insight into the causes behind the problem and the negative effects it has on the agile workflow. Potential solutions are discussed.
My contribution: As the principal author I was in charge of the study design, data collection, and analysis. The second author was in charge of the inherent literature review, but with my contribution for some of the summary of results. The second author also contributed significantly to the discussion of the results, both from the case study and the literature review.

P5: **G. K. Hanssen**, *Opening up Software Product Line Engineering*, In proceedings of the 1st International Workshop on Product Line Approaches in Software Engineering, in conjunction with the 32'nd International Conference on Software Engineering (ICSE). 2010. Cape Town: ACM.
Relevance to this thesis: This workshop paper builds on the same data as reported in P6 but elaborates the topic of open processes and cross-organizational

[2] "Code-smells" is a term close to "anti patterns" and refers to symptomatic flaws in code structures.

collaboration. Doing so, relevant literature is referred to and discussed in relation to the study of CSoft.
My contribution: All work was done by the sole author.

P6: **G. K. Hanssen**, An Emerging Software Ecosystem: A Longitudinal Case Study, submitted to the Journal of Systems and Software, special call on "Software Ecosystems", guest editor is Jan Bosch. See: http://eventseer.net/e/12074/.
Relevance to this thesis: This paper is the main contribution in the series of studies of CSoft as it summarizes the organizational development over time. It also identifies a relevant theoretical foundation (organizational ecology), to which the results are related.
My contribution: All work was done by the sole author.

(Note: the pseudonym of the case organization were *CompNN* in papers 1-3 and *CSoft* in papers 4-6 as well as in this thesis. These names refer to the same organization. The shift of name is only due to aesthetical reasons.)

1.6 Contributions

This thesis contributes to ongoing research on **(1)** how agile methods can be applied in software product line engineering (Babar, Ihme et al., 2009; Carbon, Lindvall et al., 2006; Ghanam and Maurer, 2009; Ghanam and Maurer, 2010; Hanssen and Fægri, 2008; Mohan, Ramesh et al., 2010; Noor, Rabiser et al., 2008), and **(2)** how software ecosystems shape and function (Bosch, 2009; Bosch and Bosch-Sijtsema, 2009; Jansen, Brinkkemper et al., 2009; Jansen, Finkelstein et al., 2009; Messerschmitt and Szyperski, 2003).

Through a longitudinal study of a software product line organization and its change, we have studied the practicalities and outcome of such a process combination. Through this study, we have discovered how some of the fundamental agile principles have improved collaboration with the external environment of the product line organization, consisting of customers and external organizations relating to the product line. Based on these insights we propose a theory on software ecosystems, illustrated by our studies.

In more detail, this thesis has five main contributions. C1-C3 gives insight into the combination of ASD and SPLE (based on the discussions in section 5.1). C4-C5 relates to the development of the concept– and a theory on software ecosystems (based on the discussions in section 5).

C 1. Through a longitudinal study of a software product line organization we provide detailed insight into an industrial case and how they have changed over time.

C 2. We have illustrated some of the details of how SPLE and ASD can be combined in practical terms. We describe the current case organization, their product line and their agile software product line engineering process.

C 3. We have illustrated how the incorporation of some of the central agile principles has enabled a closer cooperation with external actors.

C 4. We have explained the emergence and mode of operation of a software ecosystem, and provided a conceptual model of software ecosystems based on our findings.

C 5. We have proposed a theory of software ecosystems, rooted in socio-technical theory and the concept of organizational ecology.

The following list summarizes contributions reported in the selected papers supporting this thesis:

P1: Agile Customer Engagement: a Longitudinal Qualitative Case Study (ISESE Conference, 2006)

- Active engagement of external stakeholders in the development process requires continuous proactive management. An external stakeholders' motivation to contribute relies on immediate benefits, in return active stakeholders positively affect developers' motivation and confidence in the result.
- The adoption of agile principles within a software product line context requires a high degree of discipline to coordinate activities. Also, a technical infrastructure automating repetitive tasks improves the effectiveness of agile development practices.
- Applying agile principles in a product development context improves process visibility both internally in a development organization as well as externally among stakeholders.

P2: Collaboration and process fragility in evolutionary product development (IEEE Software, 2006)

- Maintaining a watchful, vigorous stakeholder management capability is paramount to successful application of core agile principles in a product development context. Rapid feedback on developed code, appropriate metrics, and efficient decision-making reduce uncertainty and thus improve motivation.
- To reduce the risk of architectural erosion, involvement of external stakeholders should be balanced with internals, with the focus on the product as an engineering artifact. (Relates to P4).

- An agile process with a high iteration frequency increases vulnerability to irregularities. Unresponsive stakeholders expose this fragility. Maintaining a high iteration frequency is expensive and requires a high degree of discipline at many organizational levels.

P3: Process fusion: An industrial case study on agile software product line engineering (Journal of Systems and Software, 2008)

- SPLE naturally supports a long-term strategic process, and ASD naturally supports a short-term tactical process. The day-to-day operational process connects these two. This three-process focus enables exploitation of long-term ambitions for innovation as well as small-scale tactical innovations, respectively radical and incremental innovations.
- Running an explicit strategic process and working closely with external stakeholders poses additional overhead but promotes valuable creativity.

P4: Software entropy in agile product evolution (HICSS Conference, January 2010)

- Complexity of the software product hampers productivity and quality of the agile development process. The focus on high speed in short iterations may not leave enough time to resolve problems related to system entropy.
- In order to manage system entropy, agile development processes need additional support for understanding, planning, and evaluating the impact of changes. Code smell analysis and refactoring is a viable solution.

P5: Opening Up Software Product Line Engineering (PLEASE Workshop, May 2010)

- The orientation of the software industry towards software ecosystems can be supported by an opening of 1) information flow, 2) innovation processes, and 3) technical interfaces.

P6: A Longitudinal Case Study of an Emerging Software Ecosystem: Implications for Practice and Theory (Submitted to the Journal of Systems and Software, April 2010)

- The software (product) industry is going through a change where the traditional closed system mindset is being replaced by more open collaboration between the software provider and external actors.
- Agile software development principles, in particular iterative/incremental development and active stakeholder participation, may enable a product line organization to manage and to benefit from the change in the industry.

- We propose a conceptual model of software ecosystems.
- The theory on organizational ecology is used to 1) explain results of the case study, and 2) establish a common platform for future research.

1.7 Thesis Structure

This remainder of the thesis consists of two parts.

PART I – Summary of studies

Chapter	Content
2 – Background	Presents the background for the analysis, discussion, and the conclusions of this thesis. The chapter covers three areas. First it gives insight into the present case organization and its development over time, in particular for the last five years. Second, it presents an overview of the state of the art, and the state of research on software product line engineering, agile software development, the combination of these two approaches, and the emerging concept of software ecosystems. Third, the background chapter describes a theoretical framework based on socio-technical theory and organizational ecology.
3 – Research Approach	States the research goals and the related research questions. It also motivates for and explains the research approach. The chapter gives details on how data have been collected and analyzed.
4 – Results	Presents the synopsis of the results of the four studies.
5 – Discussion and Implications	Discusses the main concepts being investigated and developed, which is agile software product line engineering, the opening of the development process, and how this leads towards an ecosystem. Further, the chapter explains the implications for theory and practice.
6 – Conclusions	Uses the discussion of results to answer the research questions and to clarify the practical and theoretical contributions of the thesis. We also provide directions for future research.

PART II – selected publications

1. Hanssen, G.K. and Fægri, T.E., Agile Customer Engagement: a Longitudinal Qualitative Case Study. In proceedings of 5th International Symposium on Empirical Software Engineering (ISESE'06), 2006, Rio de Janeiro, Brazil. ACM: p.164-173.
2. Fægri, T.E. and Hanssen, G.K., Collaboration and Process Fragility in Evolutionarily Product Development. IEEE Software, 2007. 24(3): p. 96-104.
3. Hanssen, G.K. and Fægri, T.E., Process Fusion - Agile Product Line Engineering: an Industrial Case Study. Journal of Systems and Software, 2008. 81: p. 843-854.
4. Hanssen, G.K., Yamashita, A.F, Conradi, R., and Moonen, L., Software Entropy in Agile Product Evolution. In proceedings of 43rd Hawaiian International Conference on System Sciences (HICSS'10). 2010. Hawaii, USA: IEEE Computer Society. p. 1-10.
5. Hanssen, G.K. Opening up Software Product Line Engineering. In proceedings of 1st International Workshop on Product Line Approaches in Software Engineering, in conjunction with the 32nd International Conference on Software Engineering (ICSE). 2010. Cape Town, South Africa. p. 1-7.
6. Hanssen, G.K., A Longitudinal Case Study of an Emerging Software Ecosystem: Implications for Practice and Theory, submitted to Journal of Systems and Software, under review.

Statement of authorship of joint publications from Tor Erlend Fægri
Statement of authorship of joint publications from Aiko Fallas Yamashita
Statement of authorship of joint publications from Reidar Conradi
Statement of authorship of joint publications from Leon Moonen
Listing of all publications

2 *Background*

This chapter is organized in three main sections. First, we give an overview of a set of relevant topics, which forms a basis for the later discussion of our findings. Second, we present and explain the details of the case organization we have studied: the organization, their product line, their processes, and a retrospective of how this organization has changed over the past years. Third, we present a theoretical framework explaining the concepts of socio-technical theory and organizational ecology.

2.1 State of the art and research

The topic of this thesis relates to several areas of Software Engineering, with a particular emphasis on three of them: 1) Software Product Line Engineering, 2) Agile Software Development, and 3) Software Ecosystems. The former two are relatively well-established concepts as they both have been practiced in industry and researched by academia for 10-15 years – a long time span in software engineering. This chapter explains concepts and ideas, and gives an overview of relevant research. Lately, the idea of combining SPLE and ASD, seemingly conflicting methodologies, has emerged to take advantage of their respective strengths. This chapter covers this recent development. The latter concept, Software Ecosystems, is currently emerging, enriched by ideas from several related fields, thus it is as yet "unpolished" as a sub-discipline. However, importantly to the discussion and conclusions of this thesis, the substance of this concept is presented together with some viewpoints on the probable development in the near future.

2.1.1 Software product line engineering

SPLE is an extensive and organization-wide approach to developing and evolving lines of software products. In this context, a product line is a set of related software products that have some common parts, for example software components, architectural design, data structures, but that still show distinct and different characteristics. The main concept of SPLE is to prepare for and support this variability in the characteristics of the products in the product line, in order to prepare for reuse, to reduce time-to-market, to reduce development costs, to reduce maintenance effort, to improve the quality of software products and to cope with complexity (Pohl, Böckle et al., 2005) p.9. For example, industry reports a reduction of time-to-market less than 50% (Philips Medical Systems) (*ibid.*, p.233) and a reduction of resource consumption by 20-30% (Bosch Gasoline Systems) (*ibid.*, p.133).

A similar term, product families (Linden, 2002), is also frequently used, but for simplicity the *product line* term will be used throughout this text.

Various comprehensive and detailed descriptions of SPLE practices and principles are available, for example, in the frequently quoted books by Clements and Northrop (2002), Pohl et al. (2005) and van der Linden et al. (2007). These descriptions vary somewhat in their use of terminology. At a conceptual level, they all describe SPLE as two main complementary processes. As Figure 3 (derived from (Clements and Northrop, 2002), p.30) illustrates, the first process deals with the development and maintenance of a reusable core or domain assets, typically code organized as components or modules. The second process deals with the development of software products, or applications, using these core assets for rapid and (cost) efficient composition of software products. Variants of this fundamental process are the pro-active approach where core assets are developed up-front due to an anticipated need in the near future and the re-active approach where re-usable assets are derived or harvested from products and stored for later re-use in other applications (McGregor, 2008).

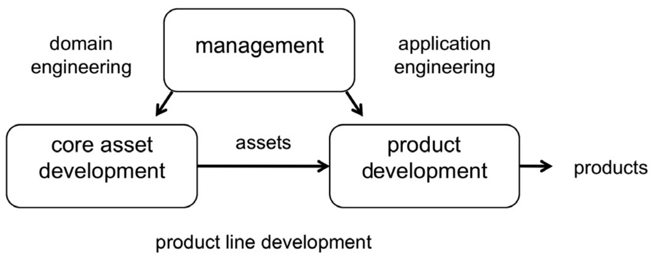


Figure 3 - High-level SPLE processes

Beyond the central dual process of domain asset development and product development, various other supportive processes and techniques are described, for example management processes and software architecture practices. Pohl et al. (2005) (p. 22) has developed a software product line engineering framework that defines and relates these practices (Figure 4). Another extensive SPLE process description is the guidelines provided by the Software Engineering Institute, which describes 29 practice areas in detail (Clements and Northrop, 2002)

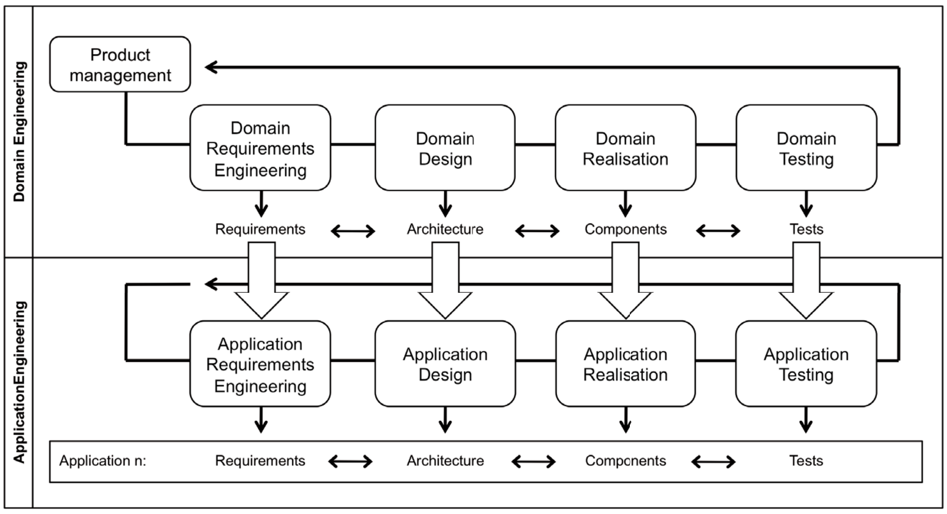


Figure 4 - A software product line engineering framework

This approach to product development and evolution of products over time is very much inspired by practices in other industries. For example, the idea is comparable to how consumer electronics products are made today. A supplier of home appliances offers lines of e.g. washing machines having variable characteristics such as number of programs, motor size etc., but that still share many common parts or components, for example the door locking mechanism and the water pump. This gives the customer the opportunity to find a product matching the intended use at a suitable price level. These two concepts of reuse and variability are equally essential when it comes to software product lines.

The fundamental principle of SPLE is that extensive initial investment should be made in the development of a platform of reusable domain-generic assets for anticipated later use in products. Following this principle presents an opportunity for the efficient construction of new products, but carries with it a risk of not getting an economically viable return from the investment, if pre-developed assets are not sufficiently reused for any reason (McGregor, 2008). This real risk is caused by the time span between development (investment) and use (pay-off).

SPLE is a holistic approach to software engineering, affecting aspects from process to technology. van der Linden et al. (2007) has defined the BAPO-model which identifies four interrelated aspects; **B**usiness, **A**rchitecture, **P**rocess and **O**rganization. In order to describe and practice SPLE, these four concerns must be addressed. The description of the case organization in section 2.1 addresses these concerns as well as providing an overview of their historical development.

2.1.2 Agile software development

Agile software development (ASD) is a way of organizing the software development process, emphasizing direct and frequent communication – preferably face-to-face, frequent deliveries of working software increments, short iterations, active customer engagement throughout the whole development life-cycle, and responsiveness to change rather than change prevention. This can be seen to be in contrast with plan-based processes, which emphasize thorough and detailed planning and design upfront followed by consequent plan conformance. Over the past ten years or so, agile methods have gained lively interest and great popularity as they promise to avoid schedule overruns, development of wrong features, excessive overhead, heavy documentation, excessive formalism, costly re-planning, and extensive management. A relatively large number of (more or less) empirical studies on the use and effects of agile methods were reported in the past years, yet the sum of these studies provide little qualified knowledge about agile software development (Dybå and Dingsøyr, 2008). The most thoroughly investigated agile techniques or practices so far are pair programming (Arisholm, Gallis et al., 2007), test-driven development (Erdogmus and Morisio, 2005; Müller and Hagner, 2002) and the customer on-site practice (part of XP) (Martin, 2009). Other, more complex techniques and aspects of ASD are still not sufficiently investigated and understood.

There exists a wealth of introductions to the basic concepts of agile software development, one of the more fundamental and most referred to is the so called agile manifesto (http://www.agilemanifesto.org) which defines four fundamental values and twelve principles. Several methods based on these ideas are in use, all sharing the common set of values and principles. The best known and the most used agile methods are Extreme Programming (XP) (Beck, 1999) and Scrum (Schwaber, 2001). The ideas behind agile software development are not new (Merisalo-Rantanen H., 2005), as they were clearly inspired by agile and lean manufacturing which has been in use in many types of industries for decades. The radical innovations in the Japanese post-war industry is probably the best known example (Takeuchi and Nonaka, 1986). Yet, some important changes need to be made to make these ideas fit software development, which unlike other industries does not include the manufacturing and handling of physical goods (Poppendieck and Poppendieck, 2003). The most fundamental principle of lean development is that of waste reduction, i.e. all work and its products not directly contributing to the development of software should be considered as waste and thus avoided or minimized. Since the first book on Extreme Programming was published in 1999 (Beck), the interest and industrial use grew surprisingly fast (Abrahamsson, Salo et al., 2002). The enormous interest seen in industry stems mostly from the grass-root, that is the developers, and can be explained by the simple and human-centric values inherent to agile methods, which may be appealing to practitioners but perhaps threatening to management du to the de-emphasizing of traditional command and control mechanisms. The basic principles are easy to grasp, and seem to address the most fundamental problems concerning developers. However, among this interest and willingness to radically change the development process, several critical voices have emerged and many reports indicate that it is not straight-forward, and in most cases it is an act of balancing agility and discipline (Boehm and Turner, 2004).

Available agile methods such as XP, Scrum and others vary in detail and focus level. Scrum for example, can be seen as a management framework whilst XP is a collection of practices and techniques. It is quite common to combine elements from various agile methods (Kniberg, 2007) leading to a well of variants. However, in principle all agile projects share the same structure, summarized in Figure 5.

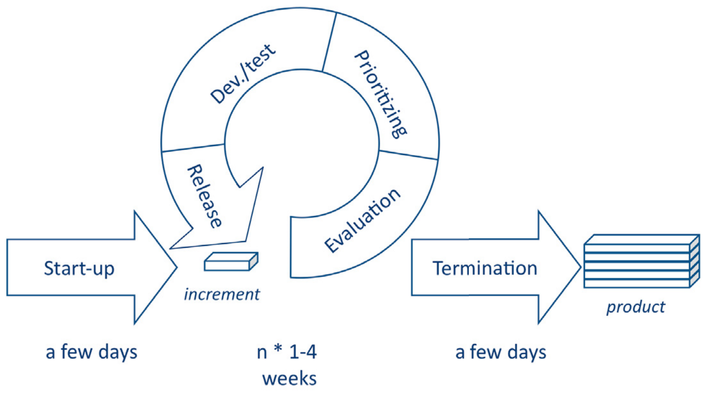


Figure 5 - Agile software development

An agile project typically has a relatively short start-up phase without extensive engineering of requirements specifications, domain models, architecture definitions etc. The major part of the project consists of a series of iterations, each lasting a few weeks normally. Each iteration starts out by evaluating – in collaboration with the customer side – the outcome of the previous iterations to verify whether the resulting product increment meets the requirements or not. Then, requirements for the next iteration are prioritized. Requirements are by principle expressed as simple and short stories. The team of developers defines estimates per requirement or feature and the customer representative does the prioritization. This forms an iteration backlog of stories to implement. Then, the development develops the next product increment, which is released – preferably as working software – to the customer representative for demonstration and evaluation. In this way, the whole development project consists of a series of iterations, each resulting in increments that in sum forms the resulting product by the end of the project (when time or money runs out).

2.1.3 Concepts combined: Agile software product line engineering

SPLE methods aim for large-scale development of product lines that serve large and diverse markets over long periods of time, whereas ASD methods address small-scale development from scratch for a single, well-defined customer. These two approaches have different home grounds. Typically when a new paradigm or field of thought is introduced and especially when it promises commercial success, researchers and industry respond by experimenting with the new ideas. Such experimentation results in these new ideas being developed in new directions. One such recent and promising development is the potential combination of SPLE and ASD (Tian and Cooper, 2006). Although relatively new, there are already a few empirical studies (Babar, Ihme et al., 2009; Carbon, Lindvall et al., 2006; Ghanam and Maurer, 2009; Ghanam and Maurer, 2010; Hanssen and Fægri, 2008; Mohan, Ramesh et al., 2010; Noor, Rabiser et al., 2008).

Carbon et al. (2006) did a simple experiment, where students were given development tasks following an agile SPLE approach based on Fraunhofers PuLSE-I framework (Bayer, Gacek et al., 2000), being extended with the practice of incremental design (which is fundamental in all agile methods), and one other practice that is specific to XP called the "planning game" (a planning meeting held in each iteration). Results showed that practices such as the planning game, for instance, includes a great potential to increase the agility of a product line organization, but also other practices like continuous integration or automated regression testing can contribute.

Noor et al. (2008) performed a small-scale industrial case study. They found that it is feasible to combine ASD and SPLE and that it gave a collaborative process relying mostly on face-to-face conversation as a means of communication, but that it requires involvement of both business and technical people to work. The authors derived three success factors from their case study. (i) It is important that participants are familiar with fundamental concepts of product line planning to allow a smooth start. (ii) Preparation is required on part of the process facilitators but also on part of the industry partner. (iii) The willingness to collaborate in a team is also essential.

Babar et al. (2009) report on a case study of a Finnish software product line organization that has been using XP and Scrum for years. Two important lessons can be learned from this study. Firstly, product development projects need documented background information about the system architecture of the product line and a project team that has tacit knowledge pertaining to and experience of constructing the system architecture. Secondly, considerable proactive exploratory work has to be done on the agile development projects before development begins. Related to these findings, the authors also observed that the studied company introduced two new roles to communicate architectural decisions and information. Hanssen at al. (2010) also observed a similar development where a software architecture team was established to serve the agile product line organization.

Ghanam and Maurer (2009),(2010) have, through two case studies found that besides being practically feasible, the combination of some XP practices and SPLE reduced rework and the cost of producing customized solutions and that it was feasible to target customers with diverse needs without having to disturb the agility of their practices. The proposed approach was found to have a potential to substantially reduce the adoption barriers of a product line practice through its incremental and non-burdening nature. The agile SPLE approach that was tested gave customer involvement a special treatment by enabling customers to pick from variants and contribute to the variability model when available variants are not satisfactory.

In a recent study, Mohan et al. (2010) applied the theory of complex adaptive systems (CAS) to define a set of practices to balance the long-term strategic objectives of SPLE and the short-term tactical objectives of ASD. Together, these practices aim to prepare the product line organization to be flexible and able to handle changing environmental conditions.

Finally, in this overview of empirical studies on the combination of SPLE and ASD we see that our own studies (Fægri and Hanssen, 2007; Hanssen and Fægri, 2006; Hanssen and Fægri, 2008) support and complement the research being discussed here. We have also, as others, concluded that it is practically feasible to implement such a combination of processes, which is supporting strategic and tactical objectives, and that an operational (day-to-day) process constitutes the experience-bearing link from the tactical to the strategic process.

To summarize, the collective impression from these relatively few existing empirical studies of the combination of ASD and SPLE is mainly that these two approaches may – in practical terms – very well be combined. On the one hand, some of the central agile practices may increase flexibility and customer collaboration. On the other hand, the concepts of SPLE are needed in order to manage the diversity of products, the larger customer base, and the long-term perspective, that are characteristics of managing and developing a product line over time.

Building on this overview of research, we take a step back and compare the key agile practices with SPLE as described in various source (Clements and Northrop, 2002; Linden, Schmid et al., 2007; Pohl, Böckle et al., 2005).

Table 2 - ASD and SPLE compared

Agile principles (http://www.agilemanifesto.org/principles.html)	**Corresponding SPLE practices**
1) Priority on early and continuous delivery of valuable software.	Dependent upon SPLE approach. A reactive model supports early delivery of valuable software (no particular bias with respect to continuous delivery).
2) Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	Foreseen changes covered by platform and variability models are cheap. Other changes become increasingly expensive if they affect domain-engineering artifacts, which may be components of multiple products, and thus increase the overall cost of maintenance.
3) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.	An SPLE approach does not reject or inhibit this principle. Product delivery is, nevertheless, somewhat slowed down compared to agile development, due to the overhead of maintaining integrity with the product platform.
4) Business people and developers must work together daily throughout the project.	An SPLE approach neither rejects nor inhibits this practice.
5) Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	At the product level, an SPLE approach neither rejects nor inhibits this practice. At the platform level, however, more formalism is required.
6) The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	SPLE, due to increased technical and/or organizational complexity, introduces the need for more explicit, formal, and disciplined communication.
7) Working software is the primary measure of progress.	SPLE incorporates significant value in the platform that is more difficult to measure. However, for product development, the SPLE approach neither rejects nor inhibits this principle.
8) Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	SPLE promotes sustainable development at a larger scale.
9) Continuous attention to technical excellence and good design enhances agility.	SPLE platforms embody technical excellence and design to support rapid product derivation.
10) Simplicity – the art of maximizing the amount of work not done – is essential.	The main motivation for SPLE is the same – but by exploiting reuse to eliminate work. In a reactive SPLE approach, the risk of unnecessary work is reduced.

Agile principles (http://www.agilemanifesto.org/principles.html)	**Corresponding SPLE practices**
11) The best architectures, requirements, and designs emerge from self-organizing teams.	The assumption of SPLE is that the investment in requirements, architectures, and designs can be reused successfully across multiple products.
12) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	An SPLE approach neither rejects nor inhibits this practice. In fact, due to investment in the platform, most SPLE organizations promote experience gathering and learning.

This comparison shows that ASD and SPLE have few principal conflicts according to our interpretation of these software engineering paradigms. They are virtually independent of each other. On most counts, they possess complimentary properties (Carbon, Lindvall et al., 2006; Tian and Cooper, 2006). Whereas SPLE attempts to introduce large-scale cross-product economic benefits through reuse, ASD focuses on delivering single products with the best possible quality for the customer. Thus, there is, prima facie, no reason why these two approaches should not be combined. On the contrary, there is a clear motivation to combine them.

2.1.4 Software Ecosystems

Software ecosystems is a very recent term, referring to a networked community of organizations, which base their relations to each other on a common interest in a central software technology. Some other definitions of this emerging concept have been proposed, for example by Jansen et al. (2009): "*a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them*" (p. 2). Another definition by Bosch (2009), focusing more on the common interest in the software and its use, is: *"the set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions"* (p. 2).

Well known examples of communities that may be seen as software ecosystems are Apples iPhone/Appstore platform and the open-source development environment Eclipse. The first is an example of a partially closed and controlled ecosystem, and the latter is an example of an open ecosystem allowing more flexibility in use and development. This simply illustrates that the ecosystem concept may refer to a wide range of configurations. Yet, they all involve two fundamental concepts: 1) a network of organizations or actors, and 2) a common interest in the development and use of a central software technology. These organizations may have different relations to the central software technology, and for this reason, different roles in the ecosystem. In our definition of the concept, there are at least three key role types. First, one organization (or a small group) acts as the *keystone organization,* and is in some way leading the development of the central software technology. The second key organizational role is the *end-users* of the central technology, who need it as a tool to carry out their business, whatever that might be. The third key role is *third party organizations* that use the central technology as a platform for producing related solutions or services. In addition to these key roles, various other related roles might be part of the ecosystem, for example standardization organizations, resellers, and operators (Jansen, Brinkkemper et al., 2009).

A fundamental property of the central software technology is that it is *extensible beyond the keystone organization* (Alspaugh, Hazeline et al., 2009). This extensibility can be achieved in various ways, for example by providing an application programming interface (API) or a software development kit (SDK), by supporting exchange of open data formats, or by offering parts of the technology as open source. Opening up the technology in these, and potentially other ways, enables external organizations to use the central software technology as a *platform* where existing services or data can be used and extended. Bosch (2009) (p. 112) proposed a Software Ecosystem Taxonomy that identifies nine potential classes of the central software technology, according to classification within two dimensions. The first one is *the category dimension*, which ranges from operating systems to applications, and to end-user programming. The second one is *the platform dimension*, ranging from desktop to web, and to mobile. The case discussed in this thesis is an *application-web category*. Accordingly, implications for practice and theory that are developed later in the thesis relate to this category.

Table 3 - Software Ecosystem Taxonomy

end-user programming	MS Excel, Mathematica, VHDL	Yahoo! Pipes Microsoft PopFly Google's mashup editor	*none so far*
application	MS Office	SalesForce, eBay, Amazon, Ning	*none so far*
operating system	MS Windows, Linux, Apple OS X	Google AppEngine, Yahoo developer, Coghead, Bungee Labs	Nokia S60, Palm, Android, iPhone
category / **platform**	**desktop**	**web**	**mobile**

A keystone organization has a special position in an ecosystem as it controls, strictly or loosely, the evolution of the central software technology. This may include various responsibilities, for example certain software product development activities such as strategic planning, R&D, and operational support. These responsibilities come in addition to activities specific to ecosystems, such as enabling efficient external extensibility, provision of insight into planning and development, and supporting ecosystem partners in various other ways.

One potential benefit of being a member of a software ecosystem is the opportunity to exploit open innovation (Chesbrough, 2006), an approach derived from open source software (OSS) processes where actors openly collaborate to achieve local and global benefits. External actors and the effort they put into the ecosystem may result in innovations being beneficial not only to themselves (and their customers) but also to the keystone organization, as this may be a very efficient way of extending and improving the central software technology as well as increasing the number of users. Closer relationships between the keystone organization and the other actors may drive both an outside-in process as well as an inside-out process, as described by Enkel et al. (Enkel, Gassman et al., 2009). Also, the proximity between the organizations in an ecosystem may enable and improve active engagement of various stakeholders in the development of the central software technology (Hanssen and Fægri, 2008).

When explaining the concept of software ecosystems it is also necessary to address how software ecosystems relate to the development of open source software (Fitzgerald, 2006). There are clear similarities between these two concepts, but also several differences, which justify the definition of software ecosystems as a unique concept. The main difference between these two relates to the underlying business model. Brown and Booch (2002) explain the open-source business model as follows:

"The basic premise of an open-source approach is that by "giving away" part of the company's intellectual property, you receive the benefits of access to a much larger market. These users then become the source of additions and enhancements to the product to increase its value, and become the target for a range of revenue-generating products and services associated with the product."

In a (closed) software ecosystem – like the case being studied in this thesis – the intellectual property (the code) is *not* shared in any way. This leads to some fundamental differences, listed in **Table 4**. (When using the term *'ecosystem'* in this thesis, we implicitly refer to the development of closed-source systems.)

Table 4 - OSS and closed-source ecosystems - similarities and differences

Similarities	A community of organizations and actors.	
	A shared interest in the development, evolution, and use of a software product.	
	Independent actors collaborate and contribute to development.	
	Open innovation.	
	New business models as compared to traditional licensed software.	
Differences	OSS	(Close source) ecosystems
	Open source code.	Closed source code.
	Ownership is shared.	Ownership and control lies with the keystone organization.
	Free use (with options for paying for specializations and related services)	Pay for use.
	Extensibility through open source code.	Extensibility through controlled interfaces.

The ultimate objective for investing in and working towards an ecosystem is that all members will derive more benefits from participating, as compared to the more traditional approach to software product development with segregated roles, a low level of collaboration, and closed processes. A well functioning ecosystem is, in summary, a complex configuration with collaboration across traditionally closed organizational borders (Bosch, 2009). Such multi-organizations are most likely not established as a deliberate planned effort. Rather, they emerge as a result of many congruent factors such as domain and business development, technology development, globalization, new collaborative patterns, and customers becoming more and more accustomed to participating in the shaping of the technology they use.

2.1.5 Organizational change

An important aspect of understanding an organization and its functions is to understand how it changes. Kurt Lewin, one of the original contributors to the study of organizational change even stated: *"You cannot understand a system until you try to change it"* (Lewin, 1951). In relation to this, he also coined the term and the concept of action research (Lewin, 1946), which is the repeated process of evaluating a situation, planning, and executing a change, evaluating the effects, and implementing a permanent change. A key feature of this model is that change is not done in a laboratory-like setting but in a real context, for instance in industry and in production. This principle is also fundamental to Lewin's original model of organizational change. According to this model, change is performed through three steps: *unfreeze – change – freeze*. This concept has been formative to the wide understanding of how organizations change. *Unfreeze* refers to an organization that exits a stable phase, an equilibrium, so the established structures and processes become open to change. *Change* refers to the actual change of the organization, and *freeze* refers to the establishment of the new organization or processes as the new standard, and the new equilibrium. This model represents a view of organizational change as planned or episodic (Weick and Quinn, 1999). A premise for such an approach is *stability* (in the business domain, technology, market etc.) – which again is a premise for planning the change and foreseeing its effects.

Stability is, however, not the case for many organizations (Emery and Trist, 1965), in particular software organizations (Baskerville, Ramesh et al., 2003), which work with rapidly developing technologies and with business domains that change in line with technology. Consequently, developing plans of change up-front, based on the knowledge of a changing and turbulent environment may not be a good idea as such plans may be obsolete, wrong, or even harmful at the time of their execution. Orlikowski and Hofman (1997) address this problem and suggest a model of improvisational change. Their model contrasts the idea of change as punctuated equilibrium (Weick and Quinn, 1999) with change as an ongoing process and a result of a series of events. They define three types of change: anticipated, emergent, and opportunity-based. Anticipated changes *"are planned ahead of time and occur as intended"*. Emergent changes are "*changes that arise spontaneously out of local innovation, which are not originally anticipated or intended*". Opportunity based changes are "*changes that are not anticipated ahead of time but are introduced purposefully and intentionally during the change process in response to an unexpected opportunity, event, or breakdown."* (*ibid*., p.4). In practice, the total change consists of an arbitrary series of these types of changes, which can be understood as responses to the internal events in the organization as well as the events in the external environment.

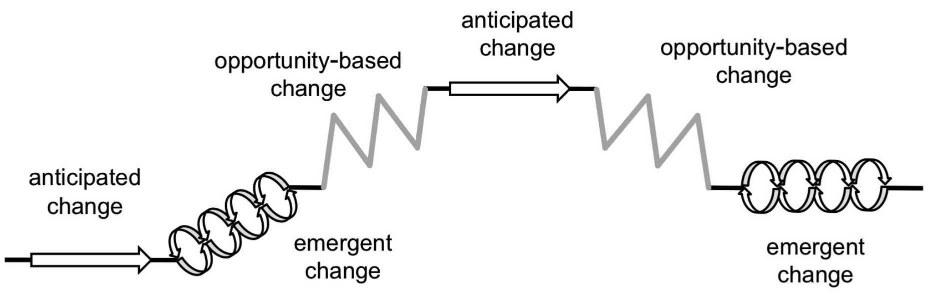


Figure 6 - Orlikowski's and Hofman's model of improvisational change

A change of an organization usually stems from the need to improve this organization; to improve a process, restructure the organization, reduce schedule, or costs etc. In line with the concept of improvisational change, and as an instrument to manage it, W. E. Deming (2000) formulated principles for transformation and the *Plan-*

Do-Study-Act improvement cycle. This is well known as *the Deming cycle*, or as *the Shewhart cycle* (*ibid*., p.88) as Deming himself named it (after W. A. Shewhart).

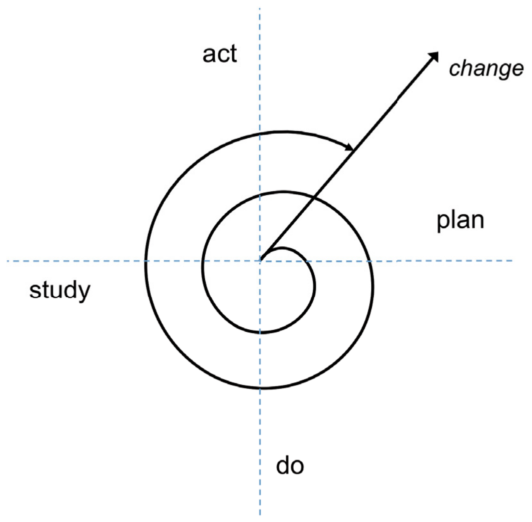


Figure 7 - The Shewhart/Deming improvement cycle

Plan – is to analyze the present situation, to observe or experience a potential for a change, and to suggest how to perform the change. *Do* – is to implement the change. *Study* – is to evaluate the change, to create an understanding of the effect and causality of the change. *Act* – is to make the change permanent, assuming that the effect of the change creates a better situation than before the change. Over time, these cycles represent a change, and an improvement. Change can be of any of the types defined by Orlikowski and Hofman, meaning that planning can vary from explicit planning to more improvisational responses to emergent and opportunity-based events.

2.2 The case

This section describes *the present* case organization, their product line and their development processes. We also give a retrospective of the past organizational change that have led to the present state.

2.2.1 Organization

CSoft is a medium-sized Norwegian software company that develops, maintains, and markets a product line under the same name. They serve the high-end segment of the market and have a wide international customer base, mainly consisting of Global 5000[3] companies. Products are marketed and sold by several resellers throughout the world. Despite facing considerable challenges in the first years, CSoft is now one of the market leaders. The company was established in 1996, and has grown continuously since then with a major increase in size through an acquisition of a former competitor in 2006. Currently, CSoft employs more than 200 people, including over 60 developers.

CSoft has several development locations across Europe and Eurasia, as well as sales and support departments in U.K. and USA. The main office, including the main section of R&D, is located in Norway.

The most relevant (to this study) roles and organizational functions are:

The product management team (PMT) is a group of five product managers, each being responsible of one of the main modules in the product line. The product managers are responsible of developing the product roadmaps, one per module, in collaboration with externals and internals. In the development projects, the product managers are the connection between the development teams, other internal roles and external actors, typically customers which act as stakeholders. Thus, this role resembles the Scrum-master role in Scrum projects, but with additional responsibilities of developing the product strategy as well as supporting development.

The development teams. Each module of the product line is developed and evolved by a fixed team of developers (no job rotation). Each team has a team leader, which is highly experienced and has the most insight into the solution and the business domain.

The Chief Technical Officer (CTO) has a special role as he has worked on the solution from the very start and thus possesses the most insight into the product line. He participates in strategic planning and plays a guru-role in development.

[3] http://theglobal5000.com/

The R&D manager plays an important role with respect to the development processes and was the initiator of the change from the plan-driven process to the agile development.

The architecture team consists of four experienced developers and has the responsibility of improving the structure/architecture of the product line and improving the technical infrastructure being used by the R&D organization. This team was formed in 2007 to counteract increasing difficulties with the growing technical complexity of the product line and the development of it.

Professional services provide support, training and adaptation of the solution to make the product work optimally for customers requesting such services. This department knows the clients, the business domain and the market very well and can be considered to be experts on the use of the product line.

Technical account management (TAM) is the customers primary contact in case of support needs or if customers want to provide input, needs or ideas. TAM communicates this type of input to the PSG, which takes it into account when developing product roadmaps.

It is important to realize that the SPLE process that is being used at CSoft is not the result of an immediate, deliberate adoption and introduction of one of the commonly used frameworks; rather it is the result of years of development and adjustments. This transition will be discussed in more detail later on.

2.2.2 Product line

The CSoft software is a highly modular product line that allows many configurations and variations in use. It contains five main modules that serve various business-related activities. In addition, it offers an Application Programming Interface (API). This is implemented as open standard web services. The composition and use of the modules varies according to customer and case. Some modules can be used in any configuration, while the use of others depends on the situation. CSoft software comes with a set of predefined configurations for the most common usage scenarios, and it provides built-in support for detailed customization to create more variants.

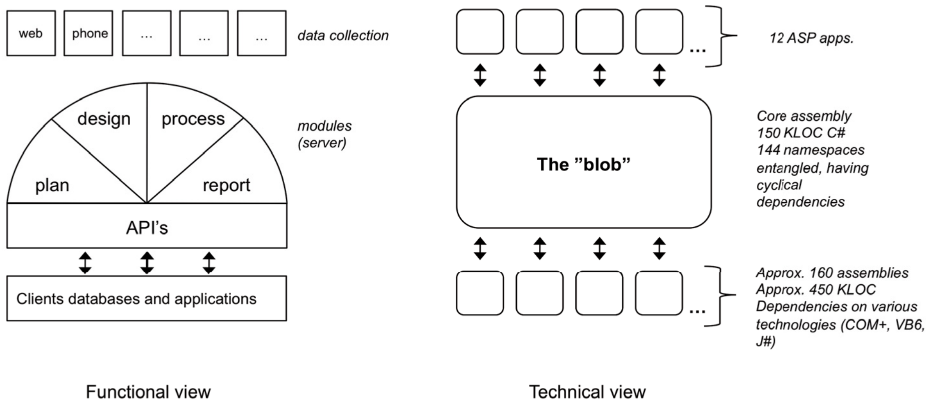


Figure 8 - High-level product line overview

The product line, or family, consists of several modules that together support a value chain of planning, data collection, analysis, and reporting of results. The composition of a specific product based on the product line can be varied in several ways:

Variability in task: Data can be collected in various ways using different modules. For example, there is one module for collecting data via telephone and one for collecting data via the Internet. All variants of the product use the same core component for storing and analyzing data, and have the same functionality for reporting. It is possible to select dynamically which features to include according to the license, in order to satisfy the greatly varying needs of the customers.

Variability in application domain: The product line is being used in three different but strongly related domains.

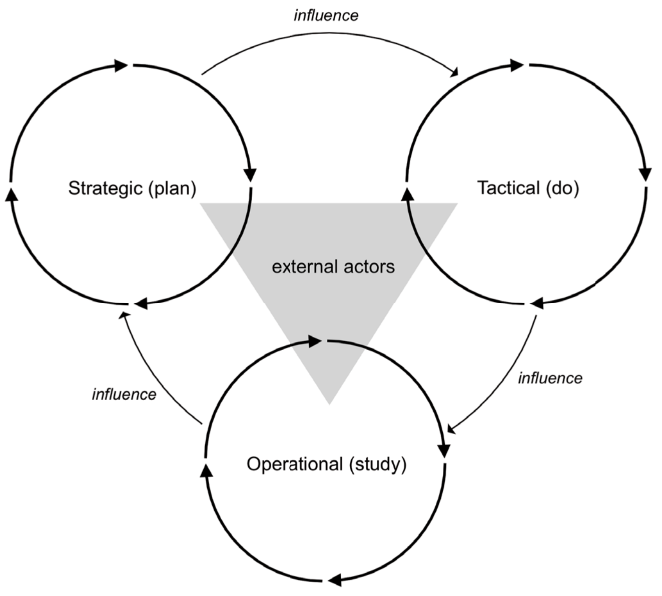
Variability in feature richness: The product comes in two main variants, the full professional suite and a simple version. The simple version offers only the basic features, is low in cost, and is not supported.

Variability in operation: The product is a three-tier application with a web-browser front-end only. This allows for a dual operational model. A customer may buy the product entirely as a service provided by CSoft, meaning that the software runs within a remote hosted environment together with all data, and all access is done through an ordinary web browser (software as a service - SAAS). Alternatively the total solution can be hosted locally. This is more suitable for customers who use the product extensively and need to retain maximum operational control. However, the largest clients only use this variant.

A common commercial model for a software product line supplier is to either sell licenses to use a product (a derivation of the line), or as a commodity for local deployment or, increasingly, as access to the software through a service. CSoft apply a different model. The principle is that cost of use is based on the extent of use; customers pay a certain amount of money per completed transaction, that is, cost is related to the actual generated business value to the customer.

2.2.3 Processes

Taking a process view of CSoft shows that the organization's development-related activities can be grouped into one of three types of processes: a long-term strategic process, an intermediate-term tactical process, and a short-term operational process. The common denominator of these three processes is their interplay with external actors (customers and third-parties that extend the product line through the API's).



	strategic (1-2 year horizon)	tactical (one release)	operational (day-to-day)
function	strategic, technological, innovation	tactical (reactive) innovation	systematize experience
process elements	SPLE practices	Evo, Impact Estimation Table (IET)	request-driven prioritization
deliverables	product roadmaps, business cases	software	support services
technical focus	CSoft software platform	release projects	fixes, support
key roles	management, PSG, CTO	R&D, PSG, CTO, architecture team	support, QA, TAM
customer collaboration	advisory board, product conference, ad-hoc	external stakeholders	end users

Figure 9 - An external-actor centric development process

Strategic (long-term, SPLE): Evolution at the SPLE level addresses the need to implement long-term strategic plans. SPLE supports CSoft's need to introduce tactical innovation, which is fuelled, for example, by innovations in technology, changes in the business models, or new business opportunities. CSoft is a product platform that contains a set of building blocks that can be tailored to provide different product offerings. This platform can be considered as the vehicle used by the Chief Technical Officer (CTO) and the Product Management Team (PMT) to implement long-term vision. More often than not, these innovations are costly in terms of effort and risk. Thus, they should be planned and introduced with great care, in order to exploit the organization's development capability. SPLE-style development inherently supports this ambition.

Tactical (medium-term, ASD): Evolution at the project level (note that a release typically includes one project per module) addresses the customer's detailed requirements and needs. Close cooperation in planning and evaluation of the two-week development iterations ensures that customers' ideas and experience with the software is translated into what we may call tactical innovation, i.e. reactive innovation that seeks to polish, improve, adjust, or otherwise make small to moderate adjustments to the product. According to the ASD method Evo, Customers' requirements are defined as product qualities and documented in an impact estimation table (IET). This is a simple list of prioritized (by a customer stakeholder) product qualities of the product under development with an estimate (by the development team) of the anticipated effort needed to implement the quality. An example of such a quality is *"Usability.Productivity"* for a given task, e.g. "*Define a simple report"*. This product quality can be defined as number of seconds needed by a novice user to complete the task. The IET also contains the goal value of the quality and the present value. Thus, the quality is improved until it has reached its goal value performance. (See Part II, paper 2, for a more detailed example). The IET correspond to the more known concept of product and sprint backlogs which are important components of the more known and practiced agile method Scrum (Kniberg, 2007).

Each development project follows a common process template of successive fortnightly iterations:

Table 5 - The iteration week plan

Week 1	One of the first days	Meeting with external stakeholder Iteration postmortem
	1 Monday	Technical meeting
		Development iteration kickoff
	2 Tuesday	Development
	3 Wednesday	
	4 Thursday	
	5 Friday	
Week 2	1 Monday	
	2 Tuesday	
	3 Wednesday	Internal demo (prepare for next meeting)
		Evo iteration planning and review meeting
	4 Thursday	Fix change requests
	5 Friday	Retrospective
		Project Status Meeting

Operational (short-term, day-to-day operations): Customers rely on the software to help them implement vital business functions in their own organizations. Thus, it is of great importance that CSoft is able to sustain a good level of customer satisfaction with the software in its day-to-day use. During operation, the products are exposed to more users, and this exposure increases the potential amount of feedback to the innovations implemented at the strategic and tactical levels.

2.2.4 A retrospective of the development of the organization

CSoft was established in 1996 and has grown continuously since then, with more than 200 employees in 2010. Their development went through three phases and has now entered a fourth. This section presents a summary of these phases of development and indicates some important milestones in the development of the organization. The timeline in Figure 10 shows main events in the development of the organization and the growth in staff. We have identified four phases of the organizational development.

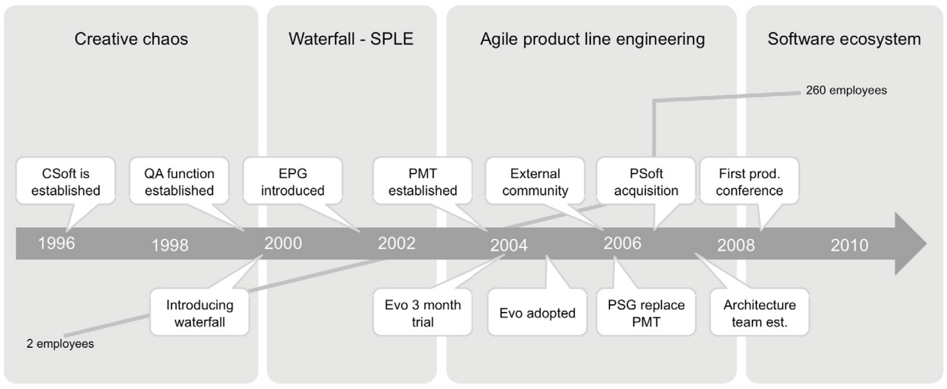


Figure 10 - Timeline of the development of the case organization

1996-1999: "Creative chaos"

The company initially grew out of a small business providing manual services to very few customers. A simple homemade supportive software tool developed into a solution that could be sold as a stand-alone software product. The main focus of the company changed, and the development of this product became the main objective. In the beginning, in 1996, there were only a few employees providing this product to a handful of customers. The process can best be described as ad-hoc as the main driver was feedback from customers obtained in almost daily interactions. A customer request was literally routed directly to the developers. This start-up phase was a creative chaos – that is, it had nearly no plans and no control but was undoubtedly extremely creative and productive. The product grew rapidly, not only in terms of features and functionality but also in terms of defects and complexity. Work became stressful, with little control and a lot of overtime work.

1999-2003: Waterfall and software product line engineering

As the number of customers grew, the organization formalized the development process according to the principles of the waterfall model, famously described by W. W. Royce (1970). This somewhat disreputable approach (Boehm, 2006; Takeuchi and Nonaka, 1986) to organizing software development emphasizes upfront detailed planning of requirements, design, and development. The development is divided into consecutive phases, where requirements are developed into a design, and the design is developed into a software system, which is tested close to deployment.

Prior to this, the R&D department was extended with a QA-manager and a few other supportive roles. The waterfall-like process was also documented as an electronic process guide (EPG), being available to all developers as a tool to spread knowledge and control process compliance (Moe, Dingsøyr et al., 2002). This structured approach established a certain level of control and helped the organization in the continuing development of their product, which grew alongside the customer base. After a few years, several problems arose clearly related to the waterfall approach (Johansen, 2005); testing and verification was postponed until late stages leading to late identification of problems which in turn caused much rework. Also, requirements were nearly solely focused on functionality, leaving out the quality perspective. The product expanded and eventually became a product line, being capable of serving various usage scenarios. To manage this increasing complexity a Product Management Team (PMT) was formed – a group of experienced employees with other responsibilities that were supposed to spend part of their time in strategic product planning. In addition various specialized functions, beyond software development, were introduced such as Technical Account Managers (TAM), the operations department, training services etc.

2004 – approx. 2008: Agile product line engineering

Due to a critical decline in process performance, management of R&D was looking for a way to improve the situation. At a software engineering conference, a few representatives from the company learned about the evolutionary development and the Evo method (Gilb, 2005). As it seemed to address some of their concerns they initiated a three-month test period to try out this radically different development approach in a real release project. Instead of an extensively prepared process adoption, they started out with a few principles, focusing on requirements management, where functional requirements were replaced by explicit expression and evaluation of product qualities, preferably stated by customers involved in the development process (Hanssen and Fægri, 2006). Early experience showed that the amount of issues near release was reduced, and that the delivered product matched customer expectations better than before. After this initial process trial, Evo was adopted on a permanent basis (Fægri and Hanssen, 2007; Hanssen and Fægri, 2008). Alongside the growth in organization and product line the PMT group was re-established as a full-time Product Strategy Group (PSG) with a dedicated management by a Chief Strategy Officer (CSO). Another supportive service, the architecture team, was established, originally with three full time members. Their task was to handle the excessively high level of system complexity, and to support R&D in architecture-related issues. In 2006 CSoft acquired a former competitor and boosted the number of employees up to more than 200. Adding new offices, for both R&D and marketing, was a considerable challenge. Through extensive internal training in the following year, the new organization was using Evo almost exclusively as the development process, and was following the release cycles.

Approx. 2008 →: Towards a software ecosystem

Since the first use of Evo, the actual use of it has further matured and was adapted to the organization's needs. During the past few years an external community has emerged, using the CSoft product line as a platform for value-adding services and products. CSoft supports the operation of this community through the offering of a set of API's for extending the product line. Also, during the most recent years, collaboration with external stakeholders, first of all customers, has been professionalized, and the first product conference was held in 2008 with extensive participation from employees, customers, prospects, and external partners. In sum – the organization is opening up and is increasing collaboration with external actors. We see this as a transition towards a *software ecosystem*.

2.3 A theoretical framework: organizational ecology

In search for a foundation to understand and communicate our findings from the study of CSoft we look to the pioneering work done by Eric Trist, Fred Emery et al. from the Tavistock Institute of Human Relations[4]. Their studies of the development and behavior of work groups and organizations have lead to theories on socio-psychological, socio-technical and socio-ecological theories whereof we look into the two latter ones.

Socio-technical theory is a view of the organization as the sum of and interplay between a social system *and* a technical system. That is 1) the people, their relations, their knowledge, and how they work together as a whole, and 2) the tools and techniques being used to perform the work. The fundamental principle of socio-technical theory is the interdependence between these two systems, meaning that to improve the performance of the organization (productivity, quality of work etc.) both subsystems have to be considered at the same time. Changing one affects the other. This

[4] http://www.moderntimesworkplace.com/archives/archives.html

principle was first drafted by Trist and Bamforth in 1951, based on their studies of changes of work processes in coalmines (Trist and Bamforth, 1951).

Through studies that followed, the theory has been developed to consider how organizations relate to their external environment (Emery and Trist, 1965). This implies that an organization has to be understood as both 1) the internal interplay between a social subsystem and a technical subsystem (the socio-technical system), and 2) the interplay between the organization and its external environment.

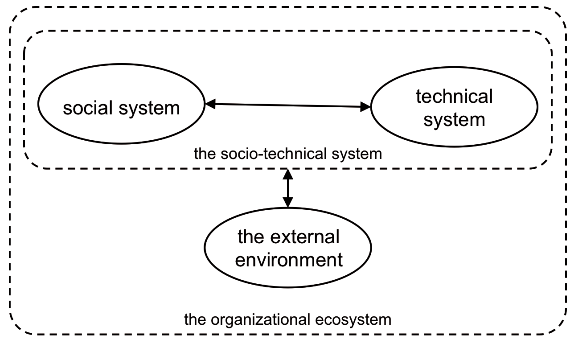


Figure 11 - The socio-technical system and the organizational ecosystem

Emery and Trist (*ibid.*) developed a simple classification system of four types of organizational environments – forming a series, in which the degree of causal texturing is increased. Thus, understanding and ordering the types of environments is useful in understanding socio-technical systems beyond the limits of a single organization.

1) The first and simplest is *the placid randomized environment* where “goods” and “bads” are unchanging, and are randomly distributed in the environment (*ibid.*, p.7). The optimal strategy is to do one’s best on a purely local basis – there is no difference between strategy (planning) and tactics (execution).

2) The second type is *the placid clustered environment* where “goods” and “bads” are not randomly distributed but band together in certain ways. Strategy is different from tactics, and survival becomes critically linked with what an organization knows about its environment. Organizations in this environment tend to become hierarchical, with a tendency towards centralized control and coordination (*ibid.*, p.8).

3) The third type, *the disturbed reactive environment*, is an environment where there is more than one organization of the same kind. The existence of a number of similar organizations becomes the dominant characteristic of the environmental field. These organizations compete, and their tactics, operations, and strategies are clearly distinguished. The flexibility encourages a certain decentralization, and it also puts a premium on quality and speed of decisions at various peripheral points (*ibid.*, p.9).

4) The fourth, and the most recent type is *the turbulent fields*. This type implies that significant variances arise from the field itself, not simply from the interaction of the component organizations. Three trends contribute to the emergence of these dynamic field forces: i) the growth to meet type-three conditions, ii) the deepening interdependence between the economic and other facets of the society, and iii) the increasing reliance on research and development to achieve the capacity to meet competitive challenge. A change gradient is continuously present in the field (*ibid.*, p.10).

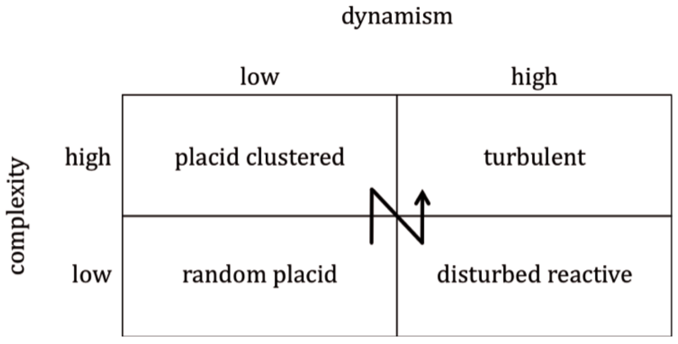


Figure 12 - Types of organizational environments

This interplay, inherent in turbulent organizational environments, has been further studied, leading to the development of the concept of **organizational ecology** (Trist, 1977). It is particularly relevant to organizations operating in complex and unstable domains. Viewing the organizational environment as an ecosystem means that it is considered to be an open system as opposed to a closed one. Organizational borders are permeable, and organizations relate dynamically to other organizations in the same field.

Developing the ecology concept further, Trist (*ibid.*) describes three classes of organizations within the turbulent environment. In a Class 1 system, member organizations are linked to a key organization among them. The key organization acts as a central referent organization, doing so even though many of the member organizations are only partially under its control, or linked to it only through interface relations. Interface relations are as basic to systems of organizational ecology as superior-subordinate relations are to bureaucratic organizations. Interface relations require negotiation as distinct from compliance – a basic difference between the two types of systems. In a Class 2 system, the referent organization is of a different kind. It is a new organization brought into being and controlled by the member organizations, rather than being one of the key constituents. A Class 3 system has no referent organization at all.

Technocratic bureaucracies have been the natural organizational form for disturbed-reactive environments (up to the 1960's or so), and this is a form that has been applied to many software engineering organizations. However, this type of system fails to adapt to conditions of persistent and pervasive environmental turbulence, mostly because it is constructed and optimized to work well in stable environments. This leads to the emergence of the new ecologically oriented systems, which show clear differences in that they promote self-regulation (as opposed to centralized control), and that they have a networked character (as opposed to segregated organizations). According to Trist (*ibid*., p.172), such systems, lacking formal structure, exist through the use of technology. Further, they also need shared values. Trist used the example of the 60/70's youth-culture that had a set of distinct (political) values. A more appropriate example from a business perspective is a shared value in growth and profit. An even more low-level example may be the software product as a shared value.

Unlike the micro-level systems (the single organization) and the macro-level systems (society), the intermediate level systems (organizational ecosystems) are hard to observe, understand, and describe due to their weak structuring. They are also the most recent type, so there is less experience with them. This relates especially to software engineering ecosystems, which is a new but rapidly advancing concept (Bosch, 2009; Messerschmitt and Szyperski, 2003). This approach is driven by the Internet as a rich and fast collaborative platform (the technology), and a common interest in the product line (the shared value).

3 Research Approach

This section will explain the chosen research approach. We begin with a general discussion of the variety of approaches to research in software engineering. This overview forms a background, and it is used to argue for the approaches we have chosen and applied, and it justifies why we disregarded others. We then continue by explaining the components of the applied research design of this thesis: how we selected the case, how data were collected, and how data were analyzed. Finally we discuss issues related to the validity of the results.

3.1 Studying software engineering

Software engineering (Vliet, 2002) is the discipline of specifying, designing, developing, deploying, and maintaining software systems. This includes many types of interdependent tasks such as management, budgeting, requirements management, application design, architectural design, coding, testing, maintenance, deployment and several other high- and low-level tasks. Most commonly, software in an industrial setting is being developed within the project format, which includes one or more suppliers and one or more contractors. However there are many variations, from developing tailor-made solutions for a well-defined customer or group of customers, to the production of standard commercial products being offered to a large and diverse market. Regardless of the variant, the basic task is to capture and understand requirements, needs, and ideas – tacit and explicit – to engineer a solution that meets requirements, within limits of time and cost. This can be a challenge as requirements are evolving or may be unclear, thus requirements engineering is an important practice within software engineering. Accordingly, testing is also a challenge; the engineered solution must be validated to ensure that it meets the requirements.

Software engineering is both a technical process and a human-intensive process. To understand this diverse practice we thus need to understand the interplay between technology and humans, who are typically organized in teams.

The term *empirical software engineering* (Basili, 1996) describes the discipline of applying empirical methods to measure, evaluate, and thus understand the practice of software engineering in general, and tools, techniques, and practices in detail. Such an understanding is needed by the industry to select an appropriate approach. The term *empirical study* (Wohlin, Runeson et al., 2000) refers to the practice of observing an object of study to collect experience-based data, and describing the object through its features, effects, and outcomes. Data are analyzed in order to understand how the object works and evolves within a given context. Typically within software engineering, the studied objects are software development methods and guidelines in general, as well as design techniques, tools, programming languages etc. Data of interest are mostly in some way connected to costs and gains, for example the level of code quality related to a given testing technique. In its most general form, empiricism is the process of learning from experience, thus being able either to validate theory or build new empirically qualified theory. This process should be transparent, explaining how data were collected, how they were analyzed, and how they relate to the existing research. Within the software engineering field, the terminology is somewhat diffuse: besides the term empirical software engineering we can find other similar terms such as evidence-based software engineering (Kitchenham, Dybå et al., 2004), experimental software engineering (Basili, 1996), and other variants. However, they all refer to the basic concept of systematizing *experience* in order to generalize new knowledge – as in most other sciences.

To place the practice of empirical software engineering in a broader context, Figure 13 shows how it relates to the software engineering discipline, and how it forms connections between research, industry, and users of technology:

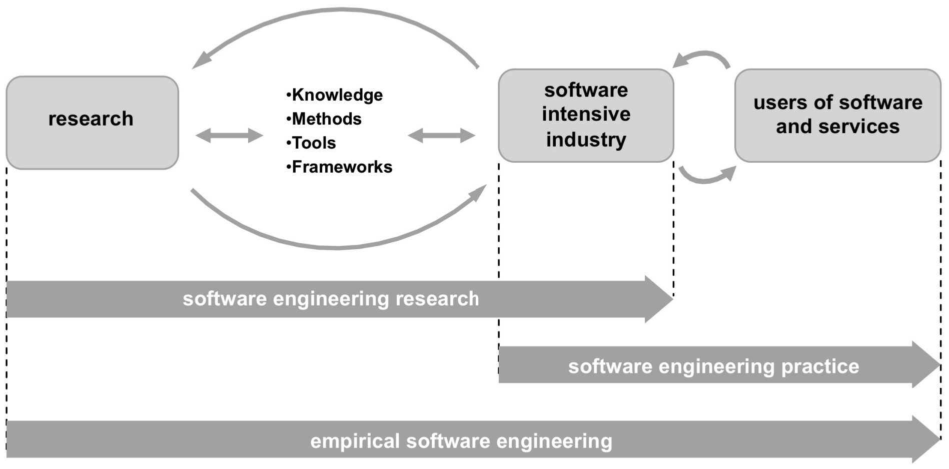

research
•Knowledge
•Methods
•Tools
•Frameworks
software intensive industry
users of software and services
software engineering research
software engineering practice
empirical software engineering

Figure 13 - Empirical software engineering

This value chain shows the role of the software engineering industry and its relations, both to their customers and users, and to the research community. The value chain here is two ways; from left to the right it shows how knowledge, based on experience (empiricism), is applied in the construction and application of software systems. Going the other way, it shows how experience from the use of software systems and development processes is captured and externalized. In this way the figure shows a learning loop where empirical software engineering is the vehicle.

There exist several strategies and tools to plan, execute, and report empirical studies. These are general approaches applicable to many domains and disciplines, including software engineering. Software engineering is in many respects an immature engineering discipline, and so is empirical software engineering. For example, looking to medicine or social sciences, we find a much more mature and widespread use of empirical methods that we can learn from (Kitchenham, Dybå et al., 2004). However, there are differences between fields, and software engineering has its peculiarities that influence how we can utilize these methods. Further on, we will give a brief overview of a set of empirical approaches with emphasis on the field of software engineering.

Wohlin et al. (2000) refers to two main types of empirical studies: qualitative and quantitative, a very high-level taxonomy of research approaches.

Qualitative studies refer to studies where data is collected from humans (either directly or through their actions) using techniques such as interviews, surveys, and observations etc. This type of data is to some extent subjective and can be biased, since the subject's or the observer's viewpoints can be affected by belief and personal standards. Qualitative research is mainly concerned with phenomena not possible to be expressed or measured quantitatively. Raw data is often represented in the form of words (written or spoken) or as pictures or illustrations (Seaman, 1999), or through instruments like questionnaires using psychometric scales (Cooper and Schindler, 2006). The type of data being collected naturally influences the type of analysis that can be performed. Typically, qualitative data cannot be analyzed mechanistically but need to be interpreted in some way, thus risking a bias. Qualitative research methods have been developed mainly by researchers in the social disciplines focusing on humans, their relations, and their actions (Seaman, 1999).

Quantitative research refers to studies that quantify properties of the phenomenon being studied, thus representing data as numbers which can be of nominal, ordinal, interval, or ratio types (Cooper and Schindler, 2006) (p. 312). This approach comes mainly from the natural sciences where data is available through measurement either by direct empirical observations or by the use of instruments and calculations. Having data in a numerical form makes it possible to do sample-based statistical calculations to find values describing populations and to compare results mathematically (less, greater or equal). Doing quantitative studies also makes it possible to demonstrate the quality of the analysis through measures of significance and validity, which are important factors when findings are to be applied in practice. However, there are many pitfalls that may introduce errors during data collection, analysis, and reporting, for instance poor construct validity.

Wohlin et al. define three high-level types of empirical strategies: surveys, case studies, and experiments. Variations of these and more types can be added, as there is no commonly accepted taxonomy of empirical strategies within the field of software engineering as yet.

A survey is a retrospective type of study where data, quantitative or qualitative or both, are collected after an event, thus not being able to control the event, for example a development project. Questionnaires and interviews are commonly used techniques for data collection. Data can be interpreted to reach either descriptive or explanatory conclusions.

A case study is basically a study of a phenomenon over time, which makes it an observational type of study. Both quantitative and qualitative data may be collected, either alone or in combination. A combination, often referred to as triangulation, is often productive as one type of data may support or complement another type (Davison, Martinsons et al., 2004). The term case study is not well defined, and it may take many forms with respect to intervention, data collection techniques, and analysis (Yin and Campbell, 2002), however it can be defined as simple as 'the study of a restricted case'.

Experimentation implies a high level of control (as opposed to a survey). An experiment can be done in a laboratory-like setting or in a real-life (field) setting. However, the most prominent feature is the ability to control the variables being studied. Control, in this context, means that the values of the variables are known through some kind of manipulation, and can be measured with a certain known level of accuracy. There are three types of variables:

1. Dependent variables, also called response variables, which represent the effect or outcome in an experiment. An example of a dependent variable may be the productivity or quality of a software development process, which again can be operationalized in various ways.
2. Independent variables are controlled or manipulated in the experiment setting. Examples are the type of design technique or development tool being used by different experiment groups. An experiment can be planned to evaluate the co-variance between several variables. The main goal of the experiment is to study how the independent variables influence the dependent variables or how changes in variables correlate.
3. In addition there may be context variables, which are also believed to have an influence on the dependent variables. It is important to know these variables and their (interfering) effect on the dependent variables to be able to better evaluate the direct relationship between the independent and the dependent variables.

Control may ensure accuracy and data of high quality, but it also imposes costs and a risk of bias. As stated by several authors, e.g. (Fenton, 1994; Kitchenham, Pfleeger et al., 2002; Tichy, 1998), both the level and the quality of experimentation is poor in the field of software engineering.

Wohlin et al. present one interpretation of the term controlled experimentation, but there exist others. Zelkowitz and Wallace (1998) give a more detailed definition and explanation of the term as they suggest a taxonomy for software engineering experimentation. First, they define four categories of experimentation: *the scientific method* (a theory explaining a phenomenon is validated through testing of a hypothesis variation), *the engineering method* (a solution to a hypothesis is developed, tested, and refined until no further improvement is needed), *the empirical method* (a hypothesis is validated by the use of statistical methods), and *the analytical method* (a formal theory is developed and results are compared with empirical observations). Through a review of multiple examples of technology validation approaches, Zelkowitz and Wallace identified 12 different experimental approaches grouped into three categories: observational, historical, and controlled. In the observational category they define project monitoring, case study, assertion, and field study. In the historical category they define literature search, legacy, lessons learned, and statistical analysis. In the third category, controlled, they define replicated experiments, synthetic environment experiments, dynamic analysis, and simulation. Some of these approaches resemble the description by Wohlin et al. Some terms are used in a similar ways, others differ.

Some other descriptions give other variations, different focus, and levels of detail, for example in the paper *Preliminary guidelines for empirical research in software engineering* by Kitchenham et al. (2002). The title is unclear, as the paper focuses mostly on experimental design, using terms slightly different from the other referenced sources. This simply illustrates the diversity of definitions and descriptions for methods of empirical research within software engineering. This diversity is believed to contribute to a certain level of confusion in the various study reports that mix terms as well as in the actual use of methods, and as a consequence, an unsatisfactory level of quality of empirical studies (Dybå and Dingsøyr, 2008).

In addition to the empirical strategies discussed here, some other approaches also deserve to be described. First, *action research* may be viewed either as an empirical strategy or a research method. However, it is mostly a framework that defines how the researcher(s) should cooperate and intervene with client(s) to plan, carry out, and evaluate a study. Davison et al. (2004) defines the concept of canonical action research through five principles. These are *diagnosing* (what is the problem?), *action planning* (what can we do about it?), *intervention or action taking* (do it), *evaluation* (assess the effect of the change with respect to the defined problem), and finally *reflection* (learn from the experience, both specifically and generally).

Another important meta-strategy related to empirical research is systematic literature reviews as described by Kitchenham (Kitchenham, Pfleeger et al., 2002). This is an approach to systematically and rigorously search, evaluate, and systematize published research results in a given field or for a given topic or problem. Systematic reviews are not an empirical strategy in the direct sense. However, it is a useful approach to interpret and understand published results from existing empirical studies to build a wider and composite view and understanding, or at least an updated overview of a research field.

The strategies briefly presented and discussed here do not give the whole picture; there are variants and combinations of these, as well as other approaches that may have been missed out in this overview.

Common to most strategies and guidelines for empirical research, regardless of flavor, is the definition of the overall process describing a set of phases. These do vary, however in general they address the following five phases:

1. *Problem definition* - that is, identifying and justifying a problem or question, sometimes related to a need for improvement. This may also include planning of a potential action to correct the problem or improve some performance. This phase is crucial to identify the most important problem to be addressed with respect to criticality and significance.
2. *Planning of the study* may include the definition of one or more hypotheses to be tested or some other type of activities to evaluate a case (producing experience) to gain knowledge about the defined problem. For historical types of studies such planning may be irrelevant, but in any case requiring some kind of intervention or data collection during an event, this must be planned.
3. *Operation or execution* is the implementation of the plan. This may cover manipulation of a subject, e.g. to carry out a treatment, or simply observations to collect data without any intervention.
4. *Analysis* is the phase where data is interpreted either to test a defined hypothesis or to build, edit or verify a theory or hypothesis.
5. *Presentation and dissemination* is the final phase where results are made available externally, thus constituting a potential addition to the existing knowledge base on the topic.

3.2 The applied research process of the thesis

As we have shown, the variety in study approaches and techniques is great, and it depends on the type of research question and the context of the study. The particular study being reported in this thesis can be described using the following attributes:

(1) A case study of a single industry organization.
(2) An interpretative field study.
(3) A longitudinal study.

Given these premises for the study of CSoft, we have applied the following principal guidelines for planning and conducting our studies:

(1) Case study: All four studies underlying this thesis have been exploratory and qualitative, and the focus and the defined research goals were rather widely defined. This made the *case study* approach the preferred overall research strategy. In contrast, controlled experimentation based on a few isolated and controlled variables, and their hypothesized relationships was not found to be suitable for understanding the selected case organization and their processes, especially as we were interested in following a development over time and its unforeseen outcomes. Further more, the state of research on agile methods, which lies at the basis of this study, is in a rather nascent state. Despite the widespread adoption of agile processes and techniques in the software industry - and the correspondingly large interest in academia - there are relatively few empirical studies of good quality (Dybå and Dingsøyr, 2008). There is nearly no theorizing in the present studies. This calls for more qualitative and explorative studies (*ibid.*, p.852).

According to Yin (2002), case studies are the preferred research strategy "*...when a «how» or «why» question is being asked about a contemporary set of events over which the investigator has little or no control.*" (p. 9). This is a very adequate description of the context of our studies of CSoft.

(2) Interpretive field study: More specifically, our study of CSoft can be classified as an *interpretive field study* according to a definition given by Klein and Myers (Klein and Myers, 1999). This definition includes seven principles for conducting interpretive field research that suits the aim of our study:

Table 6 - The application of the seven principles of interpretive field studies

Principles (from Klein and Myers (Klein and Myers, 1999), p. 72)	Practiced in the case study
1. The Principle of the Hermeneutic Circle This principle suggests that all human understanding is achieved by iterating between considering the interdependent meaning of parts and the whole that they form. This principle of human understanding is fundamental to all the other principles.	Data are collected through repeated interviews with actors playing various roles. The data collection is supported by observations and collection of relevant documentation. The growing knowledge of the case has guided the data collection.
2. The Principle of Contextualization Requires critical reflection of the social and historical background of the research setting, so that the intended audience can see how the current situation under investigation emerged.	The study of the case is conducted from two viewpoints – the present organization, and how it has emerged over time.
3. The Principle of Interaction Between the Researchers and the Subjects Requires critical reflection on how the research materials (or "data") were socially constructed through the interaction between the researchers and participants.	A large part of the collected data is based on semi-structured interviews (Seaman, 1999) that followed open interview guidelines to ensure a balance between thematic focus and room for reflection, correction, and discussions. This allows for unplanned but relevant topics to be addressed.
4. The Principle of Abstraction and Generalization Requires relating the idiographic details revealed by the data interpretation through the application of principles one and two to theoretical, general concepts that describe the nature of human understanding and social action.	Findings are related to the concept of organizational ecology (Trist, 1977). Key principles from this theoretical background are applied to the studied case (section 5.3).
5. The Principle of Dialogical Reasoning Requires sensitivity to possible contradictions between the theoretical preconceptions guiding the research design and actual findings ("the story which the data tell") with subsequent cycles of revision.	The theory applied to the case was not used to plan and guide the data collection. The applicability of the theory became evident through the analysis *after* the data had been collected.

Principles (from Klein and Myers (Klein and Myers, 1999), p. 72)	Practiced in the case study
6. The Principle of Multiple Interpretations Requires sensitivity to possible differences in interpretations among the participants as are typically expressed in multiple narratives or stories of the same sequence of events under study. Similar to multiple witness accounts even if all tell it as they saw it.	This principle was followed by collecting data from both external actors and people with various roles in the product line organization.
7. The Principle of Suspicion Requires sensitivity to possible "biases" and systematic "distortions" in the narratives collected from the participants.	The data were collected and analyzed by the author, who is external to the organization, having no formal responsibilities, interests or agenda, except to create an unbiased view of the organization and its development.

(3) Longitudinal study: An important feature of the study of CSoft is that it was a longitudinal study with data collected from 2004 to 2009. This means that we can describe how the organization has *developed over time*, some of the reasons for the changes that took place, and the results of these changes. From the start we have acknowledged that the organizational change is usually not a plan-driven and strictly controlled process. Given turbulent, flexible, and uncertain organizational and environmental conditions in today's industry, such an approach would be less appropriate (Orlikowski and Hofman, 1997). On the contrary, power, chance, opportunity, and accident are as influential in shaping outcomes as are design, negotiated agreements, and master plans (Pettigrew, 1990).

When the focus of a study is the change of an organization, a longitudinal approach is the only approach to understand the change. Tushman and Romanelli (1985) (p. 174) say:

> *"The call for longitudinal, historical perspectives stems (1) from a pervasive dissatisfaction with static, cross-sectional views of organizations that illuminate covariant attributes to organizations, but tell little of the impact of history and precedent on current organization behavior; and (2) from simple curiosity for answers to such questions as, "How and why did this firm evolve? Why did certain firms succeed while others did not?"*

This view is supported by Pettigrew (1990) who describes a theory of method for longitudinal field research on change. His fundamental view is that theoretically sound and practically useful research on change should explore the *contexts, content, and process of change* together with their *interconnections through time* (*ibid*., p.268). We have applied these guidelines by explaining the context of our study and the change that the case organization has undergone (see section 2.2.4).

3.2.1 The selection of the case organization

The selection of CSoft as the sole organization to study was based on a long-term relationship between the researchers and the company. The author and others have studied this company and their product line more or less continuously over ten years (Johansen, 2005; Moe, Dingsøyr et al., 2002). The study reported herein adds to the previous studies. Together, the reports describe how CSoft have moved from a start-up company to an organization with a mature product. The present day status of the organization is not the result of a sudden adoption of a defined SPLE method; rather, it is the result of years of organizational development and evolution. This organization and the course of its development is, therefore, a relevant subject for an exploratory and descriptive case study, such as the one reported here. CSoft is a particularly interesting case, given that the organization's use of SPLE also incorporates some of the fundamental principles of ASD. This combination of the two approaches is the result of being able to move outside disciplinary boundaries (Hughes, 1989). Other organizations considering making similar choices may well find it productive to turn to the CSoft case as a valuable source of experience.

When the object or phenomenon being studied is a large, diverse, and complex organization like CSoft, gaining access to necessary data may be difficult, for reasons of both practicality and confidentiality. In the case of CSoft, a high level of trust has been developed over several years, making it possible to access people and data that would otherwise not be available. Such sources may reveal information confidential to the organization and include even negative information about the company or its product. To formalize this relationship, we have signed a non-disclosure agreement with CSoft.

The empirical studies underpinning this thesis have all been conducted within this single case organization, CSoft. The first study started in 2004 (reported in Paper 1 (Hanssen and Fægri, 2006)), the last finished in 2009 (reported in Paper 6 – a manuscript submitted to a special issue of the Journal on Systems and Software). Together, four consecutive studies were reported in six publications, covering approximately five years of the development at CSoft.

3.2.2 Data sources

Nearly all data that were collected throughout the studies of CSoft were qualitative, and their sources were interviews, workshops, observations, and collections of relevant documents. All interviews were semi-structured, and most of them followed a predefined interview guide but with a clear rule of conduct for allowing discussions outside the guide (meeting the third principle of Klein and Myers (1999) p.72). Table 7 - Data sources, shows which interviews that were based on a guide and appendix A lists all interview guides used in all four studies as well as some background of other types of data collection. In total, we did two group interviews and twenty single person interviews. All interviews and some observations were recorded digitally and transcribed completely. Observations that were not suitable for recording, for example ad hoc conversations were documented through notes. In this way, all data became textual. Table 7 lists all collected data, explaining study number (ref. Figure 1 - Study design), data source and whether appendix A gives more details on interview guides and techniques for data collection. In addition to these data, we have had continual discussions with various members of the case organization throughout the whole study. The outcome of such discussions has not been documented explicitly but has contributed to a better understanding of the case. The table also indicates which papers that report the data. Each paper give more detail on data collection, analysis and results.

Table 7 - Data sources

Study	Data sources	A
Study 1 *(paper 1 & 2)*	**Interviews**	
	1. One group interview with six developers using the PMA technique (Birk, Dingsøyr et al., 2002)	✓
	2. Three semi-structured interviews with the participating customers	✓
	3. Five semi-structured interviews with the PMT members	✓
Study 2 *(paper 3)*	**Interviews**	
	4. One semi-structured interview with the CTO	✓
	5. One semi-structured interview with the PMT manager	✓
	Documents	
	6. Six business plans for product line modules and main features	
Study 3 *(paper 4)*	Reusing data from study 1 and 2. New data:	
	Interviews	
	7. One group interview with two of three members of the architecture team. The interview was done after the architecture analysis as an open discussion about software entropy.	
	8. One semi-structured interview with one of the team leaders	✓
	9. One semi-structured interview with a team member/developer	✓
	Other	
	10. Architecture analysis using NDepend[5]. Results are documented in (Hanssen, Yamashita et al., 2010; Hanssen, Yamshita et al., 2009; Smaccia, 2008).	
Study 4 *(paper 5&6)*	**Interviews**	
	11. One interview with the R&D manager	✓
	12. One interview with the manager of Professional Services	✓
	13. One interview with the PSG manager	✓
	14. Three interviews with PSG members	✓
	15. One interview with the a Technical Account manager	✓
	16. One follow up interview with the PSG manager to clarify issues and notes after observation of customer review meeting (source #18).	
	Observations	
	17. Product conference, London 2008	✓
	18. Customer review meeting, 2008	✓
	Documentation	
	19. Five module product plans, four presentations at the 2008 product conference (CSO's, CEO's, VP Product Marketing's presentation and customer's presentation)	

[5] http://www.ndepend.com/

The selection of interview respondents throughout the studies have been based on a growing knowledge of the case in combination with the focus and research aim of each study. For some cases, we have based the selection of interview respondents based on the input or even direct guidance from the previous respondent, an approach similar to snowball sampling (Cooper and Schindler, 2006) (p.204).

Due to the geographical distance between the researchers and the case organization as well as very limited opportunities for on-site visits for observational studies, a large portion of the collected data have been done through telephone interviews. Further on, since a large part of the data in the studies comes from interviews we risk that the collected information can be biased, incomplete or even wrongful due to misunderstandings, lack of insight etc. To reduce this risk we have (when possible/feasible):

(1) Done interviews with several individuals sharing the same function in the organization. For example, in study 1, we interviewed three customer representatives and all five members of the product management team. In study four we interviewed four out of five members if the product strategy group.

(2) We have also sought to cover a variety of organizational roles to get various viewpoints in order to be able to build a balanced overview of the organization, their product line and their development processes.

(3) Besides interviews, we have done observations and collected documentation, which have been used to complement data from the interviews.

3.2.3 Data analysis

When working with qualitative data, such as interview transcripts, documentation, and field notes, several strategies are available for analyzing the material. These depend partly on the form of the data and partly on how the results are going to be presented. Langley (1999) defines seven strategies for what she calls *sense making*. Of these, the grounded theory strategy was found to suit the CSoft case study best because it is appropriate for the analysis of eclectic and ambiguous data. Other strategies have a varying potential for accuracy, simplicity, and generality.

According to Langley, grounded theory has the potential for high accuracy, but it can be difficult to move from substantive theory to a more general level. The level of generality that can be achieved depends on a number of factors, such as the degree of the scope of replication and the source of ideas. In the study of CSoft, the combination of SPLE and ASD was investigated. These are both well-established strategies for software engineering, each with a considerable legacy of research, models, methods, and nascent theoretical foundations. This allows for a high degree of generality, because the results can be compared to a large body of recent research. The overall goal of the CSoft study has been to contribute to the development of generally applicable theories for agile software product line engineering and to the more recent concept of software ecosystems.

Grounded theory aims towards theory building. This approach to analysis was originally defined by Lewin (1946), later revitalized by Glaser and Strauss (1967), and then developed into a few variants. Regardless of variant, the main principle is to analyze (usually textual) qualitative data step-wise and bottom-up by coding text fragments, adding a meaning to data, thus following an inductive process. Codes, which are short textual descriptors of the text, can then be grouped to reveal higher-level concepts in the data material. These concepts can be further grouped into categories, which can eventually be used to put forward and support a theory explaining the phenomena being studied.

This approach to analyzing data is also referred to as constant comparison, which according to Seaman (Seaman, 1999) is the classical method for theory construction. In the various studies supporting this thesis, textual data were reviewed constantly while they were being collected in order to identify concepts of interest. For example, the concept of cross-organizational participation in planning was formulated and developed in the course of data collection. The increasing knowledge of the case was used to guide the collection of new data, thus making it an iterative process, where understanding of the case is built incrementally. Recent findings were typically used in interviews, for example to elaborate seemingly interesting concepts or to clarify confusions. This continual interplay between the use of information that had already been acquired and the information that was being acquired is an example of the hermeneutic circle as described by Klein and Myers (1999).

Due to the nature of the data that have been collected in the four studies of CSoft, which in sum are a large amount of text, it is not feasible to include all this material in this summary. Thus, we give details on how the analysis in each study have been done and supplement these descriptions with examples.

Study 1 – Adoption of Evo

Transcripts from all eight interviews and the postmortem analysis with a team of developers were analyzed using NVivo™, a software tool for analyzing textual data by tagging data and then grouping analytical fragments into larger constructs. Each tag (called a node in NVivo) explains the *meaning* of the coded text. For example, one of the interviewed customers explained the reason for spending time as an external stakeholder in the Evo project:

Interviewer [following up on previous statements in the interview]: "..do you actually define requirements for the product?"

Respondent: "Yes, and very detailed to. Only yesterday we did two requests about two things that didn't work in the reporting module, which is very important to us."

This and other text fragments was tagged with "Evo enabled participation in development". Another customer were telling about how they provided ideas to the development project:

"We provided development ideas and feedback for CSoft's new Panel Sample Builder. We were asked to provide input on what we require for pulling panel sample efficiently, and how we could help improve CSoft's existing sampling tool. "

This and other text fragments were tagged with "Providing concrete input". These two tags, and others, were later in the analysis grouped together under a higher-order tag (called tree nodes in NVivo) and called "Motivation to participate".

Through several iterations of analysis, done by the two authors of paper 1 and 2 (Fægri and Hanssen, 2007; Hanssen and Fægri, 2006), a structure of tags was developed. This can be seen as an iterative and incremental approach to analyzing qualitative (textual) data and a case of constant comparison. The defined aim of the study was "*to present a real case of agile customer engagement showing prerequisites, benefits, costs and risks in a software product setting"*. This study aim was used as an analytical lens in the study, both for designing the interview guides and for analyzing the data.

To illustrate the outcome of this type of analysis we list the structure of the tags developed for the interviews with the customers (45 nodes ordered under 10 tree-nodes):

Table 8 - Example coding nodes

Premises for Evo ○ Customer participation needs preparations ○ Knowledge of each other is important	Engagement ○ Customer was invited to participate
Communication ○ Customer appreciated dialogue ○ CSoft listened ○ Good communication ○ Frequent communication ○ Used emails ○ Used telephone ○ Wants a close dialogue	Customers perception of the process ○ Good management of expectations ○ Spent a small amount of time ○ Detailed requirements from customer ○ Was not able to test administration and performance issues ○ Satisfied with the participation ○ Satisfied with CSoft's response ○ Informative ○ Little interaction ○ Wants to be involved in future projects ○ Missed focus on some important requirements ○ Want explicit feedback from CSoft ○ Weekly iterations are too short
Customers' practical tasks ○ Found errors, which was corrected ○ Did not use the test server ○ Requested, and got a local installation ○ Tested the software and provided feedback ○ Used the test server ○ Customers' tasks perceived as casual	Customers' background ○ Have an existing relationship ○ Customer is an expert user ○ Have a close relationship with CSoft ○ Considered to be a large and important customer
Educating customers ○ Training (in the Evo process) was simple ○ Training customers (for participation) is important	Motivation to participate ○ Evo enables customer to participate ○ Expressing concrete requirements ○ Customer is highly engaged in the product ○ Need more in return for the time spent
Local organization at the customer ○ Involved other persons from own organization ○ Involved own customers	○ Previous practice ○ Did not see new versions until final release ○ Support is perceived to be impersonal ○ Feedback used to go through customer support only ○ A low level of communication ○ Development was perceived as informal

(Similar structures were developed for the interviews with the five members of the product management team and from the postmortem analysis with the developers.)

Having developed this structure of tags and an overview of the textual data, we used it in the papers to report on what we found to be the most interesting results with respect to the defined study aim. Due to the nature of the data collected (several pages of text), it is not feasible to present data in its raw form. We used the NVivo tool to extract tagged text for nodes we found interesting or even combination of nodes, for example, text fragments that were coded with more than one node.

These insights were used to develop the discussions in the papers. For example, in paper 1 (Hanssen and Fægri, 2006), we have sections discussing *"Stakeholders' practical roles", "Specifying quality goals"*, and others.

Study 2 – Agile SPLE

In study 2 we reused the data and NVivo-analysis from study 1, and added two extra interviews. The outcome of the analysis was presented in paper 3 (Hanssen and Fægri, 2008), discussing findings related to the defined research aim of the study, which was *"to describe and analyze an industrial case to understand how SPLE and ASD can be combined and to clarify associated costs and gains."* Like in paper 1 and 2, we use the analyzed data to discuss relevant topics and findings. This was supported by a set of business plans for a release of the CSoft product line.

Study 3 – Software entropy

The three transcribed interviews were analyzed in NVivo, using the same approach as described under study 1. The research goals of the study guided this analysis: 1) *How may system entropy and agile processes mutually negatively affect each other?* And, 2) *Can code smell analysis and refactoring be a viable solution?*

This analysis was supported by the outcome of the architecture analysis, which documented facts about the internal structure of the CSoft product line. The results were presented in paper 3 which reports and discusses the facts from the architecture analysis and the most important findings from the qualitative analysis of the interview data which were 1) analyzability and comprehensibility of the product line, 2) modifiability and deployability, 3) testability and stability, and 4) organization and process.

Study 4 – Software ecosystems

Eight new interviews were made as well as two observations and collection of relevant documentation. Data was analyzed in two steps:

Step1 – All data were first examined to produce an intermediate analysis report in order to produce an updated overview of the CSoft development process in terms of roles, activities, and artifacts, in addition to high-level concepts, necessary to understand how product planning and development is conducted. This analysis created a structure by grouping information coming from the various data sources. Examples of such concepts are teamwork, planning, customer interaction and innovation.

Step 2 – All data, in textual format, was analyzed using NVivo, using the same approach as explained for the other studies.

The outcome of this analysis are presented in paper 5 (Hanssen, 2010) and 6 (Hanssen, 2010). The latter includes several extracts from the data to illustrate the concepts being discussed.

The case description

The description of the case organization, the product line and their processes in section 2.2, is a summary of all information which have collected in all four studies. The overview in the subsections 2.2.1 to 2.2.3 represents the present organization and situation, while section 2.2.4 explains the development towards the present situation. This information is the result of several interviews and observations, done over several years and is supported and complemented with information from the organizations web site, marketing information and company presentations. Some of these data sources are listed in Table 7. These descriptions of facts about the case have been shared with persons from the case organization, which have verified the correctness. For example, we shared the manuscript of paper 4 with the architecture team.

4 *Results*

This chapter presents a synopsis of the results from the four studies reported in the six papers that support this thesis. As the thesis is solely based on the supporting publications (listed in section 1.5), no additional results or data are added in this chapter.

Each section gives an introduction explaining the aim of each of the studies and papers, and the main results. Further details and information about study approach, related research, analysis, discussions, and conclusions can be found in the original papers in part II of this thesis.

4.1 Study 1 – Adoption of Evo

The first study of CSoft did coincide with the first attempt to try the agile method Evo in 2004, as an alternative to the plan-driven approach that CSoft had used for several years. Due to declining process performance and problems related to the late user feedback on new functionality, the development managers decided to try Evo for one of the modules in the product line. One of the most obvious changes was the change from long-lasting sequential release projects to extremely short development iterations with frequent interactions with external actors. This case represented an unexpected and unique opportunity for the researchers, who followed this process trial. Evo was used for two main releases and two intermediate releases, covering a year and a half of activity. This study resulted in two publications, focusing on important aspects of the adoption of Evo, respectively customer engagement (Hanssen and Fægri, 2006) and process fragility (Fægri and Hanssen, 2007).

In sum, our focus in study 1 was *the initial adoption of the agile method Evo in a waterfall-oriented product line organization, with an emphasis on how developers relate to customers and the fragility of the agile approach.*

4.1.1 Paper 1

(Agile customer engagement - a longitudinal qualitative case study.)

Due to the sudden start of this software process improvement initiative and our study of it, the study aim was simply *to investigate and present an industrial case of agile customer engagement showing prerequisites, benefits, costs, and risks in a software product setting.*

Our analysis of the interviews identifies a number of prerequisites for succeeding with an agile development approach in a software product line setting. Proactive stakeholder management is the foundation. CSoft operates in a dynamic market in which long-term planning in advance of development is unrealistic. This applies equally to the customers, which are inherently unpredictable in their role as collaborative partners. CSoft depends upon mutual benefits in the collaboration to maintain motivation and contribution. Lack of continuity has significant bad effects on the performance. In turn, achieving these mutual benefits depends upon selecting relevant customers with sufficient expertise. These are all volatile and difficult factors. Thus, the capability to constantly review and manage the selection of stakeholders is critical. Furthermore, our analysis shows that Evo is a highly demanding process with respect to personal discipline and professional behavior. Due to the frequent iterations there is little room for unrestrained activity. All roles must be meticulously filled. Additionally, some sophistication in technical infrastructure, such as continuous integration of the solution under development, simple backlogs of requirements (called impact estimation tables (IET) in this case) and ASP-based software delivery, has been essential in CSoft's use of Evo.

Further on, we have seen that CSoft has achieved a number of benefits as a result of Evo and the introduction of the product management team (PMT). First, close customer cooperation has a highly motivating effect on the developers. Second, developers' confidence has increased as a result of continuous settlement of expectations in that stakeholders assist in the prioritization of goals. The direct cooperation with users is a positive experience for the developers as it increases the quality of the communication and leads to an improved understanding of customers' business problems. Third, Evo has increased the visibility of the process internally in the organization and externally among the stakeholders.

Although these benefits, we have seen that the adoption of agile customer engagement practices has incurred additional costs. Our analysis emphasizes the extra overhead in actually running the process and the human resources required. Essentially, each Evo week/iteration is a complete development process, spanning a number of phases that demands frequent changes of context and thus occupy significant resources. Also, the technical infrastructure, being a prerequisite, is costly. Furthermore, the analysis shows a significant cost incurred by the continuous need to maintain it.

Through our analysis we have come to consider increased exposure to risk as a cost. First, short iterations with insufficient attention to management and process compliance increase the fragility of the software process and create a risk of leaving developers with a high workload. Secondly, engaging in this kind of strong cooperation with a small selection of customers also means a reduced capability to capture the needs of other, non-appointed customers.

4.1.2 Paper 2

(Collaboration and process fragility in evolutionarily product development.)

Following the results from paper 1, using the same data material, we investigated and discussed in more detail the transition from a waterfall-oriented approach to an agile approach and in particular the collaboration with external stakeholders and the related fragility in the new process.

Our study of the introduction of Evo at CSoft showed that the agile process made the relationships with the customers more collaboratively and that a constant focus on delivering customer value increases the synergetic value of these relationships. This becomes a significant competitive advantage, enabling more efficient innovation transfer between CSoft and the collaborating customers. However, we also clearly saw that maintaining a watchful, vigorous stakeholder management capability, as implemented by the product management team, is paramount for Evo's success. Further more, Evo increased the product's exposure and resource usage compared to the original process. More people can observe or criticize prioritizations. This can help maintain focus on generating value, but it also influences the nature of the demands on the product. This follows implicitly from the increase in direct stakeholder feedback. However, the R&D department also saw a lack of attention to software engineering principles, therefore, companies should also select stakeholders with a particular interest in the product as an engineering artifact (called "internal stakeholders" in Evo) at intervals to counter architectural erosion and subsequent excessive costs of implementing improved quality. We also learned that short cycles of specification, design, development, and testing reduce tolerance toward inefficient tools, for example tools for automated testing and continuous integration. On the other hand, we believe that these rapid, direct feedback loops explain most of the increase in employee motivation and enthusiasm that we saw, particularly in R&D. Rapid feedback on developed code, appropriate metrics, and effective decision making help reduce uncertainty and thus increase developer motivation. The positive experiences reported by stakeholders stemmed mainly from a feeling of being listened to and able to affect the development.

Evo is an agile process but also a sophisticated one, highly dependent on diligent fulfillment of the specified roles and activities. This sophistication is also Evo's Achilles' heel. We have found that Evo, as practiced in the case we have studied, is very vulnerable to irregularities. This process fragility increases with the frequency of iterations, which put higher demands on the timeliness of role fulfillment and activities. Unresponsive stakeholders expose this fragility.

4.2 Study 2 – Combining agile software development and software product line engineering

Based on the results from study 1 and an initial understanding of some of the effects and challenges we observed, we continued our study of CSoft and their continuous adoption of Evo. Now in a larger scale, covering more parts of the.

The focus of study 2 was *the combination of ASD and SPLE and the coordination between the strategic, tactical, and operational processes*.

4.2.1 Paper 3

(Process fusion - an industrial case study on agile software product line engineering.)

Building on the results reported in paper 1 and 2 we collected more data to investigate the broader consequences of combining an ASD process with a SPLE practice. We were interested in learning how this combination affected various levels of the organization, not only the development department. The aim of the study reported in this paper was *to describe and analyze an industrial case to understand how SPLE and ASD can be combined and to clarify associated costs and gains*.

At a high level, we have found that the integrated software processes at CSoft support three key virtues of product development. (1) Technical excellence: an open and modular platform architecture implemented using industry-standard technology enables simple development and maintenance of the product line. (2) Market knowledge and relevance: the well organized, yet nimble strategic process provides adequate decision support for company management and guidance for the development projects. (3) Agility: the adoption of Evo, and agile principles such as short and frequent iterations with active participation by external stakeholders, enables fast response to changes in stakeholder requirements and accurate delivery of desired features and qualities to the users.

An important observation from our findings is that CSoft's integrated development process is not just an accidental collection of strategic, tactical and operational processes. Rather, these three processes play distinct, supplemental and important roles at three different levels in CSoft's overall software development business, all three being lightweight to reduce unnecessary work. We also saw that these three processes together have the same function and effects as the more general Shewhart/Deming improvement cycle (Deming, 2000) (See Figure 7). This is also known as the PDCA cycle, from its four main activities: Plan (plan how to satisfy improvement requirements); Do (accomplish planned actions); Check (monitor the actions and verify the outcome with respect to the planned effects); and Act (implement actions for improvement based on the acquired information).

Although a feasible combination, this approach has its costs; running the strategic process and working closely with stakeholders entails a considerable extra overhead, resources that could have been invested in development (as was the previous practice). Arguably, the most important benefit of this process configuration is that it helps CSoft to exploit both strategic long-term ambitions for innovation and smaller-scale tactical innovations, such as refinement of the software to meet more detailed end-user requirements.

Further, the process configuration is a potent foundation for further process innovations. As long as the PDCA cycle is maintained, any component in the configuration can be further refined to improve performance. If the company experiences the expected growth, the modularization gives extra organizational flexibility. For example, introduction of separate roles for platform and product development within R&D can readily be supported because the objectives of strategic and tactical development are already defined and well established. Additionally, the explicit engagement of primarily external stakeholders in the tactical development process ensures that the company is able to supplement strategic planning by selecting stakeholders based on long-term as well as short-term interest.

Engaged stakeholders provide requirements and feedback on a detailed level but it is the responsibility of the development projects to suggest practical solutions that will meet the quality goals stated by the stakeholders. This close cooperation with a few selected stakeholders promotes valuable creativity. If process innovation at CSoft had a more revolutionary style, replacing all existing practices with a pure and very formal SPLE approach, this potential could have been lost.

4.3 Study 3 – Process agility and software entropy

Through the initial studies of CSoft and their adoption of Evo we noticed a growing concern regarding problems related to the increasing entropy in the code base. Actors being close to the code, like developers and in particular the dedicated software architecture team, were worried about the increasing level of entropy in the system as an invidious result of the agile software product line engineering practice which by now had been adopted in the whole organization. Based on this, we initiated a study to understand better the relationship between the agile development process and the entropy of the system.

The focus of study 3 was *the emerging problem of software entropy, the potential reinforcing effect from the agile development process, and potential solutions to improve the situation.*

4.3.1 Paper 4

(Software entropy in agile product evolution.)

Based on the increased understanding of the agile software product line engineering at CSoft we sought to investigate a concern of software entropy, which have emerged from the previous studies. We set out to answer to research questions: First, *do system entropy and agile processes negatively affect each other?* And secondly, *can "code-smell" analysis and refactoring be a viable solution?*

First of all, we constructed an overview of the technical structure of the CSoft product line, needed to understand problems of system entropy. The system has been under constant development since 1996 and is based on several technologies that have emerged over those years. Aging solutions from years ago are still part of the system, such as older ASP solutions, COM+ components, VB6 code and other legacy technologies. Today, most new code is developed in C#, and is spread over approximately 160 .Net assemblies. The complete product is best described as a traditional three-tier system with an MS SQL Server driving the data layer, a business layer and a presentation layer based on a dozen ASP.Net applications. There is a clean separation between the presentation- and the business layer. However the most obvious problem in the software is what the architects refer to as "the Blob": a very large assembly (aptly named Core) consisting of approximately 150K lines of code in 144 namespaces. Section 1.1.1 contains more details.

Through interviews with the architecture team and actors from one of the development teams we uncovered four types of problems related to the level of entropy in the system.

(a) Analyzability and comprehensibility. Due to the high complexity of the system, it is very hard for developers to get an overview of the code and its structure. Especially the central component has grown extremely large and has many internal references (each namespace depends directly or indirectly on another namespace), making it difficult to understand how it really works. This was clearly not by design, but the result of years of intense development. The system is structured as vertical modules, but as it is now there are too many relationships between the verticals – changing one will inevitable affect many others. New developers joining R&D have a steep learning curve and require close follow-up over a long period of time by more experienced developers. There exists no documentation or models that explain the structure of the system, even though this clearly would be highly useful both to existing and new developers. Even worse, having problems understanding how the code is structured leads to a fear of changing the code, both for adding new features and for improving existing code. The unclear internal structure creates a cognitive overload and a common (unfortunate) way to deal with this is code duplication: instead of modifying existing code, developers

create their own copy over which they have full control. This leads to a larger cognitive overload for other developers; only making the problem worse – it has become a self-reinforcing effect.

(b) Modifiability and deployability. As a result of the duplication and entanglement of code, developers frequently need to perform so-called 'shotgun surgery', meaning that even the modification of a small detail forces them to identify and change code in many places. These problems slow down the development process and the potential for errors increases due to the high chance of overlooking one or more locations. Having to deal with bad code is frustrating to the developers as they in some ways in practical terms are enforced to build bad code on bad code as there is no room to actually resolve the problem. Besides development and maintenance, also deployment of the product suffers from its structure: The current core component aggregates features and functionality for every possible configuration of the product and it has to be released as a whole, even though only a fraction of the functionality may actually be needed for a particular configuration.

(c) Testability and stability. Due to the size of the code and the many cross-references, there are too many paths through the code to test them all systematically. The test coverage is not high enough and existing tests have shown to be unstable and inconsistent. For example, the same tests run on similar systems may produce different outcomes that are hard to explain. Also, a lot of the existing tests are extremely large, meaning that they too are hard to maintain and use. When a test fail, it often takes a lot of time to locate and fix the actual problem that triggered the failure. Although such tests are supposed to act as a safety net and give developers the courage to make changes they are not trusted. This increases the fear or at least reluctance to change existing code – since the effects of a change are hard to foresee and errors can have considerable negative effects. Nevertheless, regression testing is done, albeit with a lower than desired quality.

(d) Organization and process. As both the business domain and the system are highly complex, each of the development teams (4-6 developers in each) has an expert (the so-called guru). This guru has high technical skills and extensive experience with the code, which is vital for the team to solve its tasks. Consequently, this organization represents a considerable vulnerability; losing just a few of these gurus would have devastating effects on the development. The development process is based on two-week iterations and it is a strong focus on delivering working software by the end of each iteration. A negative effect of this focus is that delivering quality software is at times traded in for creating a working version. Each iteration ends with a review, but the high velocity typically does not give enough time to catch all issues. This causes extra work close to a release when the system is thoroughly tested as a whole, yet entropy is allowed to grow from release to release. The development teams are set up to have

separate areas of concern, each team being responsible for a part of the total product, e.g., the reporting solution or the data storage. The idea is to build competence around a well-defined part. Unfortunately, the structure of the system does not reflect this organization in practice, because functionality is spread throughout the code. This forces the teams to operate outside their area of concern, which has shown to negatively affect their ability to produce enough new and improved features of the product in their releases. The total request for improvements from the market is constantly higher than what actually is delivered, thus indicating a need to improve development efficiency.

As part of the discussions with the architects, we also collected several of their high-level ideas to further improve the product and development process:

(a) Process automation. Currently too much testing is done manually and more automation is desired. In addition, to establish an efficient and trustworthy safety net for the developers, tests need to become more stable and trustworthy. With this in place, the architects can introduce what they call "pain-driven development". That is, when a developer introduces or changes code that breaks the tests, he or she will get notified immediately to correct it.

(b) Restructuring and refactoring goals. The architects feel that components of the software need to be de-coupled from the core and the overlapping and duplicated code has to be removed. They also agreed that the system should have a clearer separation of concerns were vertical modularization should reflect business segments and horizontally, the system should better separate business and platform related code.

(c) Continuous monitoring of quality. The architects proposed a principle that they refer to as "quality-from-now", meaning that any change to the code should be analyzed at development time, to check that it does not conflict with defined rules of good design. This can, for example, be achieved using tools to detect code smells and monitor potential problems nearly constantly during development. The architects believe that this approach would considerably reduce the fear of changing the code.

4.4 Study 4 – Software ecosystem emergence

By now, having followed CSoft over five years - observing how they have changed their development process - it became evident that this change also affected how they relate to external actors, beyond single customers acting as stakeholders. This initiated a new iteration of data collection and analysis with the primary objective of understanding how the organization relates to its external organizational environment. These new data revealed an image of an organization that was opening up their strategy and development processes, moving towards a situation that can be understood as a software ecosystem in development.

The focus of this final study of CSoft was *the increasing openness of information sharing and cross-organizational collaboration.*

4.4.1 Paper 5

(Opening up product line engineering.)

Based on the lessons learned from the study of CSoft so far, this paper looks into the change that the product line organization is experiencing. The aim of the paper was *(i) to explain how the SPLE process can be opened up and why this may be a necessary development, (ii) to illustrate some practical issues, and (iii) to provide some preliminary guidelines to industry and indicators for future research.*

Actions taken: The initial waterfall-like approach with extensive up-front planning required a considerable amount of resources and blocked input from customers and other important stakeholders during the development phase. Through the adoption of core agile development principles, fewer resources are spent on making detailed up-front plans and controlling plan conformance during development. The iterative process opens for continuous corrections during development based on a close dialogue with stakeholders, including invited customers. This enables a reactive approach to SPLE where no assets are changed or added to the line unless they are explicitly founded on concrete needs.
Observed effects: The resulting SPLE process at CSoft helps addressing some of the inherent risks that comes with this approach. McGregor (2008) describes that, for a pro-active approach, there is a risk of developing assets that will become obsolete and not used in later applications. For a reactive approach, there is a risk of missing short-termed business opportunities due to increased time-to-market and that it might require considerable effort to, reactively, prepare assets for later reuse. CSoft's incremental and iterative approach meets these risks as a compromise between the proactive and reactive approaches. Continuous corrections by involved stakeholders ensure that changes and additions to the product line actually will be used. The process also opens for corrections close to the release, if found necessary. In addition, the proximity of representatives from the business domain and the extensive use of direct and conversational exchange of information improve the ability to catch both explicit and implicit (tacit) requirements (Grunbacher and Briggs, 2001). One potential effect on the negative side is a reduced maintainability of the product line. The agile development process emphasizing a continuous focus on short termed goals seems to reduce the focus on maintaining the code and architecture properly, thus causing escalating system entropy which seriously hampers the organizations ability to improve and develop the existing code base.

Enabling factors: The opening of the SPLE process at CSoft has been supported by some additional measures. One of the most noteworthy is that a community of third party organizations has emerged, partly supported by CSoft and partly as an effect of the accessibility to the product line via the API's. Approximately 60 organizations are now offering software products and/or services partly or completely based on the CSoft product line. This means that CSoft may keep full focus on developing the core product line and that customers are offered a large number of specializations and applications of the product line. The emergence and growth of this community come as a result of not only the technical accessibility via the API's but also the sharing and external visibility of product plans and roadmaps. Key clients are given insight into these plans; some are even visiting the R&D department to elaborate business needs and ideas, and to discuss potential solutions directly with the development teams. CSoft has also (once so far) organized a large product conference where strategies were announced, new features introduced and third parties got to expose their solutions.
Contextual factors: This change in CSoft's approach to developing their product line is moderated by a set of contextual factors. First of all, the changes done to the development process is a reaction to the volatility of the business domain being addressed. This particular domain is still shaping, its boundaries still being determined. This is an alternating process; the technology being developed and offered through the product line is creating new business opportunities, which in return leads to new requirements. In this context, it is more important to be able to respond quickly to the market than to produce reusable components that might be useful at a later moment. Enabling an agile and responsive development organization gives it control over how the domain and its supportive technology develops – a clear advantage in a competitive business. Another factor affecting the shaping of the SPLE process is the strongly emerging software-as-a-service (SAAS) delivery model. Software is not procured as a commodity, installed and managed locally, but rather as a service accessed over the Internet. This simplifies the delivery of software and consequently also the development process. For example, this model makes it practical to engage selected stakeholders, which frequently and with a minimum of effort can access the latest increments to assess and provide corrective feedback.

4.4.2 Paper 6

(A longitudinal case study of an emerging software ecosystem: implications for practice and theory.)

Based on the previous studies of CSoft we collected more data, now from a more mature organization than for the first studies, to develop a more thorough understanding of the development towards a situation, which can be named as a software ecosystem.

Our research question was: *Why and how is software product line engineering developing towards a software ecosystem?*

The results from this study relates to CSoft's engagement of customers and the emerging third party community constituting a software ecosystem.

The main motivation for customers to spend time participating in the development projects is the ability to affect development: no payment or any other compensation is provided. One of the product managers explained:

> *"...they see their wishes or their requirements or whatever in the product at the end. And then you get very nice feedback like 'I can see that I said this and that, and in the next release you did it'".*

This collaboration forms a self-regulating system where the supplier and the stakeholders mutually adapt to each other through their shared interest in developing the software product line. This usually works well, but there is always a risk of having external stakeholders, which do not provide the necessary input, as explained by the manager of the PSG team:

> *"Everybody has busy jobs and projects that need to be on time etc. It happens quite often that we have to cancel these meetings or that they haven't done anything since the last time. Then we can only show them what we've done and get some ad-hoc feedback..."*

The PSG manager also explained that it is relatively easy to discuss ideas, but that it is more of a challenge when they are included in the development process:

> *"There's no problem to get them to discuss high-level plans, but it varies when it comes to the development process"*

Maintaining the motivation for participating is an important task for the PSG.

Interestingly, large and leading customers tend to expect and demand to be more and more involved at both planning and development stages. One example is a product conference keynote given by the VP from one of the large customers. He stated several 'requirements' (this was the word he used) for being involved, for example:

> *"Regular meetings with product development teams", "To work as a stakeholder on new software developments that are key to us.", "Help to guide product strategy."*, and others.

Finding the "right" stakeholders for participation in the development projects is not done through a formal and structured process, but is mostly based on the collective knowledge about the customers. The PSG manager says,

> *"We don't have a formal process for selecting stakeholders. We have internal discussions, listening to sales people etc. We know which customers have asked for certain features or improvements or those that are heavy users of a particular type of functionality."*

In addition, experience from previous participation is also useful as explained by the R&D manager:

"We have become better at selecting [external stakeholders]. Those that have disappointed us are never asked again. You end up with a pool of persons that you know you can trust."

An important part of recruiting customers as external stakeholders is to communicate to them clearly the opportunity, which is given to them. The PSG manager explains:

"..quite often we talk about our development process. We do it in sales situations because it tells that our goal is to solve business problems for our customers. This level of interaction and the way we try to listen has been well received, and now some customers insist on being involved in development."

In the very start when Evo and collaboration with external stakeholders were at an experimental stage it took quite some effort to recruit stakeholders to the development projects and to keep them active (Fægri and Hanssen, 2007). After some releases where collaboration with external stakeholders have become an integrated part of the development process the situation is turned upside-down. When asked to explain this relationship, one of the product managers told us:

"It's almost a problem because as soon as you offer the capability of being a stakeholder, the hardest part is rejecting people, turning them away from actively participate. So people are very keen on participating."

Co-creating the product line is one of the most significant effects of engaging and communicating with external stakeholders. The rationale is simple, CSoft have the most up to date knowledge of the technology, and the ability to make use of it in the development. Likewise, customers hold the most up to date knowledge of their own business domain, and how it seems to develop. These two pools of knowledge and competence are joined in several ways. One important arena for sharing and gaining knowledge is the product conference where management, strategists, developers and other internal actors get to meet externals from various customers and third parties. Equally important – customers can meet other customers, third parties can meet customers or internals etc. This shows the networked character of the ecosystem that is shaped around the product line. Some examples from the product conference in 2008:

A former customer of a competing solution was seeking experienced customers to discuss the product line and share experience. Several providers of third party products and services were having stands at the conference, communicating with both existing and potential customers and developers from CSoft.

Another major event, which is more directly focused on the development of the product line, is the annual Advisory Board meeting. Top management from some of the largest and most demanding clients meet with the PSG and other actors who are involved in the shaping of the product strategy. A PSG member explains that they meet to:

"...discuss high level product strategy and how the demands of their companies and the market are developing." Bringing together major competitors like this was a daring thing to do according to the PSG manager: "The first time we did this it was a

bit exciting – would they discuss issues openly, and would they open up? It turned out that they did very fast. They have many concurrent needs, and even though they are competitors they see the value of doing this."

From a practical viewpoint, we see that tools and infrastructure for collaboration are important enablers for co-creating the product line. Especially the Webex™[6] online meeting solution lowers the threshold for having frequent and detailed meetings with stakeholders:

From our observation of one of the customer review meetings we saw a lot of very detailed discussions that were made possible by on-the-fly demonstration of the software through the screen sharing solution. This sparked detailed discussions both on the customer side and among the development team. The meeting resulted in a list of clear actions points to be addressed in the next development iteration.

Close corrective feedback in the Evo development projects is another approach to co-creating the product line, but on a tactical level. One of the developers describes the meeting with the external stakeholder at the end of the two-week Evo iteration:

"What you get during a meeting is often very valuable. Especially when you are about to move in the wrong direction, which you can adjust. We get feedback saying that our solution is not quite what they had in mind or what they need."

This demonstrates one important function of the agile process; the development teams get nearly immediate (within two weeks) and detailed (face-to-face) feedback. This closeness to a few selected customers means that CSoft must also consider the needs of other customers, as they are the referent organization, which always has the last word in the development of the product line. This is partly achieved through Evo's focus on product qualities instead of product features, which are typically emphasized in plan-driven development methods. This is a useful abstraction, and it turns the focus from predefined design (features) to effect and impact (qualities). Both the product roadmaps and the evaluation meetings at the end of the Evo iterations evaluate the product qualities. This means that both the development teams and the external stakeholders have to consider *why* something is needed, leaving the *how* to the developers. The PSG manager explains:

"..we take one step back, and try to think about why our stakeholder needs this, and then rethink other ways of solving their problem. It is in this type of process that the smart things can turn up – that your thinking is totally new and that you come up with a solution which may be a totally different way of doing it, maybe faster...".

Catching and following up on customer ideas on an ad-hoc basis is equally important as involving customers in regular processes such as roadmapping and the Evo development projects. At the product conference:

A customer representative told about a case where his company gave input to CSoft on some changes they would have liked to see. This led CSoft to invite a delegation

[6] www.webex.com

from the (abroad) customer to the R&D department in Oslo. Ten CSoft people spent the whole day discussing the solution with four representatives of the customer. This was perceived very positively, and in the end actually affected the software.

Some of the largest customers may also request dedicated workshops to discuss needs and ideas. The PSG manager talks about this:

"...for some of our largest customers, mostly by their initiative, we organize workshops once a year, usually on a strategic level. They want to know what the roadmaps may bring for the next couple of years, and talk a lot about their needs etc.".

The close contact with customers is also a valuable source of learning about competing solutions. One of the team leaders talks about customers visiting the R&D department:

"In these meetings they demonstrated the solution they used today, and actually demonstrated how they used the competing solutions – what worked well and what needed improvements, as well as ideas they might have. These meetings gave the team a wealth of details, and it was quite clear what to deliver to the stakeholder."

The PSG manager also explains the value of learning of the <u>use of</u> competing solutions:

"...alternatively they do it using other tools today when not using our solution. The option to work more closely with them and to get that knowledge made us more capable to meet their needs better than before when the development was more of the black-box type".

A phrase from one of the roadmaps illustrates the business impact this may have:

"Through a client we have been given a thorough demonstration of the competing solution NN, and by implementing support for [some advanced functionality] and a couple of small features, the X-module will by far exceed their corresponding functionality. These improvements alone will ensure we win one [sales] deal, and have also been brought up by several other clients/prospects."

Learning the business processes and domain is another valuable outcome from the direct contact with selected stakeholders. One of the developers talks about one of the stakeholder meetings in an Evo project:

"...we have tried to solve a task in a way we believed would be reasonable, but to people who actually use this it is obvious that we have misunderstood the process. This gives us guidance as early as possible."

A PSG member tells about another case:

"They [the customer] were here in a workshop for two days. We presented the roadmap [for module X] and they presented their wishes and their business, [related to] what they are doing."

This illustrates the shared interest that the customers and the supplier have – customers want to learn about the product line and its development. Correspondingly, CSoft learns about the business that their product line is supporting.

At present around 60 external organizations base their business completely or partly on CSoft as a platform. This can be value-adding solutions or products, related services, and consulting. Examples are solutions for data visualization or voice data capture technology, assistance in using various components in the product line, and training. This networked community (Fricker, 2009) has not been planned and deliberately established by CSoft, it has emerged spontaneously over the past years. This emergence is mostly driven by customers' need for additional features and services on one hand, and the opportunity to extend and use the product line as a platform on the other hand. Also, building solutions and providing services based on the product line means that external organizations get immediate access to a large group of established users of the product line.

Providers of third-party solutions are considered to be important external stakeholders, and are included in the development of the product line in very much the same way as customers.

> *During a product conference, a representative from a third party company, delivering an integrated product, explained that when they needed to improve the integration with the CSoft platform they took on the role of an Evo stakeholder. Communication was mostly done by phone, supported by web meetings with screen sharing.*

Offering an efficient integration technology enables a third party community. Over the past few years a set of simple APIs have been offered to enable external actors to make extensions to the product line. The development of these APIs have followed the development of the product line, where each new release has improved existing and offered new APIs due to requests from external actors. This means that there is a long (a year) connection time between a request for an interface and its actual release. As more and more externals have made use of this connection point to the software it has been given increasingly higher priority in the development of the product line. An excerpt from one of the roadmaps exemplifies this:

> *"We are in dialogue with some clients/prospects who are building their portal in a Content Management System, and need to integrate content from [module X] into it. Some competitors seem to have APIs that are easier to use than our SOAP[7] based APIs, making it easier to integrate with other portals/communities. It is therefore an ambition to provide an easier API for including [module X] content into an external portal."*

Due to the extensive use of the API's by externals and their increasing demand for integration with the product line it became clear that the simple web-service based interface had become obsolete. This has led CSoft to develop and offer a new API called FlexibilityFramework[8] (FF), which enables a closer integration to core services in the product line than the previous (and still existing) simple messaging-based APIs

[7] Simple Object Access Protocol, http://www.w3.org/TR/soap12-part1/

[8] A fictitious name to protect the anonymity of the case.

offer. A recent webcast, where the CSO presents FF explains further the motivation for this improved interface:

> *"CSoft is like a supertanker. It is large, can take huge loads, travel far, and take heavy weather. These are all very positive things; on the other hand, the consequence of that approach is that we are quite careful at looking after the supertanker. That means various procedures, on policy, on quality assurance and so forth. And that means that we get less nimble than we would like. The question we posed ourselves is how can we behave like a speedboat while having all the benefits of the supertanker? I'd like you to think of FF as the speedboat. The tanker is still there. It will still take heavy loads and perform extremely well, but in order to be nimble we can build a few speedboats. And they have independent lives from the supertanker and can run on different development schedules."*

The last argument is worth a comment; with this new interface to the product line external actors are disconnected from the long release cycles of the product line, and can develop value-adding solutions independently. This is likely to further drive the growth of the third party community.

Actively supporting the community has become a regular activity in addition to the continuous development of the product line. As this community has emerged and grown, CSoft have seen its value, and started to actively support it. In 2007 a dedicated web-portal was launched to make this community visible and each partner is listed and presented. There are five types of partners, those offering technology that is integrated with the product line, those offering value adding services, some can prepare the use of the product line, some can use it on behalf of clients and some offer consultancy services.

5 *Discussion and implications*

In the discussion of the results of the studies of CSoft we will emphasize 1) how the combination of SPLE and ASD created a more open development organization and 2) how this change leads towards the formation of a software ecosystem. Based on this discussion we state implications for theory and practice.

5.1 An agile software product line engineering process

5.1.1 Actions taken to combine the processes

By comparing the processes and organization at CSoft with what can be referred to as a traditional SPLE approach we see that CSoft has taken two major steps to shape their development process: (i) the overall simplification of processes and organization, and (ii) the close engagement of customers, both in planning and in development.

Simplification of process and organization: CSoft has simplified their development process in the following three ways.

1) There is no clear separation between core asset development and product development, which is an important distinction in traditional SPLE. The idea of having two distinct processes is to enable the separation of the different concerns of building a robust platform on the one hand, and of effectively building products based on that platform on the other. In the CSoft case, there is no dedicated sub-process for core asset development, yet the total organization manages to handle both the development of core assets and applications simultaneously (Sanchez and Mahoney, 1996). Variability is implemented through a simple system where features are activated or deactivated according to the needs of customers. This simplification with respect to product development and variability management means that there are fewer roles and fewer dedicated sub-activities. Although simpler, it also means that the product line has less variability. This drawback is balanced because it is easier to manage the CSoft product line with the development resources focused on the next release, not on managing an extensive set of variations, which can be costly (Bosch, Florijn et al., 2001).

2) The second simplification is how requirements are managed. Related to the first type of simplification discussed above, CSoft does not implement a dual requirements engineering processes, one for core assets and one for application development as explained by traditional SPLE approaches such as the SEI guidelines (Clements and Northrop, 2002) (p.109). The equivalent is the strategic and tactical processes as described in section 2.2.3, but with the difference that the roadmaps represent high-level guidelines for the long-term development of the product line and not requirements for

developing core assets in the product line. At CSoft, requirements engineering is done as part of every iteration where external stakeholders define and re-define requirements based on experience from the previous iterations – in line with principles of ASD (Takeuchi and Nonaka, 1986). Furthermore, CSoft's agile SPLE approach means that the product managers in the PSG and the development teams does make use of practices such as domain analysis techniques, stakeholder-view modeling, feature modeling etc. (described e.g. by the SEI guidelines, (Clements and Northrop, 2002) (p. 114)).

3) The third type of simplification concerns internal communication and coordination. The general principle of openness, visibility, and direct communication is adopted from ASD (Cockburn, 2002). The status of development of roadmaps and the status of the ongoing software development is made visible to everyone internally, and to some extent also to external stakeholders. The primary artifact that communication is based on is the most recent version of the working software, rather than a set of models or documentation of the product line. In addition, direct face-to-face communication is emphasized. Written communication is kept to a minimum. This strategy improves the efficiency of communication (Daft, Lengel et al., 1987). Teams usually work in open offices to enhance communication. The product managers from the PSG are present at most meetings. When someone cannot be physically present at a meeting (typically external stakeholders abroad), the development teams use an online conferencing system with voice communication and screen sharing for live demonstrations of the software. Feedback from involved stakeholders is provided primarily as direct oral feedback, not as written communiqués, thereby improving efficiency (Bosch, 2001).

Engagement of customers: Another major step taken to enable the agile software product line engineering process is the engagement of customers in both planning and development of the product line. Interacting with customers is an obvious but nevertheless critical factor for success in any type of software engineering (Reel, 1999), software product line engineering included. From a purely business perspective, collaboration with actors outside the organization is considered to be important for creating and sustaining competitive advantage (Sawhney, Verona et al., 2005). The objective of interacting with the customer is to define short and long term requirements, to gain access to domain and business knowledge, to verify and to validate results, and even in some cases to involve the customers in the innovation process (von Hippel, 1996). According to Damodarran (1996), collaborating wisely may yield several benefits for software development, such as improved system quality due to more accurate user requirements, the avoidance of costly system features that the user does not want or cannot use, improved levels of end-user acceptance of the system, and better understanding of the system by the user resulting in more effective use.

In contract-regulated development, it is obvious who the customer is and what the conditions for collaboration are. In SPLE, this is not evident at all due to the large number of customers. At CSoft, the multiple links between internal and external stakeholders play an important role. Members of the PSG interact frequently with existing customers, prospective customers, and third parties such as producers of related products and services. Others work in technical account management, support services, and consulting services, and assist customers in using the products. In addition, there is the annual Advisory Board meeting, several dedicated meetings with specific customers, and product conferences – events enabling customer communication. All in all, this means that CSoft has wide and rich communication with direct and indirect users of their product line. In some cases, the relationship with given customers is very informal.

Acquiring information from the customer base and other important actors, and maintaining relationships, takes a lot of resources. This however pays off because it has become a valuable source of knowledge for the company. Communication is mainly direct and verbal (usually over the telephone), not on formal documentation (indirect). This is what Keil and Carmel (1995) define as direct customer-developer links. They show that direct communication between the supplier and a participating customer reduces problems caused by filtering and distortion of information. The more links that are used, the better the communication becomes, up to a certain point. In short, the value of this direct communication is twofold: (i) it is simpler, and (ii) it improves the quality of the communication. The claimed value of direct communication is supported by the media richness theory (MRT) (Daft, Lengel et al., 1987), which ranks several media according to "richness" in communication (face-to-face being the richest, written documents the poorest).

Although valuable, we have also seen that customer relationships might be fragile, and there is a clear need to manage the multiple inputs received from, and the expectations of, numerous external actors. This concern, that the customer role can be very demanding have been pointed out in recent research on the customer role in ASD. For example, Martin (2009) did a series of case studies of XP-projects and found that participating customers were experiencing fatigue over time, threatening sustainability of the customer role.

One important aspect of a customer's motivation to invest time and resources is the ability to affect the course of development. However, input, requirements, and ideas may sometimes conflict with other concerns of the product line, which means that the PSG need to consider the input carefully, and that the final decision on which changes should be implemented in the product line must be an internal matter. In some cases, this may be difficult and there is an evident risk of losing the engagement of a customer if the response to his input is considered too weak or insignificant. As an example, when we studied the initial adoption of Evo at CSoft, we observed that one customer (stakeholder) withdrew from its allocated project because the goals had been reached successfully (Fægri and Hanssen, 2007) (p. 102). However, this is an inevitable price that the development organization has to pay. Users' involvement in the development process may vary from informative, through consultative, to participative (Damodarran (Damodaran, 1996)). CSoft users may be considered to be informative for planning and consultative for development, but never truly participative with respect to decision-making. One risk of not giving customers full participative powers is that the development may "…fail to reflect real human and organizational needs" (*ibid.*).

The development iterations are short and the increments are small and focused, thus driving what may be called incremental innovations as described by e.g. Lettl (2007). In the long term, focusing on incremental innovations at the cost of radical innovations can be a serious threat to the product line and consequently the business. Böckle (2005) refers to this problem as *innovation lock-in*, and points out that this can be a barrier in software product line engineering, because components and variability are predefined. This problem is recognized within CSoft, but no changes are planned as yet to address it. It is likely that if they were given the opportunity, CSoft customers could play an active role in radical innovations. Lettl's study (2007) gives examples of capable and knowledgeable end-users that act as technology inventors. However, doing so would probably require an extension (and complication) of the present development process.

When discussing ASD and customer engagement we also have to address one of the most fundamental practices in ASD, which is the 'customer on-site' practice, an important part of XP (Beck and Andres, 2004). According to a systematic literature review by Dybå and Dingsøyr (2008), this is also one of the most researched practices within ASD. Examples are (Hanssen and Fægri, 2006; Hansson, Dittrich et al., 2004; Korkala, Abrahamsson et al., 2006; Korkala, Pikkarainen et al., 2009; Koskela and Abrahamsson, 2004; Martin, Biddle et al., 2004; Martin, 2009). With respect to the study of CSoft it is important to notice that the principle of customer *on-site* is relevant to smaller and simpler development projects with a single customer or a small group of customers, which is the initial home ground of most agile methods, at least XP. However, in cases where a software organization serves a large market with a high number of customers (like CSoft) this approach to customer interaction is not feasible due to the high number of customers, their diversity and their geographically distribution. Still, applying ASD in such a context needs to find ways to collaborate closely with these customers. Scaling up ASD has emerged as a sub-field of its own (Eckstein, 2004; Larman and Vodde, 2008; Leffingwell, 2007), but the understanding on how to establish efficient customer interaction in large-scale development is still only nascent. We believe that the study of CSoft may contribute to some deeper understanding of customer engagement in large-scale product line development.

5.1.2 Effects of the combined process

Risk reduction: Developing and improving a software product line may carry with it a set of risks that need to be managed. McGregor (2008) describes a set of risks that are inherent in SPLE, depending on whether a proactive or reactive strategy is applied. With a proactive strategy, where assets are predeveloped with the assumption that they will be used in future applications, there is a risk that these assets may become obsolete, thereby becoming a lost investment of resources. With a reactive strategy where assets are harvested from applications for later reuse, there is a risk that short-term business opportunities will be missed because the production of applications is not as fast as it could have been. There is also a risk that a lot of rework will be needed to prepare such assets for future and more generalized reuse. CSoft's approach for managing their product line is best defined as incremental. Further it constitutes a compromise between the proactive and reactive approaches, which reduces the risks mentioned above. Given that all development is aimed towards the next release a year ahead with external stakeholders involved in each increment, there is a low risk of developing features that will not be used. This is actually a central motivation for using the Evo process (Gilb, 2005), which is driven by short-term goals. Another observed effect of the agile approach to requirements management is that frequency of interaction and closeness to customers increase the ability to capture both explicit and implicit (tacit) requirements (Grunbacher and Briggs, 2001). An example of this, from an iteration review meeting with one of the development teams and a customer team, is when the product manager demonstrates a new feature and says *"we're al anxious to see what happens"*. The customer breaks in *"this is nice, because in the current version you have to do this manually"*. This initiated more detailed discussions between the team leader and the customer.

Organizational development: Over a period of 13 years, CSoft has gone through roughly three phases of organizational development, as described in more detail in section 2.2.4. The introduction of ASD (Evo) has been a vital factor for re-establishing efficiency and the ability to respond to the market (Fægri and Hanssen, 2007). This adds to the findings of a study of key business factors in SPLE by Ahmed and Capretz (2006), who conclude that an SPLE organization needs to understand that the customers' business process supports the product line, which in turn supports the business process. This mutual support is one of the important benefits of interacting closely with customers. The third-party community of related companies that base their business on the CSoft product line contributes greatly to the development of the organization because this symbiotic relationship helps CSoft to maintain a strict focus on the continuous development of the product line, leaving specialized ways of use and extensions to others.

Reduced maintainability: The present situation at CSoft has emerged over several releases as the organization made it a priority to serve the market rapidly with new and improved features and functional qualities at the expense of internal "tidiness". This may have been a wise strategy to establish a position as the present market leader, however, this more or less deliberate strategy for the shaping of the organization and the product line has come at a cost of escalating system entropy, which makes it more and more difficult to improve the product line. Through a recent study of the maintainability of CSoft product line architecture (Hanssen, Yamashita et al., 2010; Hanssen, Yamshita et al., 2009) we found that the R&D department experiences severe problems with respect to maintainability of the product line related to the agile development process. The two-week Evo iterations means that the development teams needs to focus on short-term goals, at the expense of the overall structuring of the product line. It is very difficult for even experienced developers to understand the inner structure and workings of the core parts of the product line. We found that the CSoft product line has a high complexity, which negatively affects the developer's analyzability and comprehensibility, which again leads to, reduced modifiability and deployability of the software. We also found that the system is hard to test and that developers actually may display a fear of changing the code because it is hard to see effects of changes. To deal with the complexity, each development team needs a team leader that is highly skilled in one of the modules in the product line. Loosing one of these experts represents a great vulnerability. All in all we see that the agile approach to evolving the product line makes these problems permanent.

Community building: The products and services from the third-party community, which is supported by CSoft through the API and by exposing their partners, represents in total a large variability and flexibility in the CSoft product line, far larger than CSoft would have been able to develop and manage by themselves. As a product line grows in size and complexity, it becomes a platform that opens up opportunities for the development of related products and services and, which represents a business potential but also present a large challenge for the organization (Cusumano, Kahl et al., 2006). Opening up the technical interface has earlier shown to support the growth of an external community, for example IBM's opening of the PC architecture. This enabled the establishment and growth of a community (Moore, 1993), but also the growth of competitors.

The existence of the third-party community supports CSoft establishment of their agile product line organization in two respects. Firstly, it allows CSoft to maintain the maximum focus on the development and progression of the core product line (Zook, 2010). This is important - if the level of complexity were to become too high, their ability to improve the product line would be seriously impeded. Secondly, the third-party community increases the actual value of the product line to the customers, growing sales, and thus increasing the probability of a self-funding and economically sustainable SPLE process. In order to enable the third-party community to direct their efforts and development appropriately, CSoft share their high-level plans openly and, in some cases, invite third parties to planning and development meetings, just as they do with their customers.

This nearly symbiotic relationship between the third-party community and CSoft, as a SPLE organization and a provider of a partially open platform, can be described as a software ecosystem (Bosch, 2009), which is an emerging concept within software engineering and a potentially important shift in how the industry relates to other actors in the market. The term "ecosystem" is borrowed from biology (just like the term "evolution" is), and can be defined as a unit of mutually dependent organisms co-existing in the same habitat. Here, it is used as an analogy to the software industry where organizations are organisms and the marketplace is the habitat. Bosch (*ibid.*) discusses this concept in relation to SPLE and comments that *"…enabling a software ecosystem causes processes optimized for intra-organizational purposes to no longer work"* (p. 177). He calls for a change from a centralized approach to a decentralized one, in which individual and partially self-managed teams are able to develop their solutions without having to coordinate with a central organization. This is what happened at CSoft, as part of the adoption of agile development practices. It is reasonable to believe that Evo, with participation from actors outside the organization and the self-managed component teams, has been an enabling factor in the establishment of a viable software ecosystem.

Openness and visibility: CSoft has developed into a partially open organization, making their plans and processes available, both internally and in part externally. They also promote frequent communication across various levels and roles in the organization, not necessarily following a regular pattern from case to case. The work is driven by the PSG, but is best seen as a collective effort with contributions being welcomed from practically all parts of the organization. Besides creating a common responsibility for and ownership of the process, this modus operandi also opens up the planning process to the influx of ideas from external sources, and helps to balance the concerns of various stakeholders. This openness is also very much in evidence outside the company, for example, at seminars and product conferences. Long-term visions and ideas, and more detailed and short-term plans, are communicated to a mix of new and potential customers as well as third parties. These events open up a dialogue with external actors that may have complementary needs or ideas and even corrective feedback, which may become valuable input to the planning process.

This way of interacting with customers and third parties resembles what Chesbrough and others have termed *open innovation* (Chesbrough, 2003; Enkel, Gassman et al., 2009). Open innovation is, as the name implies, an approach in which innovation takes place partly across organizational boundaries, rather than within them. Such boundary-crossing processes of innovation may take many forms. Open source software development is an example of open innovation. As Chesbrough (2003) says, *"…open innovation is based on a landscape of abundant knowledge, which must be used readily if it is to provide value for the company that created it."*. In the case of CSoft, the rapid development cycles and frequent corrections from invited stakeholders enable the development organization to achieve this readiness.

Enkel et al. (2009) presents open innovation as consisting of one of three core processes. The first, which is called the outside-in process, enriches the company's own knowledge base through the integration of supplier, customers, and external knowledge sourcing. The second is called the inside-out process, in which ideas are brought to the market, which may generate new streams of income through spin-offs or joint ventures. The third, which is called the coupled process, combines the first two and best describes CSoft's approach. This openness across organizational borders and the ability to manage both internal and external innovation processes may constitute a clear competitive advantage (von Hippel, 2005). In the case of CSoft, it has enabled the organization to improve the processes by which their software product line continues to develop.

Company culture: Since its incorporation about 13 years ago, CSoft has grown considerably. Despite its growth in size and complexity, CSoft has maintained an interest in taking opportunities to improve, and has retained the willingness to change. For example, prior to the introduction of Evo, the situation became untenable because of the development process rigidity. The decision to dramatically change the development process was made very swiftly and took effect within weeks (Fægri and Hanssen, 2007). This is an example of the company culture, which is, to a large degree, aware of the company's weaknesses and open to new approaches. Although inspired by the latest trends in the software industry, the organization has managed to maintain a rather sober attitude, trying to consider what may and may not be relevant to it in practice. Hughes refers to this as *"enthusiastic problem solving and dedicated system building"* without being constrained by disciplinary and knowledge boundaries (Hughes, 1989). At CSoft, this culture has enabled the organization to adopt their agile software product line approach. Studying standard descriptions of ASD or SPLE may give the impression that these two approaches are irreconcilable. However, by selecting process components from each, without being constrained by what may be called the disciplinary rules of each approach, CSoft have managed to apply a simple, yet seemingly powerful combination.

5.1.3 Contextual factors

Domain volatility: One assumption about SPLE organizations and their target domains is that the detailed practice of product line engineering as described by e.g. the Software Engineering Institute (SEI) is suitable where the domain is fairly stable (McGregor, 2008). Serving a domain that does not change too much, means that there is a lower risk in pre-developing assets for later (re)use in application engineering. In contrast, the domain that CSoft serves is unstable, because its boundaries are still being determined. New technologies and business ideas emerge frequently; hence, it is more important to be able to respond quickly to the market than to produce reusable components that might be useful at some later time, but are not sure to be.

The software-as-a-service delivery model: Early in the development of the CSoft product line, the software-as-a-service (SAAS) deployment model (Dubey and Wagle, 2007) was adopted. When a new release is ready, all parts of the product line are released simultaneously and are deployed on a server farm. This allows customers to use the new release instantly through an ordinary web browser. This model ensures simple release of new versions of the product line, offers an easy upgrade path for the customers, and provides opportunities for reducing the costs since no local operational infrastructure and services are needed. In addition, the model is advantageous when external stakeholders are providing feedback on the recent increment – which is released on a test server, as SAAS.

5.2 An emerging software ecosystem

Building on the understanding of how the combined processes work in the product line organization and the changes this has lead to, we continue the discussion by looking into how this has enabled a change from software product line engineering towards a situation which eventually can be described as a software ecosystem. We summarize this part of the discussion by proposing a model of a software ecosystem.

5.2.1 Changing from agile software product line engineering to a software ecosystem

The initial motivation for changing the waterfall-like development process by adopting Evo in 2004/2005 was that CSoft struggled with unstable requirements, incurring high costs due to little flexibility in the process. Much emphasis was given to extensive and thorough requirements engineering upfront, but with little effect (Hanssen and Fægri, 2006). The immediate experience from involving stakeholders in the short Evo development iterations was that developers felt more comfortable and secure by having this close and continuous dialogue on requirements and results (*ibid.*). However, in the first release projects using Evo, it became a considerable challenge to maintain the motivation of the external stakeholders throughout the project. As we have seen, the new process was fragile (Fægri and Hanssen, 2007).

(Change 1 – engaging customers) From the more recent study (Hanssen and Fægri, 2008) we see that this has clearly changed; now external stakeholders are keen to participate – CSoft actually have to turn down stakeholder candidates. This change is the result of a learning process that took place during the first years of using Evo – customers have gotten to know of this practice and some gained experience as stakeholders. The engagement of customers and users is generally considered to be an important success factor in any kind of software development (Chiasson and Green, 2007; Keil and Carmel, 1995).

(Change 2 – learning by doing) We can also observe another change that took place internally at CSoft. The first experiment with Evo was done as an R&D-internal matter, like a kitchen experiment. However, as this turned out to be an improvement of the development practice, this way of working eventually became adopted by the rest of the organization. Now, all parts of the organization, from operational support to the top management, are supporting this practice. An example is the CEO explaining the software development process Evo and its strategic importance in his keynote at a large product conference. Another example is the strengthening of the PSG, which has a liaison function between customers and development teams. This tells us that changing a product line organization takes effort and time, and that both internal and external actors need to learn from practice to accept this opening of the organization and its work processes.

(Change 3 – increasing visibility) Another change we can see from the results is an increasingly higher external visibility of plans and strategies. Initially this kind of information was kept confidential, but it is now more and more openly communicated through various channels such as product conferences, at ad-hoc meetings with customers and in development projects. It has turned out that doing so does not introduce the presumed risk of leaking vital information to competitors, but that it is rather an advantage. As external actors see what might be coming, they can relate it to their own business, and potentially respond to it.

(Change 4 – increasing extensibility) Another related change is the opening of the product line at the technical level with the APIs. Initially this represented a minimal and very limited opportunity for extending the product line, but it quickly grew to a considerable extent as it represented a tangible business value. This aspect has eventually been given more attention, and has been designated as strategically important in some of the roadmaps. We see several benefits from allowing externals to use the product line as a platform. Firstly, it increases the variability of the product line – it can be used in more specialized ways, serving more needs. Secondly, existing users represent a great opportunity to the third parties (being the second component of a symbiosis-like relationship). Thirdly, letting externals deal with specialization and minor extensions enables the product line organization itself to maintain focus on developing the core product line (Zook, 2010). This may be the most important effect.

5.2.2 A case of improvisational change

As the four studies of CSoft show, the organization has undergone an extensive change over the time under study. Using the theoretical background on organizational change from section 2.1.5 as a lens, we see that CSoft is *not* a case of episodic change as described in Lewin's model (1951). The organization was not "unfrozen" in terms of removing inertia and anxiety of change. Rather, the state of the organization had stretched to a limit where the "pain" (poor process performance, low motivation, and code errors) was so high that nearly any change was perceived as better than none. One of the product managers even stated laconically: *"Whatever changes we implemented, it would probably be a good thing."* Thus, the motivation to change had grown in parallel with, and as a consequence of, the increasingly bad performance. The decision to try Evo can be seen as a case of opportunity-based change (Orlikowski and Hofman, 1997) as the decision was made quite abruptly, based on the newly acquired knowledge about Evo and iterative and incremental development.

Following CSoft through this period of time (Fægri and Hanssen, 2007) we observed several iterations of the Deming/Shewhart improvement cycle. At first only a few aspects of the new process were applied and only in one project. Then, having verified that iterative work was applicable, external stakeholders were invited in the next cycle.

The introduction and adoption of Evo can be seen as a large opportunity-based change, taking several cycles to implement. From our continued study of CSoft we have also seen examples of other types of change. For example, we saw that the iteration length was increased from one to two weeks (in release 9.1 - the third release where Evo was used (*ibid*., p.101)), and we later observed the introduction of the green-week concept (an week dedicated to error correction and stabilization of the system). These may be seen as cases of emergent change (Orlikowski and Hofman, 1997).

All in all – we see CSoft and the development they have undergone over the past years as an example of improvisational change. Further, we believe that this case also exemplifies that a low level of formalization of processes (self-managed teams, little process documentation, no domain models) and the flat organizational structure of this company makes improvisational change easier to implement. As Dybå (2000) says: "*Too much reliance on previously learned patterns tends to limit the explorative behavior necessary for improvisation.*" (p. 83).

5.2.3 A conceptual model of a software ecosystem

The study of CSoft, as a supplier of a product line, and their collaboration with external actors leads us to develop a conceptual model of a software ecosystem of the web/application type (Bosch, 2009).

CSoft has changed their role to become a central actor in a network of organizations having an interest in the product line. Customers and third parties form the external environment and can participate in both strategy making and actual development projects. Likewise, third parties can also collaborate with the supplier, as well as its customers through offering value-adding products and/or services. The product line is the central asset that collaborations and interactions are based upon. This networked organization is not deliberately created – it has developed through symbiotic relationships, meaning that each actor in some way benefits from the relationship with the others. These benefits must be in place in order to make the ecosystem work as a whole.

Table 9 - Values from being a network actor

	supplier	customer	3d party
supplier		Sales of licenses and services.	Maintaining focus on the core product line. Input to plans and development.
customer	Business enabling tools and services. Early insight into technology development.		Value adding solutions and services.
3d party	Access to an established customer base and a technical platform. Early insight into technology development.	Sales of licenses and services.	

The supplier has the central role and controls the development of the ecosystem to a large extent by controlling the flow of information and involvement of other participants. This controlling role is called the keystone advantage by Iansiti & Levien (2004). In the type of software ecosystem being described here, this lack of "democratic" influence is compensated by the deliberate support from the product line supplier. External actors are supported through information about strategies and plans and – for a few selected external actors – through direct involvement in both planning and development.

A fundamental aspect of an ecosystem is the opportunity the various actors have to learn about the product line, its development, and its use. Also, talking directly to customers, allows for learning about the domain. On the other hand, customers and third parties learn about the ongoing development. Such proximity are found to be important to balance long-term strategic objectives of SPLE and the short-term tactical objectives of ASD, and to make the development organization adaptive with respect to its external environment (Mohan, Ramesh et al., 2010).

However, there is also a reason to be concerned with the effects of increased customer proximity and iterative development. As we have seen from CSoft (Hanssen, Yamashita et al., 2010) this can aggravate software entropy due to the strong focus on low-level product qualities, at the cost of high-level qualities such as for example system architecture.

Developing a product line within such an ecosystem has the potential of producing some beneficial effects. By having external organizations dealing with minor extensions or specialized variants, the supplier can maintain focus on developing the core (Zook, 2010). Also, having externals to build on the product line, using it as a platform, increases the diversity and potential value of the product line and increases its variability. Collaborating with deliberately selected external actors enables open innovation (Chesbrough, 2003), both in the long-term in strategic planning and in the short-term. We interpret CSoft's ecosystem and its evolutionary development process as a variant of Chesbrough's open innovation model (*ibid.*, p.xxv):

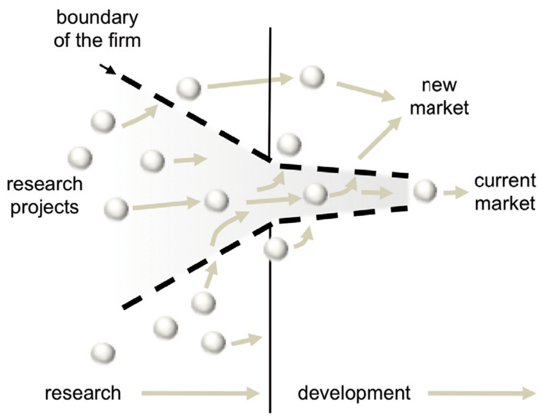


Figure 14 - Open innovation

The borders of the organization are permeable in the sense that external organizations influence and even contribute to research (strategic planning and road-mapping) and development (agile development projects). New releases enter the market, which is extended by external actors who run their own research and development processes, building on and extending the product line.

Another aspect of the ecosystem is how it relates to knowledge creation and dissemination. Compared with a closed approach (CSoft prior to the adoption of Evo and the engagement of external stakeholders) an ecosystem enables new spaces for creation of knowledge. We have described some of these, for example the advisory board, ad-hoc meetings with customers, and the participation of stakeholders in the development projects. Nonaka and Konno (2008) refer to such spaces as *"ba"* - Japanese for *space*. "Ba" can be physical (e.g. office space), virtual (e.g. e-mail conversations), mental (e.g. ideas), or any combination of the above. Knowledge creation is a spiraling process of interactions between explicit and tacit knowledge. Nonaka's SECI[9]-model describes four types of knowledge transition (*ibid.*, p.43). *Socialization* is the transfer of tacit knowledge between individuals. *Externalization* is the transition of tacit knowledge to explicit knowledge, which can be understood by others indirectly. *Combination* is the construction of more complex sets of explicit knowledge. *Internalization* is the transition of explicit knowledge into the organization's collective tacit knowledge. Applying these principles to the study of CSoft, we see that the establishment of the ecosystem, with its spaces/ba of contact

[9] Socialization, Externalization, Combination, Internalization

between internals and externals enables knowledge creation through such transitions. For example, frequent meetings between teams of developers and external stakeholders (as close to face-to-face as practically and technically possible) enable transition of tacit knowledge (socialization). During the iteration evaluation meetings stakeholders can explain directly to the development team how they work and how they use the software product under development. This demonstrates the effect of the agile components in the SPLE process where *direct* and *frequent* communication enables knowledge creation.

To wrap up this discussion, we propose a simple conceptual model of a software ecosystem. This model represents the case we have studied, and could serve as a basis to reflect on and guide other cases.

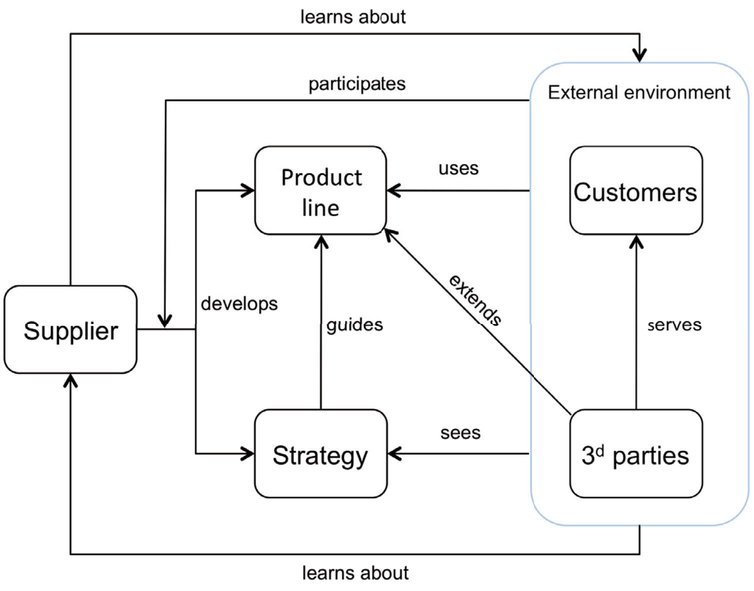


Figure 15 - A conceptual model of a software ecosystem

The model illustrates the main actors in a software ecosystem. The supplier (keystone organization) develops the product line. This development is guided by a strategy, which points out needs and opportunities and main paths of development. Both the strategy and the development of the product line are to some extent visible to external actors. These consist of (at least) customers and third parties, but can also involve others. Third parties use the product line as a platform to serve customers with additional solutions and services. Being a part of an ecosystem means that these actors learn about each other. The supplier learns about requirements, needs, ideas, opportunities etc. In return, external actors learn about the development of the technology of common interest (the product line), and may even participate actively in the development.

5.3 Implications for theory

We believe that software ecosystems have the potential of becoming an important field of practice and research in the years to come. As shown in section 2.1.4, the interest is growing and some work has been done, but the overall impression is that this concept is yet poorly developed and understood. As a contribution to future development of this concept, we propose to form a theoretical platform. Just like the taxonomy suggested by Bosch (2009), a theory of software ecosystems, as suggested in this section, can become valuable and useful to generalize the concept and bring together results from more empirical studies. It may over time develop towards a unified and empirically justified understanding of the concept (Sjøberg, Dybå et al., 2007), being useful to both research and practice.

The theory of organizational ecology (Trist, 1977) which was briefly presented in the background section (2.3), are used a framework for the following reasons:

- It builds on the concept of *socio-technical systems*, which covers the interaction between humans and technology. 'Technology' also covers organizational structures, processes, techniques etc. This is a useful concept in order to understand the CSoft case, which is a social system, a technical system *and* the interaction between these. For example, the people in this organization collaborate internally to continuously develop a software product line, the technology. The other way around, this technology clearly affects how these people develop the technology, for example illustrated by the problems related to the system entropy.
- It defines the concept of *the external environment*. This has become a very important aspect for CSoft as they have moved from a closed water-fall like approach with little emphasis on, and contact with, external actors to the present practice where customers and third parties are closely involved in planning and development, supported by ASD principles such as short iterations, incremental development and frequent feedback through interaction with external actors.
- It discusses types of organizational environments, ordered according to complexity whereof the most complex type and most recent in development, called *turbulent fields* describes CSoft's environment. CSoft operates in a turbulent domain where their product line is constantly changing, the development organization is changing, as well as the technology they use and the business domain they serve. Trist says *"..with the increasing salience of turbulent conditions, systems of aggregate control*

are becoming increasingly insufficient and inefficient, no matter how large the outgrowths." (ibid., p.169).

Related to the study of CSoft reported in this thesis, we use Trist's theory as a framework and derive a set of theoretical propositions suitable for describing software ecosystems. This means that we have not done a 1:1 adoption of concepts from Trist's description, but used it as a basis for our own. Sjøberg et al. (2007) explains three ways in which theories are built, whereof the second type is applied here: *"Theories from other disciplines may be adapted to software engineering before use."* (p. 5). This means here that the theory of organizational ecology identifies concepts, which we have found to be relevant to the field of software engineering. As a mean to describe this theory we have looked to the guidelines and a five-step approach provided by Sjøberg et al. (ibid., p.13) and thus describe it as a set of propositions, which explains how basic constructs relate to each other (step 1 and 2). Five propositions have been derived from Trist's work and adapted to software engineering. In addition, we also propose two new ones, based on the study of CSoft. Each of the seven propositions is supported by explanations from our case study to justify the theory (step 3). Finally we discuss the scope of the theory (step 4). Step 5 is to test the theory through empirical research and will be left to later work. Later empirical studies may extend, change, confirm or criticize the theory, as presented here.

1. *Member organizations in a software ecosystem are linked to a key organization among them, which acts as* ***a central referent organization****, doing so even though many of the* ***members*** *are only partially under its control or linked to it only through* ***interface relations****. (This is a Class 1 system according to Trist's classification (1977) (p. 165))*

   CSoft is an example of a referent organization in a software ecosystem, where other members are their customers, and third party organizations that use CSoft's technology as a platform for providing related products and/or services. None of these external organizations are formally controlled by CSoft. However, all activity in the ecosystem is related to the product line, which is owned, developed and thus controlled by CSoft. These relationships resemble those in open source initiatives, however with the distinction that the software is closed and owned by one actor[10]. The interface relations between the referent organization and other members are nurtured by ASD practices such as customer engagement in the planning of iterations and the evaluation of the outcome and by the open communication of product strategies, for example on product conferences or on an ad-hoc basis.

[10] Future research on open source ecosystems may consider Trist's second class of socio-ecological organization as an explanatory model.

2. *Software ecosystems promote **self-regulation**.*

   Our study of CSoft shows that the collaborative approach can be seen as a self-regulating system in that the referent organization to a large degree adapts to its external environment, and that the external environment adapts to the referent organization. The main enabler for this is the close and frequent co-operation between CSoft and externals, which takes place in the development iterations. Demonstration of results, feedback and correction takes place in each iteration. This stands in contrast to the previously centralized control that was applied in the development of the product line where feedback was rare.

3. *Software ecosystems have a **networked character**.*

   CSoft and its external environment constitute an open-ended network of customers and third party organizations, where all relate to the referent organization but with the additional opportunity of relationships between member organizations outside the direct control of the referent organization. For example, some of CSoft's customers are also customers of third part organizations, using related services or extensions or specializations of the product line. The referent organization benefits from this network and in the case of CSoft also support it actively, for example by marketing third parties and by offering technological interfaces enabling these third-party operations.

4. *Software ecosystems exist through the use of **information and communication technology (ICT)**.*

   The ecosystem, which CSoft is a part of, relies considerably on the use of ICT to enable the collaboration with its external environment. Examples are the web-meetings with external stakeholders in the development iterations. This enables collaboration with actors at different locations than the development teams. We have also seen that the software-as-a-service deployment model makes distribution of increments easy and efficient, which enables external stakeholders to test increments and provide feedback.

5. *Software ecosystems exhibit **shared values**.*

   In the CSoft ecosystem the software (product line) is a shared value. For CSoft, the value is revenue from licenses and services, for the customers the value is improved business operations, and for the third parties the value is revenue from sales of value-adding services and solutions (see Table 9). This common interest creates a motivation to care collaboratively for the shared value. For example, since the product line has become an inevitable part of the business of the customers, they are willing to invest time to participate in the development of this value.

These five propositions constitute a foundation of a theory for software ecosystems. In addition to these principles adopted from Trist's work (*ibid.*) we also propose two extensions:

6. ***The shared value of a software ecosystem is both the software product and the business domain****.*

   The focus of the referent organization is not restricted to the product line only. It also covers the business domain that is supported/enabled by the product line. The better the product line is at supporting the business of customers and third parties, the stronger the incentive to use the product line becomes as well as the incentive to participate in the continuous development of it. The continuous dialogue between the referent organization and external stakeholders, in each iteration in the development projects, addresses both how to develop or improve the product line as well as how the product line supports the business of the stakeholders. We see this as a case of open innovation.

7. *As a software ecosystem emerges,* ***control moves from the supplier of the software to its users****.*

   Prior to the adoption of Evo, the only guiding input from the customers was through formal requirements documents. Feedback came late, very close to release. CSoft controlled all aspects of the development. However, this was not a beneficial situation due to low input from customers. Now, some of this control has moved to (participating) external stakeholders that to a larger extent control the development of the product line. Yet, CSoft still owns the product line and controls the main lines of development, but now in combination with vital input from external actors. Here lies a clear distinction from open source initiatives.

The fourth step of building a software engineering theory (Sjøberg, Dybå et al., 2007) is to determine the scope. The propositions given here are based on the study of a single software product line organization (and inspired by a general theory on organizational ecology). With this in mind, we identify the following limits of the scope:

- The proposed theory concerns software product line organizations similar to CSoft, which emphasize iterative and incremental development and participation by externals in strategy making and in development.

- We believe that this theory is relevant to software organizations serving business domains that is still shaping and not mature domains.
- We believe that this theory is relevant to software organizations that develop products that are vital to the business performance of the customers and where the use of the product is long-term.

5.4 Implications for practice

Based on our combined study of CSoft, our overview of related research and literature, and our discussions we propose a set of implications for practice. We shape these implications as advice or guidelines for software organizations similar to CSoft – or – software organizations that consider a change towards agile software product line engineering and software ecosystems.

- Support the external environment by being open. Sharing information on plans, strategies, and development may create a fundament for collaboration and new patterns of innovation. If appropriate, encouraging and supporting a third party community can be a valuable extension to the normal development of the product line.
- Reducing the variability in the product line consequently reduces the costs of maintaining and managing variability. The potential drawback of reduced variability can be compensated through offering interfaces for extending the product line by externals, and thus increasing the overall variability.
- Establishing and benefiting from a software ecosystem takes time. A successful development relies on repeated cycles of experimentation and learning. This learning process needs to involve all types of actors.

5.5 Limitations

The case study of CSoft is subject to four limitations.

(1) This is a single case study, which naturally affects the *generalizability* of the conclusions. Yet there are good reasons for choosing such an approach. First of all, the number of relevant cases is still low. Nevertheless, we have had the opportunity to study a case, which we believe is both advanced and relevant to other parts of the software industry. In addition, focusing on a single case means that the study can be more thorough than a study of multiple cases, with respect to available resources. Yin (2002) discusses the single case study design (p. 38-41) and presents several arguments in favor of choosing such a design. One of these is particularly applicable to CSoft, namely

that it is a unique case. According to Yin, such a study may act as a prelude to further studies of a relatively new topic, such as software ecosystems in this case.

(2) A large part of the data that has been collected comes from group and single-respondent interviews with internals and externals. This type of data can potentially be *biased*, incomplete or even wrongful due to misunderstandings, lack of insight etc. We have sought to address this threat to validity by collecting data from various respondents and supplementing these data with documents and observations.

(3) The third limitation concerns the *completeness* of the study. Only a subset of the employees was contacted. Likewise, relatively few samples of all available documentation were collected and analyzed. This is due to natural limitations such as limited time and resources.

(4) The fourth limitation concerns the *applicability* of the findings and conclusions of this study. The organization investigated is a medium-size product line organization and a web/application type of ecosystem (according to the taxonomy proposed by Bosch (2009)). Thus, results do not necessarily apply to all other types of software ecosystems.

6 *Conclusions and further work*

The overall goal of this study has been *to understand the need for and implications of a more open approach to software product line engineering*. To establish this understanding, we have followed a growing software product line organization and the changes they have undergone over a period of approximately six years.

This longitudinal study of CSoft and their development began by investigating the abrupt change of the development process through the adoption of some key agile principles in the inflexible development process. The main focus of study 1 was how the development organization changed the way they related to their customers. The insight into this change and the results of the study led to the collection of new data used in studies 2 and 3, focusing on agile software product line engineering and software entropy, respectively. Through these studies, we became aware of how the relationships with external actors were changing, and we initiated the collection of more data to investigate in more detail how this organization developed their relationship to external actors, leading to study 4.

This brief retrospect shows that the focus and the motivation of the studies have shaped and developed during the course of the study of CSoft. This means that the research questions have also developed in this way.

6.1 Answering the research questions

RQ1: How can software product line engineering and agile software development be combined?

Firstly, we have shown that combining the plan-driven product line engineering approach with an incremental and iterative agile development process is practically feasible. Developing and evolving a product line requires long term planning, which results in high-level plans. These are used to initiate shorter-term agile development projects. We have described the process combination as a sum of three processes with varying time-horizons. Long-term planning and activities related to the evolution of the product line are organized within a strategic process. Development is organized as a tactical process. These two interact with the operational process, encompassing day-to-day functions. Active engagement of and collaboration with external stakeholders is the driving force in this process combination. We have found several desirable effects of collaborating with externals such as increased motivation among developers as well as faster and more frequent feedback. Continuous collaboration with externals requires continuous management of the relationship to maintain motivation and to replace stakeholders if needed. Frequent feedback and re-planning from short iterations come with the risk of architectural erosion and increased system entropy. We have seen the need to compensate potential erosion by a dedicated function (the architecture team), which ensures the maintenance and development of the overall architecture of the product line. We have also observed process fragility caused by the increased frequency of iterative development. An agile development process within a product line organization requires a high level of process discipline, and small deviations may cause process instability. We have observed such fragility when external stakeholders failed to fulfill their role. Also, we have demonstrated that frequent collaboration with external stakeholders in strategic, tactical, and operational processes creates additional overhead as compared to a traditional plan-driven product line approach.

RQ2: How does a software ecosystem shape?

Based on the answer to RQ 1, we have further investigated how the process combination led towards a situation that we describe as a software ecosystem. Fundamental to this change is the process of opening up. We have shown how the case organization moved towards a software ecosystem by 1) opening the information flow internally and externally, 2) opening the innovation processes by collaborating with external actors at various levels, and 3) opening the technical interfaces, enabling external actors to use the product line as a platform for additional services and solutions. The transition to an ecosystem has come along with a change from a traditional closed systems mindset to an open systems mindset. We have shown how actors in the external environment contribute to both long-term strategic planning and to short-term tactical development. Externals can be both customers/end-users and third parties. We have also observed how the product line organization actively supports a community of external actors and the benefits that come from such collaboration. Examples of such benefits are: increased focus on the development of the core product line, increased variability of the product line, and business opportunities for third parties.

The change from a closed product line approach to a partly open ecosystem happened as an improvisational change process. It did not happen through the implementation of a plan of change but began with careful testing of some core principles in practice. Based on experience, new changes where introduced, forming a continuous process improvement initiative.

To foster further studies of the emergence and functions of software ecosystems we proposed a theory of software ecosystems, derived from socio-technical theory and organizational ecology.

Finally, we revisit our overall research objective:

Understanding the need for and implications of a more open approach to software product line engineering.

The software product industry recognizes an increasing need to become more open and responsive to its market. Software technology becomes an increasingly important part of most businesses, and the users of this technology expect a more influential role for themselves in its development (Messerschmitt and Szyperski, 2003; von Hippel, 2005). This attitude is reinforced by an increased opportunity to collaborate in terms of technology, tools, and agile processes.

The implications of this change are an increased proximity to the customers and a potential to better capture needs and opportunities in the market. The software product supplier gets a larger role in the shaping of both software products and services, and the business domain they support. Understanding, supporting, and exploiting software ecosystems become a competitive advantage.

6.2 Directions for future work

(1) As Jansen et al. also point out (2009), we need to see more empirical studies of various types of software ecosystems, how they develop, and the effects they produce. Such studies should naturally be focused towards the industry, and be longitudinal as well as exploratory.

(2) To build a common understanding of software ecosystems: how they shape, how they work, and what their effects are, we advise a further refining of a theory of software ecosystems, as the one proposed in section 5.3.

(3) The emergence of software ecosystems comes with new business models affecting intellectual property rights, economic models, competition etc. We need to see more dedicated studies of these issues to realize the potential of ecosystems.

(4) Software ecosystems are closely related to the more mature concept of open source software development. We need to better understand the similarities and the differences in order to transfer knowledge between these two related domains (Fitzgerald, 2006).

(5) The engine of a software ecosystem is the collaboration with external actors. We have showed some examples through our studies, but this is a broad topic that needs further investigation.

(6) The study of software ecosystems potentially relates to several disciplines such as business strategy, sociology, technology and innovation management, economy, and others. We have briefly touched a few of these and we see a need to investigate these links further.

(7) Software ecosystems affect the shape of control structures. We believe that control shifts from the supplier towards the users, a transition that needs to be better understood.

References

Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J. (2002). *Agile software development methods - Review and analysis*, VTT Electronics**:** 112.

Adler, P. S. (2005). *The Evolving Object of Software Development.* Organization **12**(3): 401-435.

Ahmed, F. and Capretz, L. F. (2006). *Managing the business of software product line: An empirical investigation of key business factors.* Journal on Information and Software Technology **49**(2): 194-208.

Alspaugh, T. A., Hazeline, U. A. and Scacchi, W. (2009). *The Role of Software Licenses in Open Architecture Ecosystems*. In proceedings of First International Workshop on Software Ecosystems (IWSECO´09), Milan, Italy, September 27. 4-18.

Arisholm, E., Gallis, H. G., Dybå, T. and Sjøberg, D. I. K. (2007). *Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise.* IEEE Transactions on Software Engineering **33**(2): 65-86.

Babar, M. A., Ihme, T. and Pikkarainen, M. (2009). *An industrial case of exploiting product line architectures in agile software development*. In proceedings of 13th International Software Product Line Conference, San Fransisco, California, 24-28 August. 171-179.

Basili, V. R. (1996). *The Role of Experimentation in Software Engineering: Past, Current, and Future*. In proceedings of 18th International Conference on Software Engineering (ICSE´96), Berlin, Germany, March 25-29. 442-449.

Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J. and Slaughter, S. (2003). *Is "Internet-speed" software development different?* IEEE Software **20**(6): 70-77.

Bayer, J., Gacek, C., Muthig, D. and Widen, T. (2000). *PuLSE-I: Deriving Instances from a Product Line Infrastructure*. In proceedings of Seventh IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS 2000), Edinburgh, Scotland, 3-7 April. 237-245.

Beck, K. (1999). *Extreme programming explained: embrace change*. Boston: Addison-Wesley Professional.

Beck, K. and Andres, C. (2004). *Extreme programming explained: embrace change (2nd Edition)*. Boston: Addison-Wesley Professional.

Birk, A., Dingsøyr, T. and Stålhane, T. (2002). *Postmortem: Never Leave a Project without It.* IEEE Software **19**(3): 43 - 45.

Böckle, G. (2005). *Innovation management for product line engineering organizations*. In proceedings of 9th International Software Product Line Engineering Conference (SPLE'05), Rennes, France, 26-29 September. 124-134.

Böckle, G., Clements, P., McGregor, J. D. and Muthig, D. (2004). *Calculating ROI for software product lines.* IEEE Software **21**(3): 23-31.

Boehm, B. (2006). *A view of 20th and 21st century software engineering* International conference on Software engineering (ICSE).

Boehm, B. and Turner, R. (2004). *Balancing Agility and Discipline - A Guide for the Perplexed.* Boston: Addison-Wesley.

Bosch, J. (2001). *Software product lines: organizational alternatives*. In proceedings of 23d International Conference on Software Engineering (ICSE'01), Toronto, Canada 12-19 May. 91-100.

Bosch, J. (2009). *From Software Product Lines to Software Ecosystems*. In proceedings of 13th International Software Product Line Conference (SPLC'09), San Fransisco, USA, 24-28 August. 111-119.

Bosch, J. and Bosch-Sijtsema, P. (2009). *From integration to composition: On the impact of software product lines, global development and ecosystems.* Journal on Systems and Software **83**(1): 67-76.

Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, H. and K., P. (2001). *Variability Issues in Software Product Lines*. In proceedings of Fourth International Workshop on Product Family Engineering, Bilbao, Spain, 3-5 October. 1-9.

Brown, A. W. and Booch, G. (2002). *Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors*. In proceedings of 7th International Conference on Software Reuse: Methods, Techniques, and Tools, Austin, USA, April 15-19. 123-136.

Buhrdorf, R., Churchett, D. and Krueger, C. W. (2004). *Salion's Experience with a Reactive Software Product Line Approach*. In proceedings of Software Product-Family Engineering, Siena, Italy. 317-322.

Carbon, R., Lindvall, M., Muthig, D. and Costa, P. (2006). *Integrating Product Line Engineering and Agile Methods: Flexible Design Up-front vs. Incremental Design*. In proceedings of 1st International Workshop on Agile Product Line Engineering (APLE'06), Baltimore, USA, 21 August. 1-8.

Chesbrough, H. (2003). *Open Innovation: The New Imperative for Creating And Profiting from Technology*. Boston: Harvard Business School Publishing Corporation.

Chesbrough, H. (2006). *Open Innovation: A New Paradigm for Understanding Industrial Innovation*. In Open Innovation: Researching a New Paradigm. Chesbrough, H., Vanhaverbeke, W. and West, J. (eds.). Oxford: Oxford University Press**:** 1-12.

Chesbrough, H. W. (2003). *The Era of Open Innovation.* MIT Sloan Management Review **44**(3): 9.

Chiasson, M. W. and Green, L. W. (2007). *Questioning the IT artefact: user practices that can, could, and cannot be supported in packaged-software designs.* European Journal of Information Systems(16): 542-554.

Clements, P. C. and Northrop, L. (2002). *Software Product Lines: Practices and Patterns*. New Jersey: Addison-Wesley.

Cockburn, A. (2002). *Agile Software Development*. Boston: Addison-Wesley.

Cooper, D. R. and Schindler, P. S. (2006). *Business Research Methods*. New York: McGraw-Hill.

Cusumano, M., Kahl, S. and Suarez, F. F. (2006). *Product, Process, and Service: A New Industry Lifecycle Model (Working Paper)*, Harvard Business School.

Daft, R. L., Lengel, R. H. and Trevino, L. K. (1987). *Message Equivocality, Media Selection, and Manager Performance: Implications for Information Systems.* MIS Quarterly **11**(3): 355-366.

Damodaran, L. (1996). *User involvement in the systems design process - a practical guide for users.* Behavior & Information Technology **15**(6): 363-377.

Davison, R. M., Martinsons, M. G. and Kock, N. (2004). *Principles of canonical action research.* Information Systems Journal **14**(1): 65-86.

Deming, W. E. (2000). *Out of the Crisis*. Cambridge: The MIT Press.

Dubey, A. and Wagle, D. (2007). *Delivering software as a service.* The McKinsey Quarterly **May**: 1-12.

Dybå, T. (2000). *Improvisation in small software organizations.* IEEE Software **17**(5): 82 - 87.

Dybå, T. and Dingsøyr, T. (2008). *Empirical Studies of Agile Software Development: A Systematic Review.* Information and Software Technology **50**(9-10): 833-859.

Dybå, T., Dingsøyr, T. and Moe, N. B. (2004). *Process Improvement in Practice - A Handbook for IT Companies*. Dordrecht: Kluwer Academic Publishers.

Eckstein, J. (2004). *Agile Software Development in the Large - Diving Into the Deep*. New York: Dorset House Publishing.

Emery, F. E. and Trist, E. L. (1965). *The Causal Texture of Organizational Environments.* Human Relations **18**: 21-32.

Enkel, E., Gassman, O. and Chesbrough, H. (2009). *Open R&D and open innovation: exploring the phenomenon.* R&D Management **39**(4): 311-316.

Erdogmus, H. and Morisio, M. (2005). *On the Effectiveness of the Test-First Approach to Programming.* IEEE Transactions on Software Engineering **31**(3): 226-237.

Fægri, T. E. and Hanssen, G. K. (2007). *Collaboration and Process Fragility in Evolutionarily Product Development.* IEEE Software **24**(3): 96-104.

Fenton, N. (1994). *Software Measurement: A Necessary Scientific Basis.* IEEE Transactions on Software Engineering **20**(3): 199-205.

Fitzgerald, B. (2006). *The Transformation of Open Source Software.* MIS Quarterly **30**(3): 587-598.

Fricker, S. (2009). *Specification and Analysis of Requirements Negotiation Strategy in Software Ecosystems*. In proceedings of First International Workshop on Software Ecosystems, Milan, Italy, 28 September. 19-33.

Ghanam, Y. and Maurer, F. (2009). *Extreme Product Line Engineering: Managing Variability & Traceability via Executable Specifications*. In proceedings of Agile Conference 2009, Chicago, USA, 24-28 August. 41-48.

Ghanam, Y. and Maurer, F. (2010). *Extreme Product Line Engineering – Refactoring for Variability: A Test-Driven Approach*. In proceedings of 11th International Conference on Extreme Programming (XP2010), Trondheim, Norway, 1-4 June. 43-57.

Gilb, T. (2005). *Competitive Engineering: A handbook for systems engineering, requirements engineering, and software engineering using Planguage*. Burlington: Elsevier Butterworth-Heinemann.

Glaser, B. G. and Strauss, A. L. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. New York: Aldine Transaction.

Grunbacher, P. and Briggs, R. O. (2001). *Surfacing tacit knowledge in requirements negotiation: experiences using EasyWinWin*. In proceedings of 34th Hawaii International Conference on System Sciences (HICSS'01), Hawaii, USA, 3-6 January. 1-8.

Hanssen, G. K. (2010). *A Longitudinal Case Study of an Emerging Software Ecosystem: Implications for Practice and Theory* Journal on Systems and Software **Under review**.

Hanssen, G. K. (2010). *Opening up Software Product Line Engineering*. In proceedings of 1st

International Workshop on Product Line Approaches in Software Engineering, in conjunction with the 32'nd International Conference on Software Engineering (ICSE), Cape Town, 2 May. 1-7.

Hanssen, G. K. and Fægri, T. E. (2006). *Agile Customer Engagement: a Longitudinal Qualitative Case Study*. In proceedings of 5th International Symposium on Empirical Software Engineering (ISESE'06), Rio de Janeiro, Brazil, 21-22 September. 164-173

Hanssen, G. K. and Fægri, T. E. (2008). *Process Fusion - Agile Product Line Engineering: an Industrial Case Study.* Journal of Systems and Software **81**: 843-854.

Hanssen, G. K., Yamashita, A. F., Conradi, R. and Moonen, L. (2010). *Software Entropy in Agile Product Evolution*. In proceedings of 43d Hawaiian International Conference on System Sciences (HICSS'10), Hawaii, USA, 4-7 January. 1-10.

Hanssen, G. K., Yamshita, A. F., Conradi, R. and Moonen, L. (2009). *Maintenance and agile development: challenges, opportunities and future directions*. In proceedings of 25th International Conference on Software Maintenance (ICSM'09), Edmonton, Canada, 20-24 September. 487-490.

Hansson, C., Dittrich, Y. and Randall, D. (2004). *Agile processes enhancing user participation for small providers of off-the-shelf software*. In proceedings of Extreme Programming and Agile Processes in Software Engineering (XP 2004), Garmisch-Partenkirchen, Germany. 175-183.

Hughes, T. P. (1989). *The Evolution of Large Technological Systems*. In The Social Construction of Technological Systems. Bijker, W., Hughes, T. P. and Pinch, T. (eds.). Camebridge: The MIT Press**:** 51-82.

Iansiti, M. and Levien, R. (2004). *Strategy as Ecology.* Harward Business Review(March): 1-12.

Jansen, S., Brinkkemper, S. and Finkelstein, A. (2009). *Business Network Management as a Survival Strategy: A Tale of Two Software Ecosystems*. In proceedings of First International Workshop on Software Ecosystems, Milan, Italy, 28 September. 34-48.

Jansen, S., Finkelstein, A. and Brinkkemper, S. (2009). *A sense of community: A research agenda for software ecosystems*. In proceedings of 31st International Conference on Software Engineering (ICSE'09), Vancouver, Canada, 16-24 May. 187-190.

Johansen, T. (2005). *Using evolutionary project management (Evo) to create faster, more userfriendly and more productive software. Experience report from FIRM AS.* In proceedings of 6th International Conference on Product Focused Software Process Improvement (PROFES'05), Oulu, Finland, 13-15 June. 216-223.

Keil, M. and Carmel, E. (1995). *Customer-Developer Links in Software Development.* Communications of the ACM **38**(5): 33-44.

Kiesgen, T. and Verlage, M. (2005). *Five years of product line engineering in a small company*.

Kitchenham, B. A., Dybå, T. and Jørgensen, M. (2004). *Evidence-based Software Engineering*. In proceedings of International Conference on Software Engineering (ICSE´04), Edinburgh, Scotland, 23-28 May. 273–281.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K. and Rosenberg, J. (2002). *Preliminary guidelines for empirical research in software engineering.* IEEE Transactions on Software Engineering **28**(8): 721 -734.

Klein, H. K. and Myers, M. D. (1999). *A set of principles for conducting and evaluating interpretive field studies in information systems.* MIS Quarterly **23**(1): 67 - 93.

Kniberg, H. (2007). *Scrum and XP from the Trenches - How we do Scrum*: InfoQ.

Korkala, M., Abrahamsson, P. and Kyllönen, P. (2006). *A Case Study on the Impact of Customer Communication on Defects in Agile Software Development*. In proceedings of Agile Conference, 23-28 July. 76-88.

Korkala, M., Pikkarainen, M. and Conboy, K. (2009). *Distributed Agile Development: A Case Study of Customer Communication Challenges*. In proceedings of Agile Processes in Software Engineering and Extreme Programming (XP´09), Sardinia, Italy. 161-167.

Koskela, J. and Abrahamsson, P. (2004). *On-site customer in an XP project: Empirical results from a case study*. In proceedings of European Conference on Software Process Improvement (EuroSPI 2004), Trondheim, Norway, 10-12 November. 1-11.

Langley, A. (1999). *Strategies for Theorizing from Process Data.* Academy of Management Journal **24**(4): 691-710.

Larman, C. and Vodde, B. (2008). *Scaling Lean & Agile Development - Thinking and Organizing Tools for Large-Scale Scrum*. Boston: Pearson Eduction Inc.

Leffingwell, D. (2007). *Scaling Software Agility - Best Practices for Large Enterprises*. Boston: Pearson Education Inc.

Lettl, C. (2007). *User involvement competence for radical innovation.* Journal of Engineering and technology management **24**(1-2): 53-75.

Lewin, K. (1946). *Action Research and Minority Problems.* Journal of Social Issues **2**(4): 34-46.

Lewin, K. (1951). *Field Theory in Social Sciences*. New York: Harper & Row.

Linden, F. v. d. (2002). *Software product families in Europe: The ESAPS and CAFÉ projects*. IEEE Software. **19:** 41-49.

Linden, F. v. d., Schmid, K. and Rommes, E. (2007). *Software Product Lines in Action*. Berlin Heidelberg: Springer Verlag.

Martin, A., Biddle, R. and Noble, J. (2004). *The XP customer role in practice: three studies*. In proceedings of Agile Development Conference, 2004, Salt Lake City, USA, 22-26 June. 42-54.

Martin, A. M. (2009). The role of customers in extreme programming projects (thesis). Victoria University of Wellington.

McGregor, J. D. (2008). *Agile Software Product Lines, Deconstructed.* Journal of Object Technology **7**(8): 7-19.

Merisalo-Rantanen H., T. T., Rossi M. (2005). *Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases.* Journal of Database Management **16**(4): 41-61.

Messerschmitt, D. G. and Szyperski, C. (2003). *Software Ecosystems, Understanding an Indespensable Technology and Industry*. Cambridge: The MIT Press.

Moe, N. B., Dingsøyr, T., Dybå, T. and Johansen, T. (2002). *Process guides as software process improvement in a small company*. In proceedings of EuroSPI, Nuremberg, Germany, 18-20 September. 177-188.

Mohan, K., Ramesh, B. and Sugumaran, V. (2010). *Integrating Software Product Line Engineering and Agile Development.* IEEE Software **27**(3): 48-55.

Moore, J. F. (1993). *Predators and prey: A new ecology of competition.* Harvard Business Review **71**(3): 75-86.

Müller, M. and Hagner, O. (2002). *Experiment about test-first programming.* Software, IEE Proceedings **149**(5): 131-136.

Nonaka, I. and Konno, N. (2008). *The concept of "ba": Building a foundation for knowledge creation.* California Management Review **40**(3): 40-54.

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

Orlikowski, W. J. and Hofman, J. D. (1997). *An Improvisational Model for Change Management: The Case of Groupware Technologies.* Sloan Management Review **Winter**.

Pettigrew, A. M. (1990). *Longitudinal Field Research on Change: Theory and Practice* Organization Science **1**(3): 267-292.

Pohl, K., Böckle, G. and F., v. d. L. (2005). *Software Product Line Engineering: Foundations, Principles, and Techniques*. Heidelberg: Springer Verlag.

Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit for Software Development Managers*. New Jersey: Addison Wesley.

Reel, J. S. (1999). *Critical Success Factors In Software Projects.* IEEE Software **16**(3): 18-23.

Royce, W. W. (1970). *Managing the development of large software systems*. In proceedings of IEEE WESCON, Loas Angeles, USA, 25-28 August. 1-9.

Sanchez, R. and Mahoney, J. T. (1996). *Modularity, Flexibility, and Knowledge Management in Product and Organization Design.* Strategic Management Journal **17**(Special Issue: Knowledge and the Firm): 63-76.

Sawhney, M., Verona, G. and Prandelli, E. (2005). *Collaborating to Create: the Internet as a Platform for Customer Engagement in Product Evolution.* Journal on Interactive Marketing **19**(4): 4-28.

Schwaber, K., Beedle, M. (2001). *Agile Software Development with Scrum*. New Jersey: Prentice Hall.

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

Sjøberg, D. I. K., Dybå, T., Anda, B. C. D. and Hannay, J. E. (2007). *Building Theories in Software Engineering*. In Advanced Topics in Empirical Software Engineering. Shull, F., Singer, J. and Sjøberg, D. I. K. (eds.). Heidelberg: Springer Verlag.

Smaccia, P. (2008). *Getting rid of spaghetti code in the real-world: a Case Study.* Retrieved March, 2009, from http://codebetter.com/blogs/patricksmacchia/archive/2008/09/23/getting-rid-of-spaghetti-code-in-the-real-world.aspx.

Takeuchi, H. and Nonaka, I. (1986). *The New New Product Development Game.* Harward Buisiness Review(Jan/Feb).

Tian, K. and Cooper, K. (2006). *Agile and Software Product Line Methods: Are They So Different?* In proceedings of 1st International Workshop on Agile Product Line Engineering (APLE'06), Baltimore, USA, 21 August.

Tichy, W. (1998). *Should Computer Scientists Experiment More?* IEEE Computer **31**(5): 32-40.

Trist, E. L. (1977). *A Concept of Organizational Ecology.* Australian Journal of Management **2**(2): 161-175.

Trist, E. L. and Bamforth, K. W. (1951). *Some social and psychological consequences of the longwall method of coal-getting.* Human Relations **4**(1): 3-38.

Tushman, M. L. and Romanelli, E. (1985). *Organizational Evolution: A Metamorphosis Model of Convergence and Reorientation*. In Organization Change: A Comprehensive Reader. Burke, W. W., Lake, D. G. and Paine, J. W. (eds.). San Fransisco: Jossey-Bass.

Vliet, H. v. (2002). *Software Engineering - Principles and Practice*. New York: John Wiley & Sons.

von Hippel, E. (1996). *Lead Users: A Source of Novel Product Concepts.* Management Science **32**(7): 791-805.

von Hippel, E. (2005). *Democratizing innovation*. Cambrige: MIT Press.

Weick, K. E. and Quinn, R. E. (1999). *Organizational Change and Development.* Annual review of psychology **50**(Feb.): 361-386.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. and Wesslén, A. (2000). *Experimentation in software engineering: an introduction*. Boston: Kluwer Academic Publishers.

Yin, R. and Campbell, D. T. (2002). *Case Study Research*. Thousand Oaks: Sage Publications Inc.

Zelkowitz, M. V. and Wallace, D. R. (1998). *Experimental Models for Validating Technology.* IEEE Computer **31**(5): 23-31.

Zook, C. (2010). *Profit from the core: a return to growth in turbulent times*. Boston: Bain & Company, Inc.

Appendix A

Study 1 – Adoption of Evo

Data #1: Group interview (postmortem analysis) with 6 developers

A PMA is a group exercise where a homogenous group, in this case all developers from one of the development teams, is given the task of documenting positive and negative experiences related to a defined topic. In this case the topic was the software development process and the recent adoption of the agile software development method Evo. In short, the participants are asked to write short statements on post-it notes to explain positive experiences. Then each participant places each note on a whiteboard explaining the meaning to the rest of the group. After this, the group together rearranges the notes to group similar or related topics together. Finally the group classifies each group of notes by giving it a name. Relationships between groups can also be indicated. The same procedure is repeated for negative experiences, which usually gives input to improvement actions.

Having this map of experience, the participants can prioritize the groups of negative experience according to importance or severity etc. The highest prioritized topic(s) can then undergo a root-cause analysis (RCA). This is simply done by having the same group drawing an Ishikawa diagram (fishbone), which shows causal relationships leading to the problem or issue.

A PMA is a lightweight technique that establishes a first overview of the topic under investigation. This information is high-level and based on subjective experience/opinions. The strength of this method is that the results represent the consensus of the group. See (Dybå, Dingsøyr et al., 2004) for more details on these techniques.

Data #2: Guide for three semi structured interviews with customers

- Customer background information (briefly)
    - Personal
        - Name (person and company)
        - Briefly about your background/experience
        - Role(s) and responsibilities
        - CSoft experience level
    - Company
        - Type of business
        - Years as CSoft customer (Company)
        - Type of customer (ASP or server)
        - Type of CSoft-use
- Have you participated in previous projects and, if so, how?
- Describe the CSoft 9.0 project
    - How did you, as a representative from the company, get appointed?
    - Describe the project start

    - Describe your every-day routines in the project.
        - Activities/contact with CSoft
        - Responsibilities/roles
        - Other…
    - What happened if you, personally, couldn't respond?
- If not mentioned before: how did you express your requirements and needs, and what do you think of it?
- Did CSoft respond to your feedback?
    - Can you see that your contribution has affected the product?
    - Was your contribution to the project worth the effort?
    - In general: was this a good process?
- Three most negative aspects of the project?
- Three most positive aspects of the project?
- How do you, internally in your company, determine requirements and response to CSoft?
    - How do you function as a representative for your company?
- How do you compare your experiences with this project to previous projects (if you have any such relevant experience)?
- Did you get any knowledge about the practical process prior to, or as part of, the project?
- How and why did you get involved in the project?
- Would you participate in a new project if asked by CSoft and do you have any improvement suggestions to the process?
- Are there other aspects that you feel are important to the study?

Data #3: Guide for five semi structured interviews with PMT members

- Personal
    - Name
    - Role
    - Role in PMT
    - Years in CSoft
    - Years in PMT
    - EVO knowledge/training?
    - How well do you know EVO?
    - How did you get this knowledge?
- About PMT
    - Could you describe PMT briefly, with respect to function, tasks, responsibilities and interaction with other parts of CSoft and interaction with customers and users?
    - How do you get customers' requirements and requests for features?
    - How do you document requirements and requests for features?
    - Do you interact with customers and if so, how?
    - How can you assure that you get the right picture of the customer's requirements and requests for features, both the explicit and the tacit/unexpressed?
    - How do you interact with the development project?
- About EVO and the role of PMT, seen from a PMT perspective
    - How do you interact with customer?
    - How do you interact with the development projects?
    - What are the most negative aspects of this?

    - What are the most positive aspects of this?
- Expectations to EVO? (before project start)
    - From a PMT perspective?
    - Generally?
- Positive effects of EVO, seen from a PMT perspective
    - What are the most positive effects of EVO, so far.
- Negative effects of EVO, seen from a PMT perspective
    - What are the most negative effects of EVO, so far?
- Product quality; how does EVO affect the product quality?
- About the realism in being a surrogate user
    - How realistic is it to represent the users/customers?
    - Have you any suggestions to improve this function?
- Other issues
    - In this context, have you any other information of importance.

Study 2 – Agile SPL

Data #4: Guide for one semi structured interview with the CTO

- The architecture proper
    - Can you name key characteristics prescribed by the CSoft architecture?
        - General principles (e.g. components, Service Oriented Architecture…)
        - Patterns (e.g. layering, client-server, facade etc.)
        - Are the server-version and the ASP-version of CSoft architecturally identical?
    - How is the architecture proper represented/maintained? (e.g. documents, models supported by tools…)
        - Are developers routinely trained, briefed or simply expected to "teach themselves" the architecture on a need-to-know basis?
    - What are the oldest "explicit" architecture elements in the current architecture? (and how old are they?)
    - Can you roughly estimate the size of the CSoft code today? What is the expected growth rate?
- Designing and maintaining the architecture
    - How was the architecture of CSoft first designed?
    - What are the main architectural drivers in the current generation of CSoft products? Name 2-3 of them (for example: modularization, supporting multiple product variants, performance, maintainability, testability…)
    - Is the architecture of CSoft adapted/designed/constructed to suit the agile development approach Evo?
        - In particular: Is it your opinion that the introduction of Evo increased or decreased the need to pay attention to architectural issues?
    - Name some examples of events/situations/business opportunities that have caused major new revisions to the CSoft architecture
        - Do you see different stimuli causing architecture revisions now, using an agile software process, compared to before, when using a sequential process?

    - Some years ago, the CSoft architecture was “modernized” as it was ported to the .Net platform. Which changes did you do to CSoft? Did you add “something” to support customization and potential future extensions?
    - How is the architecture of CSoft maintained?
        - Which stakeholders are involved and what are their responsibilities?
        - CTO: You interact with the Evo-process once per iteration to check decisions that affects the overall structure/architecture of the product; what guidelines do you use when doing so? (For example: roadmaps, overall plans, design guidelines etc.)
        - Do you have any examples of conflicts and how they were resolved?
        - A formal or informal process?
        - How do you deal with conflicts between short-term project/solution objectives, CSoft architecture, and product roadmaps?
        - We have heard so-called “interim releases” of CSoft being mentioned – are they mainly motivated by the need to target platform/architecture-oriented work?
        - Please describe the influence of the product roadmaps with respect to the maintenance of the CSoft architecture
    - Have you ever been forced to say no to a business opportunity because of too large divergence from CSoft architecture?

Data #5: Guide for one semi structured interview with PMT manager

- Product roadmaps
    - Can you describe a product roadmap?
    - What is the main objective(s) of a product roadmap?
    - Who ”owns” the product roadmap?
    - How does guidelines in a roadmap (long term) relate to project-level requirements (short term)
    - Are the various solutions included in the roadmap?
    - Do you have an “architecture roadmap”?
- Product planning in general
    - How is product planning done?
        - Formal/informal process?
        - Which stakeholders are involved?
        - How do you capture future needs and how do these get into development projects?
            - What are the main sources of ideas for future development plans? In your experience, has this changed as a result of using Evo?
            - Do you see a difference in the kind of ideas stemming from different sources?
            - If yes, do you use this consciously to direct a “balance” in requirements towards the product?
        - Do you consciously deal with the balance between short-term and long-term (architectural) issues during product planning?
        - How are product roadmaps used in projects?

        - What is the typical lifecycle of a road map? Are there different phases of product planning?
    - Which events/situations/occurrences may trigger major rewriting of road maps?
    - Apart from the problem of predicting the future, what is the most difficult aspect of product planning?
    - In your opinion; is the "nature" of product planning different in agile software development compared to traditional, plan-based/sequential development? If so, how?
    - Do you see particular challenges/critical factors stemming from agile development that affects product planning?
- Product planning with multiple solutions
    - What are the most important effects on product planning caused by offering multiple solutions?
    - How do you define the scope of CSoft? When/how do you extend the scope?
- Working with product plans
    - Who are the main users of product plans and/or product roadmaps?
    - How do the development projects use the roadmaps?
    - Are roadmaps affecting how you select and communicate with customers being engaged in development projects?
- Product planning and software architecture
    - Does (expected) future needs affect product architecture? Are parts of the solution (components) pre-developed?
    - What is your opinion of the architecture of CSoft with respect to being an enabler or inhibiter of innovation? Please elaborate.
- Misc
    - How many versions of CSoft exist?
        - Only one version alive (except several older ones in use in the market)?
    - A bit about the five standard configurations:
        - We assume that the close engagement of selected customers provides a rich source for detailed feedback and requirements to CSoft functionality. Is it fair to say that this primarily address user-near parts of the total system (e.g. the GUI-layer?)
        - What type of directions do you use to develop and improve other (non-visual) parts of CSoft?
    - When extending CSoft (new features/modules/etc.); what types of input do you use? Custom development, internal decisions (based on market analysis?), others?
    - What role does feedback to support play in developing roadmaps and in development projects?

Study 3 – Software entropy

Data #8: Guide for one semi-structured interview with a team leader

- How does the team know what do develop?
    - How does the developers get to know requirements and specifications?

- How do you test and verify new and improved code?
- How do you and your team contribute in the planning of the next release and ahead?
- How often do you speak with users and customers?
- How do you do this?
- Who else are you in touch with?
- How much of your work can be described as refactoring or improvement of architecture and structure?
- Which challenges do you have with respect to improving the systems architecture and structure?
- How and by whom are decisions of such work made?
- We have learned that the structure in CSoft has some weaknesses – how does this affect your job?
- What is the largest change with respect to Evo as compared with the old process?

Data #9: Guide for one semi structured interview with one of the developers

- Can you please describe what your job is about?
- Do you and the team contribute in the planning of new releases (roadmaps and plans)?
- In which ways do you have contact with external stakeholders during development? How does this affect the job you do?
- What is the best about the way you work today and which are the biggest challenges?
- We have learned that the structure in CSoft has some weaknesses – how does this affect your job?
- What are your ideas to improve the situation?

Study 4 – Software ecosystems

Data #11: Guide for one interview with the R&D manager

- [In this interview I would like to focus on how you communicate with the market and with your customers]
- Would you say that the roadmap is the start of requirements specification?
- What is new about your process as compared to before?
- When developing roadmaps, who do you talk with, which type of information do you get and how are decisions made?
- Could you tell a bit about the annual customer panel meeting?
- Who are your major competitors in the market?
- Can you tell a bit about your pricing policy?
- Are the development teams co-located or distributed?
- Do you still practice the green-week concept?

  How do you go about verifying the deliveries from the iterations?
- How does a stakeholder provide feedback?
- Do you have an active software process initiative at the time?

Data #12: Guide for one interview with the manager of professional services

- I would like to hear a bit about your role at CSoft, your tasks and the area of responsibility you have.
- Do you (Professional Services) participate in the development of roadmaps, which is managed by the PSG?
- Could you tell a bit about the department you run?
- I have recently observed one of your customer review meetings; do you communicate with the development teams in a similar fashion?
- Could you explain what the 'quality center' is?
- Do you/how do you participate in the development of roadmaps?
- Who operates your solution?
- How do you respond when customers report bugs or errors?

Data #13: Guide for one interview with the PSG manager

- About you
    - Roles, responsibility and tasks?
- About PSG
    - Who is part of PSG?
    - What is the role of PSG?
    - How does the PSG differ from PMT?
    - What was the rationale behind establishing the PSG?
- About customer engagement
    - What is the rationale of involving customers?
    - How does customers contribute in strategic planning?
    - How does customers contribute in product development (Evo)?
    - How do you motivate customers to participate?
    - How do you identify/select customers to participate?
    - How does the ideal customer (for participation) look like?
    - Engagement of third parties?
- Modes of involvement
    - What channels/practices do you use for engaging customers?
        - Operational and strategic user groups
        - Sales/market
        - Forums
        - Others?
- Third parties
    - How do you work with third parties?
- Innovation
    - How does new ideas/solutions come up?
    - Do you pilot ideas, make mock-ups, demos etc.? How do you evaluate these?
- Connection with R&D
    - In strategic planning
    - During development
- Roadmap
    - What is the role of product roadmaps in strategic planning and in R&D?
    - How is the process of developing roadmaps?
        - Who participates?
        - How do PSG contribute in developing roadmaps?

Data #14: Guide for three interviews with PSG members

We used the same guide as for the interview with the PSG manager (Data #13, above).

Data #15: Guide for one interview with a Technical Account manager

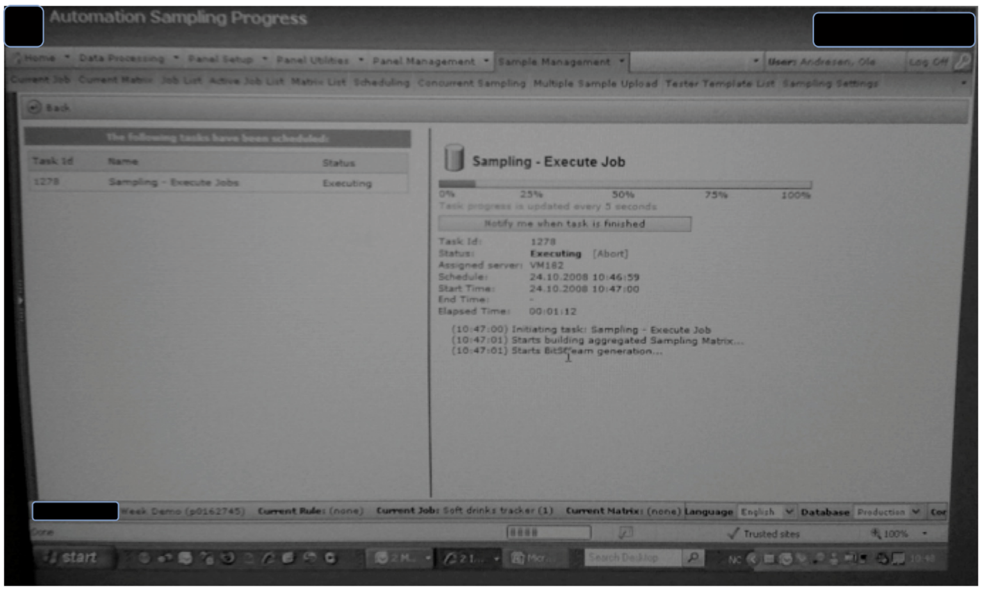
- About you
    - Roles, responsibility and tasks?
- About TAM
- Are you mostly involved into sales and dealing with prospects or do you do any kind of support?
- Do you participate in any way in making roadmaps?
- Do you meet external stakeholders in web conferences like the development teams do?
- Having gained some experience with the Evo process; what do you think of it?

Data #17: Observation of the 2008 product conference

The author participated at the whole event (three days). Selected sessions were observed and notes were made. Relevant presentations were collected. Relevant discussions in breaks and at social events were also documented.

Data #18: Observation of review meeting

A customer review meeting was observed via a web-meeting tool. The conversation in the meeting was recorded and later transcribed. Screenshots were taken throughout the presentation and demonstration. One example:


[redacted] Automation Sampling Progress [redacted]

The following tasks have been scheduled:

Task Id	Name	Status
1278	Sampling - Execute Jobs	Executing

Sampling - Execute Job

0% 25% 50% 75% 100%

Task progress is updated every 5 seconds

Notify me when task is finished

Task Id:	1278
Status:	**Executing** [Abort]
Assigned server:	VM182
Schedule:	24.10.2008 10:46:59
Start Time:	24.10.2008 10:47:00
End Time:	-
Elapsed Time:	00:01:12

```
(10:47:00) Initiating task: Sampling - Execute Job
(10:47:01) Starts building aggregated Sampling Matrix...
(10:47:01) Starts BitStream generation...
```

[redacted] Week Demo (p0162745) **Current Rule:** (none) **Current Job:** Soft drinks tracker (1) **Current Matrix:** (none) **Language** English **Database** Production

PART II – selected publications

Paper 1

Hanssen, G. K. and Fægri, T. E. (2006). Agile Customer Engagement: a Longitudinal Qualitative Case Study. In proceedings of 5th International Symposium on Empirical Software Engineering (ISESE'06), Rio de Janeiro, Brazil, 21-22 September. 164-173

Paper 2

Fægri, T. E. and Hanssen, G. K. (2007). Collaboration and Process Fragility in Evolutionarily Product Development. IEEE Software 24(3): 96-104.

Paper 3

Hanssen, G. K. and Fægri, T. E. (2008). Process Fusion - Agile Product Line Engineering: an Industrial Case Study. Journal of Systems and Software 81: 843-854.

Available online at www.sciencedirect.com

ScienceDirect

The Journal of Systems and Software 81 (2008) 843–854

The Journal of Systems and Software

www.elsevier.com/locate/jss

Process fusion: An industrial case study on agile software product line engineering

Geir K. Hanssen [a,b,*], Tor E. Fægri [a]

[a] *SINTEF ICT, NO-7465 Trondheim, Norway*

[b] *Norwegian University of Science and Technology, Department of Computer and Information Science, NO-7491 Trondheim, Norway*

Received 27 March 2007; received in revised form 19 July 2007; accepted 10 October 2007
Available online 7 November 2007

Abstract

This paper presents a case study of a software product company that has successfully integrated practices from software product line engineering and agile software development. We show how practices from the two fields support the company's strategic and tactical ambitions, respectively. We also discuss how the company integrates strategic, tactical and operational processes to optimize collaboration and consequently improve its ability to meet market needs, opportunities and challenges. The findings from this study are relevant to software product companies seeking ways to balance agility and product management. The findings also contribute to research on industrializing software engineering.

Keywords: Software product development; Software product management; Software product line engineering; Agile software development

1. Introduction

Software engineering is a complex task that involves fast-paced product development, competitive market conditions, knowledge management, organizational factors, rapidly evolving technologies, etc. As a means of handling this complexity, establishing control and using resources efficiently, development organizations normally adopt a method that defines how their software engineering activities should be carried out within the given context. A method may cover a broad range of issues; typically estimation, design, and development, among others. A great many methodologies are available and they vary a lot; from strict plan-based approaches to agile approaches and any variant in between. Regardless of the method being used, most software projects strive to reach a balance between three basic goals: satisfactory software quality (scope), the right cost and timely delivery. Attempting to satisfy these requirements causes further complications to arise, particularly with respect to long-term product management issues.

This paper discusses two popular development approaches; software product line engineering (SPLE) and agile software development (ASD). It describes how these have been combined to improve the ability to achieve the three basic goals. These two approaches can, in their most radical forms, be placed in each end of a plan-based/agile spectrum (Boehm, 2002). The former is based on planning and preparations for efficient software development based on rapid construction by assembling predeveloped assets, while the latter aims at efficient change response instead of extensive up-front planning. They may correspondingly be categorized as proactive and reactive approaches to software engineering. The principles of SPLE have been in use for a long time and industrial experience shows that the approach has substantial advantages when it comes to cost-efficient development, product quality and the ability

* Corresponding author. Address: SINTEF ICT, NO-7465 Trondheim, Norway.
E-mail addresses: ghanssen@sintef.no (G.K. Hanssen), tor.e.fegri@sintef.no (T.E. Fægri).

to deliver on time (Birk et al., 2003; Linden, 2002; Bosch, 2000; Clements and Northrop, 2002). ASD is a more recent trend within the software engineering community and has attracted wide interest in both industry and academia. Initial evidence from the results of ongoing research suggests that using ASD is advantageous, given the right context (Boehm, 2002; Erickson et al., 2005). Both approaches have the same overall objective; that of improving software development efficiency.

Recently, there has been interest in investigating whether, and if so how, SPLE and ASD can be combined to complement each other (Carbon et al., 2006; Tian and Cooper, 2006), but little empirical research on the topic has been conducted. In this paper, we present our findings from a case study of a medium-sized, Norwegian software product company called CompNN (real name made anonymous). CompNN has used and matured the ASD method Evo, short for *Evolutionary Project Management* (Gilb, 2005) for three years, covering four releases of their main product.

Our study is based on a well established industry–researcher relationship that began several years ago. The study expands upon a previous one, in which we investigated the introduction and preliminary effects of Evo (Hanssen and Fægri, 2006). This previous study gave us insights into several important aspects of the use of an agile process in a product development context, with respect to benefits, problems and prerequisites.

The aim of the study reported herein was to describe and analyze an industrial case to understand how SPLE and ASD can be combined and to clarify associated costs and gains.

Next, we provide an overview of SPLE and ASD and explain how they both relate to, and conflict with, each other (Section 2). Then we explain the context of the case study (Section 3). Then, we present the study method (Section 4). Subsequently, we present the findings from the study (Section 5), discuss the results (Section 6) and offer our conclusions (Section 7).

2. State of the practice

SPLE is an approach to software engineering in which a set of products is built on a common set of core assets (Bosch, 2000; Clements and Northrop, 2002). SPLE draws upon principles similar to those that have been used for decades in most other industries, in which production includes the assembly of prefabricated parts. SPLE is claimed to have numerous benefits: large-scale productivity gains, decreased time-to-market, increased product quality, increased customer satisfaction, more efficient use of human resources, and improved ability to apply mass customization, maintain market presence and sustain unprecedented growth. However, achieving these claimed benefits requires careful attention to a wide range of issues (Linden, 2002; Knauber et al., 2000; Käkölä and Dueñas, 2006).

The basic assumption underpinning SPLE's claimed advantages is that there is sufficient commonality among multiple products to justify investment in their systematic reuse. Reusable assets, of various kinds, are said to constitute the *product line platform*. Examples include requirements, architectures, designs, code and test cases. Domain engineering consists of the set of activities necessary to define, extend and maintain the platform. The platform, the key to an SPLE approach, bears the cost of domain engineering, which can, presumably, be amortized across multiple, derived products.[1]

Multiple preconditions must be satisfied if an SPLE approach is to be profitable, but the most fundamental of these is that the market for the products be predicted accurately. If prediction of market trends is inaccurate, SPLE will *inhibit* the business potential of the organization by enforcing wasteful effort related to adherence to reference architectures, organizational overhead, reuse strategies, etc., essentially slowing down the software production. In addition, product management is more complicated due to complex dependencies between product variants and the scope of the platform (Bosch, 2006; Helferich et al., 2006). SPLE has an influence on most practices in software engineering in an organization. Thus, SPLE activities must be based on the strategic ambitions of the organization and, in the service of such ambitions, careful planning. SPLE demands long-term investment and effort. Nevertheless, given the right processes and a suitable product platform, new products can be derived faster, at lower cost, and with more predictable quality (Birk et al., 2003). SPLE has already become economically necessary for some companies (Linden, 2002).

There are more or less rigorous ways to apply SPLE practices. For example, product derivation can be implemented using proactive engineering or reactive engineering. In a proactive approach, the reusable assets are developed before the product is built. In a reactive approach, the product is built first and subsequently the reusable assets are recovered from the product and integrated back into the platform. Further, variability models are used to represent variation among the products that the platform should support, i.e. which variation defines the scope of the product line? A vast number of different techniques and supportive artifacts exist for doing SPLE, but their discussion lies beyond the scope of this paper. For further details, please refer to, e.g. (Bosch, 2000; Clements and Northrop, 2002). It is useful to consider SPLE as a spectrum of approaches to software engineering.[2] SPLE may have a wide impact on business, organization, process and technological factors (Sugumaran et al., 2006). Therefore, organizations that adopt SPLE must choose their own path by introducing a set of practices that suit the

[1] We say that a product is *derived* from the platform if a significant part of the product is reused from it.

[2] http://www.sei.cmu.edu/productlines/framework.html (accessed December 2006).

objectives and capabilities of the organization (Lorge Parnas and Clements, 1986). Naturally, as with any other innovation in software processes, SPLE practices should be reviewed and revised continually to ensure learning and improvement.

Agile software development (ASD) lies at the other end of the plan-based/agile spectrum. ASD focuses little on planning and expects limited predictability. Further, it does not address issues of reuse. Hence, it contrasts sharply with the plan-centric SPLE approach. While SPLE supports the strategic objectives of the organization, ASD primarily benefits tactical ambitions, for example, addressing the immediate demands of customers or targeting smaller-scale development efforts.

ASD is a common name for a set of defined methods for software development. Some of the best known and used are Extreme Programming (Beck, 2000) and Scrum (Schwaber and Beedle, 2001). Although these agile methods vary in focus and presentation, they are all based on a few simple guidelines for ASD, defined by the *Agile Manifesto*[3] which states four basic values: (1) *Individuals and interactions over processes and tools*; ASD emphasize self-organizing teams, close cooperation with the user(s) and informal and direct communication. (2) *Working software over comprehensive documentation*; the main goal, and main proof of progress, is working software, not models or demonstrators. Documentation is kept on a strictly need-to-have basis. (3) *Customer collaboration over contract negotiation*; the customer is given an active role in development and thus has a direct responsibility to state requirements and correspondingly to verify both incremental and final results. (4) *Responding to change over following a plan*; development is a dynamic and creative endeavor, so plans are kept on a minimum with the focus being on flexibility and response to change. In addition to these four values, the Agile Manifesto also defines a handful of principles that form rules for development. These again are used to form a set of practices in the various agile methods. Some methods emphasize practice and technique descriptions (such as Extreme Programming), while others place more emphasis on management activities (such as Scrum) (Abrahamsson et al., 2002).

As part of the interest in, and uptake of, agile methods in industry a variety of experiences have been reported. Most of these can be defined as simple, single-case studies, but there are some reports that have investigated more deeply and have contributed to a growing knowledge base on the costs and benefits of agile methods. The agile method Extreme Programming has attracted the most attention. Within this method, the practice of pair programming has been the most investigated by far (Erickson et al., 2005). In this regard, a recent large controlled experiment has clarified the conditions under which pairs of developers perform better than individuals and vice versa (Arisholm et al., 2007). Task complexity and level of experience were shown to influence the outcome. Another relatively well-investigated agile practice is test-driven development (TDD) (Müller and Hagner, 2002; George and Williams, 2004; Erdogmus and Morisio, 2005). The results are somewhat contradictory. Erdogmus and Morisio (2005) showed that TDD improves productivity but not quality, yet their results also indicated that TDD programmers achieve more consistent quality results. With respect to Extreme Programming as a whole, Erickson et al. (2005) has reviewed numerous studies on the method and found no clear evidence as to its benefits. A possible reason is that Extreme Programming, as along with other agile methods, is constituted by a set of diverse guidelines, each of which should be investigated individually.

SPLE and ASD have significant differences due to the fundamentally different challenges they are intended to solve. SPLE addresses longer-term strategic objectives related to life-cycle product management, whereas ASD addresses short-term tactical objectives, such as single projects developing one specific product. In practice, an organization will face both strategic and tactical challenges. Thus, it makes sense to investigate whether, and if so how, these two approaches can complement each other. The table below summarizes how the SPLE approach corresponds to the twelve principles of the Agile Manifesto (see Table 1).

This comparison shows that ASD and SPLE have few conflicts. They are virtually independent of each other. On most counts, they possess complimentary properties (Carbon et al., 2006; Tian and Cooper, 2006). Whereas SPLE attempts to introduce large-scale cross-product economic benefits through reuse, ASD focuses on delivering single products with the best possible quality for the customer. Thus, there is, prima facie, no reason why these two approaches should not be combined. On the contrary, there is a clear motivation to combine them.

SPLE as such is agnostic regarding processes of software development. One can use product line engineering with different software process models. However, intuitively, there are a number of interesting issues to be addressed when considering the use of agile methods in a SPLE setting:

– SPLE advocates a conscious adherence to architecture. Software reuse beyond the most trivial level demands that numerous constraints and guidelines be respected regarding both the reused asset and the consumer of the asset. Agile methods have a reputation for paying little attention to architecture, and occasionally being downright damaging to it. Naturally, architecture erosion will result if constraints and guidelines are broken. Hence, it is useful to discuss the practices used by agile product companies that enable long-term architecture conformity and prosperity of the product line.
– Agile methods are (arguably) best suited to projects of modest to moderate size (Boehm and Turner, 2004). The different products (or variants) in a product line

[3] www.agilemanifesto.org (accessed November 2006).

Table 1
Comparing ASD and SPLE

Agile principle	SPLE correspondence
Priority on early and continuous delivery of valuable software	Dependent upon SPLE approach. A reactive model supports early delivery of valuable software (no particular bias with respect to continuous delivery)
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage	Foreseen changes covered by platform and variability models are cheap. Other changes become increasingly expensive closer to domain engineering artifacts as they may participate as elements in multiple products, and thus increase the cost of maintenance
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale	An SPLE approach does not reject or inhibit this principle. Product delivery is, nevertheless, somewhat slowed down compared to agile development, due to overhead in maintaining integrity with the product platform
Business people and developers must work together daily throughout the project	An SPLE approach neither rejects nor inhibits this practice
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done	At the product level, an SPLE approach neither rejects nor inhibits this practice. At the platform level, however, more formalism is required
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation	SPLE, due to increased technical and/or organizational complexity, introduces a need for more explicit, formal and disciplined communication
Working software is the primary measure of progress	SPLE incorporates significant value in the platform that is more difficult to measure However, for product development, an SPLE approach neither reject nor inhibits this principle
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely	SPLE promotes sustainable development at a larger scale
Continuous attention to technical excellence and good design enhances agility	SPLE platforms encode technical excellence and design to support rapid product derivation
Simplicity – the art of maximizing the amount of work not done – is – essential	The main motivation for SPLE is the same – but by using reuse as vehicle for eliminating work. In a reactive SPLE approach, the risk of doing unnecessary work is reduced
The best architectures, requirements, and designs emerge from self-organizing teams	The assumption of SPLE is that investments in requirements, architectures, and designs can be reused successfully across multiple products
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly	An SPLE approach neither rejects nor inhibits this practice. In fact, due to investments in the platform, most SPLE organization promotes experience gathering and learning

can be considered as semi-autonomous projects within the line. In this regard, SPLE may be considered as a way of partitioning the work of software product companies and subsequently assisting the use of agile methods in larger-scale development efforts.

How can potential benefits and challenges be managed in practice?

3. Study context

CompNN is a medium-sized Norwegian software company that develops, maintains and markets a product line called ProdNN (real name made anonymous). The products derived from the product line are aimed at the high-end market of market and customer surveys. CompNN has a wide customer base that includes some of the world's largest market research agencies. The company was established by three friends in 1996 and has since grown steadily, such that it now has about 90 employees and offices in Norway, the UK, Vietnam, and the USA. There has been a gradual shift from building custom-made applications to a product line. Today, licenses for the ProdNN products constitute most of the revenue.

The bulk of research and development is done at the main office in Norway supported by developers in Vietnam. The other offices are mainly devoted to marketing, sales, and customer support. However, as we will show later, they have a clear role to play in providing important input to the development process. Physical presence near the largest markets abroad is important both to serve customers and to include them in the development of the product.

ProdNN can be defined as a single product, but there are many ways to use it. The system is modular. It has five main modules (with numerous submodules): *authoring* to assist the planning of surveys, *panel* to support panel surveys, a *survey engine* that serves most use cases, *reporting* to produce survey reports, and *data transfer* to feed the database for analysis. The use of these modules varies according to the use case. Some modules are central and always in use, while the use of the others depends on the situation. As CompNN's customers vary greatly in size and operation, the ProdNN tool is built to be customizable. CompNN operates with a set of predefined configurations for the most common use cases, but there is also built-in support for detailed customization to support more variants. However, in some cases, a customer has needs that go beyond the functionality available in the product platform. In these cases, CompNN may provide a custom-developed solution for the customer, with the proviso that the result may be included in the platform. If this is not acceptable to the customer CompNN will deny the request. This strategy of restrictive scoping is chosen to avoid too heavy a burden with respect to single-case development, because that would hamper the continuous development of the platform.

The adoption of Evo was a deliberate choice to move on from an inefficient waterfall-like process (Fægri and Hanssen, 2007). In this case, SPLE was not introduced deliberately, as prescribed by common guidelines, e.g. the Software Engineering Institute's guidelines.[4] However, this company's development process clearly coincides with some of the fundamental principles of SPLE. Thus, we were able to use this case to understand how the company has combined SPLE and ASD and to identify the experienced effects of the combination. We believe that this is valuable input to the process of shaping an agile product line engineering practice, which is in its early stages.

Several roles cooperate at different levels in the development of the product platform. At a tactical level, the developers in the R&D department handle the software development, guided by the Evo process. The developers are organized in small teams, each of which has three to five persons. To ensure that focus is maintained, each team is usually dedicated to the development of just one of the main modules in the platform. In addition, each team may also cooperate directly with a selected stakeholder. The R&D department has a Chief Technical Officer (CTO), who is responsible for the platform architecture and for ensuring technical alignment with the business domain, and who oversees overall product planning. CompNN puts great emphasis on new and improved qualities which leads to a high pace in the development projects and an unsatisfactory error density in released products. This issue is now being addressed by the adoption of test-driven development.

To handle the diverse processes towards the market at both strategic and tactical levels, CompNN has a Product Management Team (PMT). This function was established at the same time as Evo was introduced, in 2004. The PMT has six members, one of whom acts as the PMT manager. The PMT plays an important role at several levels. They are responsible for marketing initiatives, sales, and key account management. The product roadmap, together with supporting business cases, are the main assets developed by the PMT. Two main channels of information are used to develop these. First, there is the PMT member's knowledge of the domain, market and specific customers. Secondly, the ProdNN Advisory Board, was established recently. This is a group of the most important customers, who provide their ideas and state their requirements for future development. The members of this forum are CEOs (Chief Executive Officer), board members and other representatives from top management.

[4] http://www.sei.cmu.edu/productlines/adopting_spl.html (accessed December 2006).

One of the core principles in the adopted Evo process is the close cooperation with selected stakeholders (selected representatives of important requirements). When a product roadmap has been approved by the executive management, the PMT appoints stakeholders, who agree to participate actively in the release projects. Usually, one stakeholder per project is appointed, which again places the focus on one of the modules or a well-defined feature (Fægri and Hanssen, 2007). The criterion for appointing a stakeholder is that the development focus or goal of the project addresses a module or feature in which the stakeholder has a special interest, which in turn provides sufficient motivation for investing resources in development. In some cases, internal stakeholders are appointed, typically when the project addresses issues that are not directly visible to the end users. External stakeholders are not given any compensation for participating. The sole reason for cooperating is the opportunity to affect and direct the development.

The roles and their participation in product development activities are summarized in Fig. 1. Their interplay is further elaborated in Section 5.

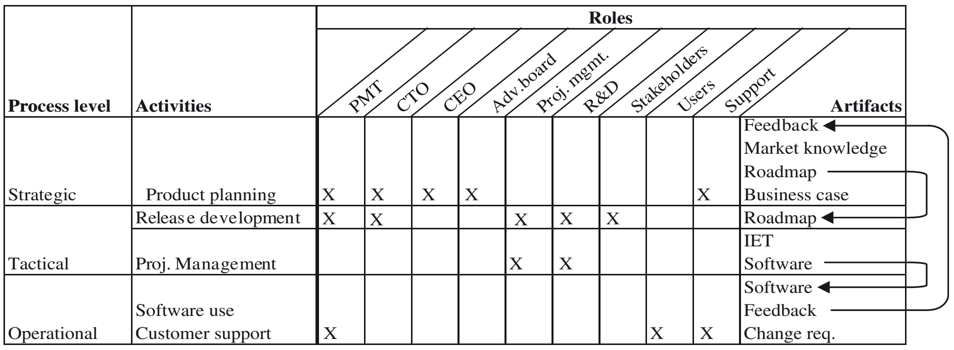
4. Study method

4.1. Rationale

As this study sought to investigate the diverse and complex topic of the fusion of product-line engineering and agile software development, which touches on a variety of factors, such as technology, architecture, process, market development, innovation, and management and organizational issues, we chose a qualitative approach. Our aim was to present a broad view and to explain how this fusion has worked in practice in actual industry, with a view to understanding how SPLE and ASD can be adopted and used to complement each other. We believed that we needed to understand the practicalities and opportunities of this fusion better before we can plan more specific studies on, for example, the effect on time, cost and quality. Thus, we performed a single-case study (Yin and Campbell, 2002) to obtain detailed knowledge about a complete product development organization to be able to understand how internal and external roles interplay at various levels.

4.2. Data collection

Our main source of data collection was a series of interviews. We used existing interviews from a previous study (Hanssen and Fægri, 2006). There were nine interviews in total, which three roles: three customers (different companies) from the development of a single release (three interviews), all members of a product management team (five interviews) and a group of six developers (group interview) where all of these had recent experience from Evo-projects. To acquire a deeper understanding of the strategic and



Process level	Activities	PMT	CTO	CEO	Adv.board	Proj. mgmt.	R&D	Stakeholders	Users	Support	Artifacts
Strategic	Product planning	X	X	X	X					X	Feedback Market knowledge Roadmap Business case
Tactical	Release development	X	X			X	X	X			Roadmap
	Proj. Management					X	X				IET Software
Operational	Software use Customer support	X							X	X	Software Feedback Change req.

Fig. 1. Roles, activities and artifacts.

tactical processes and how the development process and the product line system architecture relate to each other, we extended the interview material with thorough semi structured interviews with the PMT manager and the CTO. These two roles are central to the planning and management of the product development process. We defined an interview guide for each of the PMT manager and CTO, but the interviews were semi-structured so that additional information, which was not originally covered in the interview guides, could be acquired. All interviews except the group interview with the developers were done via telephone. Furthermore, all interviews were recorded using a Dictaphone. In sum, we thus base our case study on eleven interviews covering all roles involved in the product line engineering and the applied agile process.

As a secondary data source, we used documents such as product roadmaps and business cases, which are important assets in long- and short-term development of the product line. These documents were used to understand the overall practice of software development and market-related activities and how these relate to each other.

4.3. Data analysis

All recorded interviews were transcribed to form a basis for a thorough and detailed textual analysis. In cases of interviews that covered more than one individual per role, for example the members of the product management team, we analyzed the material according to the principles of constant comparison (Seaman, 1999) using the NVivo™ tool for textual encoding. The transcribed interviews and the documents were used to make a description of the product development (see Section 5).

4.4. Bias and limitations

Several factors restrict the credibility and generalizability of the study. (1) We collected and analyzed data from only one company. However, if we had involved more company cases, the study would have been much more difficult to perform. In addition, we only know of, and have access to, one company that has adopted and combined both SPLE and ASD practices; the one that we studied. Despite this limitation, a reader may, by considering the context information, be able to identify experiences that may be transferred to another software development organization. (2) There is a threat with respect to the completeness of our study. We did not interview all parties involved in the total process of developing the product platform, but we believe we have selected the most central; those that could give us a good basis for reflection. To strengthen the study further, we incorporated the analysis of important documentation.

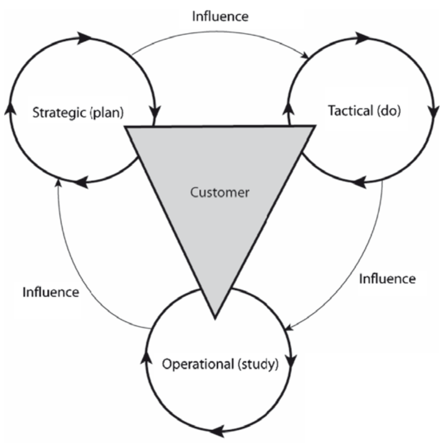
5. Findings – a holistic view of software engineering at CompNN

We now describe, from a holistic point of view, how CompNN develops and maintains their product platform. We have identified three distinct software processes and describe how they interact. Then we describe the product platform architecture and related services followed by a description of how the organization has shifted to this new development regime.

5.1. A symbiosis of three interacting customer-centric software processes

Organizations of some size need well defined structure and operational formalism to ensure that important functions are implemented and maintained in a professional manner. CompNN has matured their approach to software engineering over the last 10 years. As new challenges have appeared, CompNN has responded by making different adjustments and innovations in their processes. Because CompNN operates in a highly competitive market, they must continually look for opportunities to improve the way they do business. Thus, the description presented here is merely a snapshot of how things are done currently.

The Fig. 2 illustrates how CompNN, through a symbiosis of three different software processes, connects software development and customer collaboration. The embedded



	Stratetic (1-2yr horizon)	**Tactical (2wk-4mnth horizon)**	**Operational (day-to-day operations)**
Function	Strategic, technological innovation	Tactical (reactive) innovation	Systematize experience
Process elements	SPLE practices	Evo, IET	Request driven, prioritization
Deliverables	Product roadmaps, business cases	Software	
Technical focus	ProdNN software platform	Release projects	Fixes, updates, upgrades
Key roles	Management, PMT, CTO	R&D, PMT, CTO	Support, QA
Customer collaboration	Advisory Board	Key stakeholders	End users, ASP operations personel

Fig. 2. A customer-centric product development process.

table highlights the various issues of particular attention in the three processes. As a result, CompNN is able to embrace the market all the way down to individual users, and thus improve their ability to respond and interact with their customers. We believe this works well because the three processes serve different but important needs:

Strategic (long-term, SPLE): Evolution at the SPLE level addresses the need to implement long-term strategic plans. SPLE supports CompNN's need to introduce tactical innovation, which is fuelled, for example, by innovations in technology, changes in the business models, or new business opportunities. ProdNN is a product platform that contains a set of building blocks that can be tailored to provide different product offerings (ProdNN solutions). This platform can be considered as the vehicle used by the CTO and PMT to implement long-term visions. More often than not, these innovations are costly in terms of effort and risk. Thus, they should be planned and introduced with great care, in order to exploit the organization's development capability. SPLE-style development inherently supports this ambition.

Tactical (medium-term, ASD): Evolution at the project level (note that a release typically includes 4–6 projects) addresses the need to embrace customers and listen to their more down-to-earth needs. Close cooperation ensures that a customer's ideas and experience with the software is translated into what we may call tactical innovation, i.e. reactive innovation that seeks to polish, improve, adjust or otherwise make small to moderate adjustments to the product. As the customer's requirements mature during the project, ASD-style development with moderate-size projects works well to accomplish this aim.

Operational (short-term, day-to-day operations): Customers rely on the software to help them to implement vital business functions in their own organizations. Thus, it is of great importance that CompNN is able to sustain a good level of satisfaction with the software in its day-to-day use. During operation, the products are exposed to more users, which exposure increases the potential amount of feedback to the innovations implemented at the strategic and tactical levels.

These three processes are concurrent. They overlap in terms of activities, people and artifacts. All three embrace the customers and give CompNN the opportunity to build strong, cross-level, collaborative relationships with them.

5.1.1. The strategic process elaborated

The market process is at a strategic level and provides vital input to the work on the biannual releases and the development projects. This strategic work is operated by the product management team (PMT), who are engaged both in product development and other general market-oriented activities, such as sales, marketing, and key account management. Thus, they act as a link between market (strategic) and development (tactical) activities.

The central activity in the market process is the development of a roadmap. This is a strategic document where needs, ideas and opportunities for future development are documented. In practice, the roadmap is a set of PowerPoint slides that defines growth opportunities, main challenges and strategies. Each issue is described at a high level, to prevent the view of features and functionality from becoming too detailed. On the basis of the suggested direction for developing the product platform, the PMT also indicates how existing or new resources should be used; for example, a need to extend the R&D department. The roadmap also defines what is called 'Commitments and must haves', which is a high-level list of improvements and extensions that presumably, will meet the identified opportunities and challenges. To operationalize the roadmap, the PMT also suggests how to set up a handful of development projects to reach the defined commitments and must haves. This includes the development focus per project and staffing (number of developers per project). To establish a long-term view on development, e.g. to expose resource needs, this part of the roadmap characterizes the release projects for both upcoming and subsequent releases, though subsequent releases are characterized more loosely. The last part of the roadmap looks even further into the future and defines potential new areas for the following two years. This overview is developed further in the following year's roadmap.

The PMT develops a new roadmap once a year. A proposal is given to executive management in the autumn for approval, and is then used as a guide to decide formally which parts of the product platform will be addressed in the upcoming releases. The goal descriptions in the roadmap are deliberately kept at a high-level. Thus, the possibility is left open that projects may suggest refinements to the platform to facilitate achieving the goals. This is important, because the development projects will utilize core agile principles, such as short iterations and close customer engagement as defined by the Evo development process, to capture detailed requirements for finding the best possible practical solution for achieving the high-level goals defined in the roadmap.

The roadmap relates to a set of business cases, which are documents that illustrate the assumed effect of achieving the high-level goals. A business case usually addresses one of the modules in the product platform to obtain a manageable focus and follows a standard structure: it defines the stakeholders, the scope of the project (what is to be done), and its rationale. A business case also contains a discussion on the return on investment, identified risks and threats, and associated costs.

To provide management with the best possible foundation for making their decisions what is proposed in the roadmap, it also contains descriptions of cases that are not a part of it, thus illustrating trade-offs that must be made by executive management due to limitations of time and resources; the total quantity of ideas always exceeds the quantity that can be developed for one release. The PMT thus keep track of ideas that are excluded from the roadmap and these are revised each time a new roadmap is developed.

5.1.2. The tactical process elaborated

Gilb (2005) is the agile method that CompNN adopted at the tactical level, i.e. the level where software is developed. It is not as well-known as Extreme Programming (Beck, 2000) and Scrum (Schwaber and Beedle, 2001), yet it is clearly an agile method, because it is customer-centric and strongly emphasizes short iterations and the frequent delivery of working software. In Evo, the product evolves iteratively as new requirements are discovered, constructed into the code, and subsequently tested. Stakeholders (customers or internal) are expected to give feedback on each of the iterations of the product in order to guide the development of the next iteration. Finally, suites of integration tests are performed before the final system test. Evo is a rigorous process that draws upon principles from the risk-driven Spiral model (Boehm, 1988) but is ultimately driven by the highest possible stakeholder value in each iteration, rather than by risk reduction.

The core process of Evo is requirements management, which serves two purposes per iteration. Firstly, selected stakeholders evaluate the achievements in the previous iteration in terms of a hands-on test or some other type of review. This helps to ensure that progress is in line with stakeholders' expectations. Secondly, on the basis of stakeholders' feedback, it is decided what existing requirements need to be improved and what new requirements should be targeted for the next iteration. Requirements are not specified as functions or features, but as product *goals* that address real business concerns of the customer business concerns. Great emphasis is placed on identifying metrics and associating them with each goal. CompNN uses Impact Estimation Tables (IET) as its key project management tool. Each of the projects within a release has an IET, which is essentially a spreadsheet that lists goals vertically and project iterations horizontally. Each of the iterations is broken down into one or two solutions (design proposals) that are believed to address the goals. The structure helps to separate requirements from solutions. By documenting both requirements and project progress (measured results) per iteration, the IET becomes both a plan for, and historical record of, the project. Before release of the iteration is initiated (which initiation spans roughly two weeks), selected customers are invited to give input to product plans with initial goals for the release. During initiation, the most promising ways of addressing the initial goals are noted and laid out for a number of future iterations in the IET. CompNN's CTO explained: "This plan helps us ensure adherence to an overall technical strategy for product development". Evo at CompNN is a tightly scheduled process in which biweekly iterations follow a fixed schedule (called the 'Evo-week') that defines each role's responsibilities per day.

In addition to playing a vital role in the strategic process, the PMT also have responsibilities in the tactical process, in connection with the software development in the release projects. Because they have the best access to knowledge of the market and the customers (Keil and Carmel, 1995), the PMT are responsible for appointing customer stakeholders in the development projects and for maintaining the link between CompNN and these stakeholders. If a customer stakeholder leaves a project for any reason, it is vital that the PMT take action to appoint another stakeholder or to fill the gap in some other way.

During the second Evo-release, CompNN introduces an additional concept: the green week. This is a week dedicated to error correction and improvements, without engaging with customers directly.

Another great aid in the development process (while not being a part of Evo) is the Continuous Integration framework (CI). This is a set of well-integrated tools for code management, automated tests and builds that enable swift production of product versions ready for testing by stakeholders.

5.1.3. The operational process elaborated

The operational process consists of the day-to-day activities in connection with the customer's use of the products based on the product platform. As CompNN provides up to two releases per year, which surpasses the update frequency of most customers, there are several versions of the platform in use at any one time. In addition, there are many variants based on the product platform in use, all of which may be considered advanced applications. In sum, this means that there is great variance in the support requests to CompNN, which are handled by a dedicated support department. Depending on the situation, support may involve guidance in use of the product, scripting, programming, quality assurance of customers' use, or issues related to maintenance. Some requests also address errors and flaws in the software, even ideas for future development. Such requests are recorded in a support system and are used as input to the strategic process that is aimed at upcoming releases. In addition to assisting in the use of the software and recording issues that pertain to potential improvement, the support department develops user documentation.

5.2. Product platform architecture and services

The ProdNN architecture is deliberately kept simple to achieve flexibility. From a technical point of view, it is a traditional three-tiered application with a data layer, a business layer and a web-based (ASP) interface layer. The whole platform is based on the Microsoft.Net platform. Some old COM-components are present and constitute a legacy from the first years of shaping the platform. Recently, the R&D department did so-called interim releases, in which no external stakeholders were involved, and concerning which the only purpose was to port old code, fix errors and resolve pressing open issues. Now, these types of task are performed using free development resources in between the ordinary releases.

All end-user interaction with the software is done via a cross-platform web-based interface and two alternative delivery models; for sophisticated use or large organizations, CompNN offers ProdNN as a stand-alone server solution, which allows the customer to operate the server locally. For less demanding users, CompNN offers an ASP alternative, in which the application is hosted by CompNN.

From a functional point of view, the platform consists of five main modules (with submodules), each of which supports specific features or tasks in the survey process. Some modules are essential in all use cases, while others are optional and depend on the situation. Each module is designed to be configurable to achieve the benefit of developing one common component that can serve many situations of use. Variability in customers use is handled simply; a license file is defined for each customer, which file determines the modules that the customer can use. This solution is the same for both the local server and web-access variants. Working on the next release, development is usually organized such that one development team (defined as a project) is responsible for one of these modules. R&D will, in the future, establish dedicated teams for the modules, which teams will specialize in the features and the related sub domain. ProdNN may be extended by a customer or a provider of third-party solution providers by the use of a set of API's (offered as web services), there being one library for each of the five modules.

ProdNN is not provided as a single standard product, but is defined as a product platform that forms the basis for many variants. As a part of this paradigm, CompNN offers a set of services to assist each customer in deciding upon a variant that matches their specific needs in each case. In most cases, this is done through the 'customized solution' service, where the basic modules are composed and adjusted. This service may also include documentation, training, support and maintenance. Examples of customized solutions in use are help-desk quality feedback systems and claim management systems. In cases where a customer has requirements that are not supported by the product platform, CompNN offers the 'custom development' service which, besides customization, also involves the development of new features. This service may be offered if the new features under consideration are thought to be reusable in a future version of the product platform. It is an absolute requirement that the customer agrees to allow the new features to become available to other customers as part of the product platform. If this is not feasible for the customer, CompNN will not develop the features. This policy contributes to keeping the product platform within a manageable scope.

Due to the nature of the product platform, customers often desire to integrate it with other software systems. CompNN offers the 'system integration' service to support these cases. Typical systems integrated with ProdNN are CRM-systems, help-desk systems, and call-centre systems.

5.3. The shift to the new approach

From the start of the company, ten years ago, the development process matured from a more or less ad hoc type of process to a well-defined waterfall-inspired process. This was a non iterative process with emphasis on extensive planning and up-front design. Though this process helped them to structure the development of the product platform, it became apparent that market needs and trends were not being met to their satisfaction. Customers also started to raise questions regarding close collaboration, as the products became an important part of their core business. By chance, some representatives from the company met Tom Gilb at a workshop and became interested in the concept of iterative and evolutionary development. After an introduction, they decided to try Evo in the upcoming release. At the same time, they introduced the PMT function. These two changes were not originally planned together, but experience quickly showed that they fitted well together and formed the basis for the combination of SPLE and ASD (Johansen, 2005).

6. Discussion

The integrated software processes at CompNN support three key virtues of product development. (1) Technical excellence: an open and modular platform architecture implemented using industry-standard technology enables simple development and maintenance of the product line. (2) Market knowledge and relevance: the well organized, yet nimble strategic process provides adequate decision support for company management and guidance for the development projects. (3) Agility: the adoption of Evo, and agile principles, enables fast response to changes in stakeholder requirements and accurate delivery of desired features and qualities to the users. This process configuration has been developed and matured within the organization to provide a useful balance between discipline and agility. We believe this case is an example of how to build 'your home ground' (Boehm and Turner, 2004) for agile SPLE.

An important observation from our findings is that CompNN's integrated development process is not just an accidental collection of three processes. Rather, the three processes play distinct, supplemental and important roles at three different levels in CompNN's overall software development business. All three are lightweight to reduce waste, that is, unnecessary work (Imai, 1997). Taking a broad view of the strategic, tactical and operational processes, we see a complete process that has the same function and effects as the more general Deming improvement cycle (Deming, 2000). This is also known as the PDCA cycle, from its four main activities: *Plan* (plan how to satisfy improvement requirements); *Do* (accomplish planned actions); *Check* (monitor the actions and verify the outcome with respect to the planned effects); and *Act* (implement actions for improvement based on the acquired information). This improvement process, simple in principle, comes in many variants and is central in, for example, Total Quality Management (TQM) (Mathiassen et al., 2001). In CompNN's case, the Plan and Act steps coincide in the strategic process where business cases, technology trends and other high-level objectives are used to create the product roadmap. Both SPLE and ASD allow and encourage continuous learning. The fact that the cycles have different durations poses no noticeable problems: strategic planning is intrinsically a long-term process. The roadmap is a stable artefact during the development of two released versions. Thus, the tactical process can be managed using the roadmap as a guide to the priorities and selection of stakeholders in the Evo process. The released products may be amended with patches during their lifetime. However, this is not a problem for the support department in charge of the operative software process, because feedback from customers is associated with the patch level of their software. Indeed, the issuing of patches enables the support department to verify whether patches actually help solve the immediate problems that customers may experience. All feedback to the support department is available for the PMT when preparing the next roadmap. In this way, the three cycles feed each other with up-to-date, accurate knowledge from the customers.

It is worth noting though, that this approach has its costs; running the strategic process and working closely with stakeholders entails a considerable extra overhead, resources that could have been invested in development (as was the previous practice). Despite this drawback, CompNN holds the benefits of these additional activities to be more valuable than spending the majority of resources purely on development activities.

Arguably, the most important benefit of this process configuration is that it helps CompNN to exploit both strategic long-term ambitions for innovation and smaller-scale tactical innovations, such as refinement of the software to meet more detailed end-user requirements. In order to implement this scheme, the PMT plays a crucial role in mediating and facilitating processes (Fægri and Hanssen, 2007). Further, the process configuration is a potent foundation for further process innovations. As long as the PDCA cycle is maintained, any component in the configuration can be further refined to improve performance. If the company experience the expected growth, the modularization gives extra organizational flexibility. For example, introduction of separate roles for platform and product development within R&D can readily be supported because the objectives of strategic and tactical development are already defined and well established. Additionally, the explicit engagement of primarily external stakeholders in the tactical development process ensures that the company is able to supplement strategic planning by selecting stakeholders based on long-term as well as short-term interest. This is an extra benefit for use by the PMT.

The conscious engagement of market representatives is used as an important driver in the development process as a whole. It is interesting to see the differences in how this is done in practice at the strategic and tactical levels. During the development of roadmaps, knowledge obtained by PMT and CTO via any kind of market- or technology-oriented activities is exploited. In addition, key customer representatives influence the process directly through the ProdNN Advisory Board. Input is deliberately kept to a general level to establish a sufficiently wide scope. A first release is presented in detail, less detail is captured for the second, and only a rough outline is delivered for the 2-year vision, which allows for less prescriptive plans into the future. Having documented the main ambitions in the roadmap, customer representatives are given a much more detailed and direct role in the Evo projects. Selecting and involving the right stakeholders is critical however (Fægri and Hanssen, 2007). Engaged stakeholders provide requirements and feedback on a detailed level but it is the responsibility of the development projects to suggest practical solutions that will meet the quality goals stated by the engaged stakeholders. This close cooperation with a few selected stakeholders promotes valuable creativity (Hanssen and Fægri, 2006). If process innovation at CompNN had a more revolutionary style, replacing all existing practices with a pure and very formal SPLE approach, this potential could have been lost.

The present practice in CompNN was not defined and introduced from scratch as a major initiative for improving processes. However, the organization, the product platform, and the processes and their interplay, have each developed and matured over time. This has been a natural evolution, based on experience and needs (Dybå, 2000). Our overall impression from this case is that CompNN has been successful in combining and balancing practices from SPLE and ASD, essentially adopting those practices that are valuable to them, as a company.

Referring back to the figure, the strategic SPLE practices creates a fertile, controlled environment in which the tactical ASD practices can exploit the creative potential of the developers. Cockburn (2002) defines five dimensions to describe 'an agile home ground'. These are: *size* (number of personnel), *criticality* (loss due to impact of defects), *personnel* (level of software method understanding and use), *dynamism* (percent requirement change per month) and *culture* (percent thriving on chaos versus order). The dimensions size and criticality are not significantly affected by the SPLE/ASD synergy. Looking at the personnel dimension, we see that their SPLE practices, combined with the rather disciplined IET planning tool, give the organization a rigid framework for requirement specifications. However, due to the freedom of the solution concept in Evo, developers are still encouraged and motivated to creative problem solving. Thus, the discipline enforced by SPLE and IET enables agility and creativity. Considering the dynamism dimension, much of the long-term planning is performed within the strategic SPLE process, leaving a comfortable and controlled amount of dynamism to the individual projects. Last, regarding the culture dimension, we see that a high level of order does not inhibit agility, it enables it.

7. Conclusions

The aim of this study was to describe and analyze an industrial case to understand how SPLE and ASD can be combined and to clarify associated costs and gains.

We found that these two approaches, which at first sight may seem contradicting, in fact complement each other. SPLE and ASD work together, supporting strategic and tactical objectives, respectively. At CompNN, an operational process constitutes the experience-bearing link from the tactical to the strategic process, thus completing the improvement circle. We found that this, as a whole, constitutes a framework that balances discipline and agility (Boehm and Turner, 2004).

The overall process described and discussed here involves multiple disciplines, such as product planning, knowledge management, organizational aspects, and innovation. We believe that our conclusions constitute a valuable contribution to determining the practicalities and effects of agile SPLE, being an interesting approach to industrialized software engineering.

Acknowledgements

The authors would like to thank CompNN for their time and valuable inputs. We are also grateful to Chris Wright for proofreading. This work was done as part of the EVISOFT project funded by the Research Council of Norway under Grant 156701/220.

References

Abrahamsson, P. et al., 2002. Agile software development methods – Review and analysis. VTT Electronics, 112.

Arisholm, E. et al., 2007. Evaluating pair programming with respect to system complexity and programmer expertise. IEEE Transactions on Software Engineering 33 (2), 65–86.

Beck, K., 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley.

Birk, A. et al., 2003. Product line engineering, the state of the practice. Software, IEEE 20 (6), 52–60.

Boehm, B., 1988. A spiral model of software development and enhancement. IEEE Computer 21 (5), 61–72.

Boehm, B., 2002. Get ready for agile methods, with care. IEEE Computer 35 (1), 64–69.

Boehm, B., Turner, R., 2004. Balancing Agility and Discipline – A Guide for the Perplexed. Addison-Wesley, p. 266.

Bosch, J., 2000. Design and Use of Software Architectures – Adopting and Evolving a Product-Line Approach. Addison-Wesley.

Bosch, J., 2006. The challenges of broadening the scope of software product families. Communications of the ACM 49 (12), 41–44.

Carbon, R. et al., 2006. Integrating product line engineering and agile methods: flexible design up-front vs. incremental design. In: Workshop on Agile Product Line Engineering.

Clements, P.C., Northrop, L., 2002. Software Product Lines: Practices and Patterns. Addison-Wesley.

Cockburn, A., 2002. Agile Software Development. In: Cockburn, A. (Ed.), The Agile Software Development Series. Addison-Wesley.

Deming, W.E., 2000. Out of the Crisis. The MIT Press.

Dybå, T., 2000. Improvisation in small software organizations. IEEE Software 17 (5), 82–87.

Erdogmus, H., Morisio, M., 2005. On the effectiveness of the test-first approach to programming. IEEE Transactions on Software Engineering (31), 3.

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.

Fægri, T.E., Hanssen, G.K., 2007. Collaboration and process fragility in evolutionarily product development. IEEE Software 24 (3).

George, B., Williams, L., 2004. A structured experiment of test-driven development. Information and Software Technology 46 (5 SPEC ISS), 337–342.

Gilb, T., 2005. Competitive engineering: A handbook for systems engineering, requirements engineering, and software engineering using P language. Elsevier Butterworth-Heinemann, p. 480.

Hanssen, G.K., Fægri, T.E., 2006. Agile customer engagement: a longitudinal qualitative case study. In: International Symposium on Empirical Software Engineering (ISESE). Rio de Janeiro, Brazil.

Helferich, A., Schmid, K., Herzwurm, G., 2006. Product management for software product lines: an unsolved problem? Communications of the ACM 49 (12), 66–67.

Imai, M., 1997. Gemba Kaizen: A Commonsense Low-Cost Approach to Management. McGraw-Hill.

Johansen, T., 2005. Using evolutionary project management (Evo) to create faster, more userfriendly and more productive software. Experience report from FIRM AS. In: International Conference on Product Focused Software Process Improvement. Springer Verlag, Oulu, Finland.

Käkölä, T., Dueñas, J.C., 2006. Software Product Lines: Research Issues in Engineering and Management. Springer Verlag.

Keil, M., Carmel, E., 1995. Customer–developer links in software development. Communications of the ACM 38 (5), 33–44.

Knauber, P. et al., 2000. Applying product line concepts in small and medium-sized companies. Software, IEEE 17 (5), 88–95.

Linden, F.V.D., 2002. Software product families in Europe: The ESAPS and CAFÉ projects. In: IEEE Software, pp. 41–49.

Lorge Parnas, D., Clements, P.C., 1986. A rational design process: how and why to fake it. IEEE Transactions on Software Engineering 12 (2), 251–257.

Mathiassen, L., Ories-Heje, J., Ngwenyama, O., 2001. Improving software organizations: From principles to practice. In: The Agile Software Development Series. Addison-Wesley Professional, p. 368.

Müller, M., Hagner, O., 2002. Experiment about test-first programming. Software, IEE Proceedings 149 (5), 131–136.

Schwaber, K., Beedle, M., 2001. Agile Software Development with Scrum. Prentice Hall.

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

Sugumaran, V., Park, S., Kang, K.C., 2006. Software product line engineering. Communications of the ACM 49 (12), 28–32.

Tian, K., Cooper, K., 2006. Agile and software product line methods: are they so different? In: 1st International Workshop on Agile Product Line Engineering (APLE'06). IEEE Computer Society: Baltimore, Maryland, USA.

Yin, R., Campbell, D.T., 2002. Case Study Research. Sage Publications Inc.

Geir Kjetil Hanssen works as a researcher on software process improvement and methodologies at SINTEF ICT, Norway's largest independent research institution. He has a M.Sc. degree in informatics from the University of Trondheim and 3 years of industrial experience. Currently he also holds a position as Ph.D. student at the Norwegian University of Science and Technology.

Tor Erlend Fægri works as a researcher on software process improvement and software architecture at SINTEF ICT, Norway's largest independent research institution. He has a M.Sc. degree in computing science from the University of Glasgow and 6 years of industrial experience. Currently he also holds a position as Ph.D. student at the Norwegian University of Science and Technology.

Paper 4

Hanssen, G. K., Yamashita, A. F., Conradi, R. and Moonen, L. (2010). Software Entropy in Agile Product Evolution. In proceedings of 43d Hawaiian International Conference on System Sciences (HICSS'10), Hawaii, USA, 4-7 January. 1-10.

Paper 5

Hanssen, G. K. (2010). Opening up Software Product Line Engineering. In proceedings of 1st International Workshop on Product Line Approaches in Software Engineering, in conjunction with the 32'nd International Conference on Software Engineering (ICSE), Cape Town, 2 May. 1-7.

Paper 6

Hanssen, G. K. (2010). A Longitudinal Case Study of an Emerging Software Ecosystem: Implications for Practice and Theory. Journal on Systems and Software (under review).

A Longitudinal Case Study of an Emerging Software Ecosystem: Implications for Practice and Theory*

Geir K. Hanssen [a, b.]

[a] *Department of Computer and Information Science, Norwegian University of Science and Technology (NTNU), Sem Sælands vei 7-9, NO7491 Trondheim, Norway*

[b] *SINTEF ICT, NO7465 Trondheim, Norway*

Abstract

Software ecosystems is an emerging trend within the software industry implying a shift from closed organizations and processes towards open structures, where actors external to the software development organization are becoming more involved in development. This forms an ecosystem of organizations that are related through the shared interest in a software product, leading to new opportunities and new challenges to the industry and its organizational environment. To understand why and how this change occurs, we have followed the development of a software product line organization for a period of approximately five years. We have studied their change from a waterfall-like approach, via agile software product line engineering, towards an emerging software ecosystem. We discuss implications for practice, and propose a nascent theory on software ecosystems. We conclude that the observed change has led to an increase in collaboration across (previously closed) organizational borders, and to the development of a shared value consisting of two components: the technology (the product line, as an extensible platform), and the business it supports. Opening up both the technical interface of the product and the organizational interfaces are key enablers of such a change.

Keywords: software ecosystems, software product line engineering, agile software development, longitudinal case study.

1 Introduction

A recent development within software engineering is the emergence of *software ecosystems* [1, 2]. This new concept and its implicit reference to ecology imply a shift of focus from the internals of the software organization (the individual organism) towards its environment and the relations and actions within (the ecosystem). Viewing the software industry and the market it serves as an ecosystem may introduce a set of new challenges and opportunities [3], for example new business models, open innovation, collaborative development, issues of ownership, strategic planning, and variability management. This seems to be a part of a general development in the software industry [4], where customers expect to be more involved in the shaping of the technology they use, where innovation is no longer an internal matter, where time to market is decreasing, and adoption rates are increasing.

*) Submitted to the Journal of Systems and Software - Under review

To understand some of this ongoing development we have studied CSoft, a medium size software product line organization for a period of approximately five years. During this time they have moved from a closed and plan-driven approach to a practice of agile software product line engineering (SPLE) and now further towards a software ecosystem. Based on three recent studies of this organizational development we have designed and conducted a final study, collecting new qualitative data to investigate in more detail this development in general, and how the organization relates to its environment in particular. Thus, the research question for this study is:

> Why and how is software product line engineering developing towards a software ecosystem?

Our study will give insight into three aspects of this question: 1) how this ecosystem has emerged, 2) how the present organization works in terms of its structure, its processes, and its product line, and 3) how the organization relates to actors in its external environment.

To help develop this understanding and to contribute to the growing research on software ecosystems, we relate our findings from this case study to the general theory of organizational ecology [5, 6], which is derived from socio-technical theory. This theoretical platform has matured for decades and describes how organizations in general relate to their external environment. We have found that this knowledge is relevant, applicable and beneficial to the software engineering domain, and that existing literature on software ecosystems lacks this theoretical connection. Several recent studies have suggested definitions and developed important concepts but the terminology and connectivity between these concepts are still vague. We believe that this field of research can benefit from the development of an empirically grounded theory of software ecosystems. The results obtained from this study are used to propose such a theory, as a starting point. We also discuss implications for practice as well as providing advice for further research.

2 Background

2.1 Socio-technical theory and organizational ecology

Socio-technical theory is a view of the organization as the sum of and interplay between a social system and a technical system. That is 1) the people, their relations, their knowledge, and how they work together as a whole, and 2) the tools and techniques being used to perform the work. A fundamental principle of socio-technical theory is the natural interdependence between these two systems, meaning that to improve the performance of the organization (productivity, quality of work etc.) both subsystems have to be considered at the same time. Changing one affects the other. This principle was first drafted by Trist and Bamforth in 1951, based on their studies of changes of work processes in coalmines [7].

Through studies that followed, the theory has been developed to consider how organizations relate to their external environment [5]. This implies that an organization has to be understood as both 1) the internal interplay between a social subsystem and a technical subsystem (the socio-technical system), and 2) the interplay between the organization and its external environment.

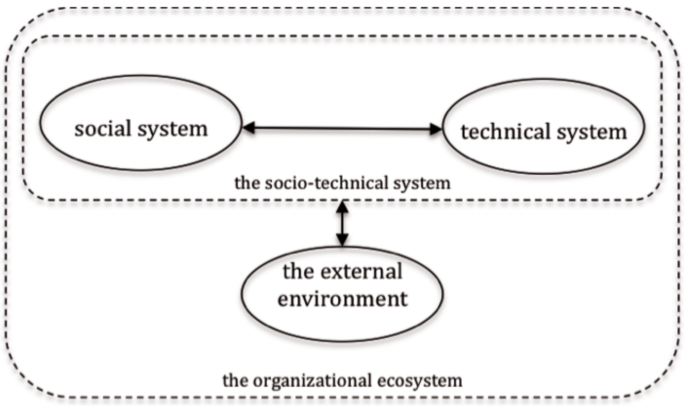


Figure 1 - The socio-technical system and the organizational ecosystem

Emery and Trist [ibid.] developed a simple classification system of four types of organizational environments – forming a series, in which the degree of causal texturing is increased. Thus, understanding and ordering the types of environments is useful in understanding socio-technical systems beyond the limits of a single organization. The first, and the simplest type, is 'the placid, randomized environment' where "goods" and "bads" are unchanging, and are randomly distributed in the environment [ibid., p.7]. The optimal strategy is to 'do one's best' on a purely local basis – there is no difference between strategy (planning) and tactics (execution). The second type is 'the placid, clustered environment' where "goods" and "bads" are not randomly distributed but band together in certain ways. Strategy is different from tactics, and survival becomes critically linked with what an organization knows about its environment. Organizations in this environment tend to become hierarchical, with a tendency towards centralized control and coordination [ibid., p. 8]. The third type, 'the disturbed-reactive environment', is an environment where there is more than one organization of the same kind. The existence of a number of similar organizations becomes the dominant characteristic of the environmental field. These organizations compete, and their tactics, operations and strategy are distinguished. The flexibility encourages a certain decentralization, and it also puts a premium on quality and speed of decisions at various peripheral points [ibid., p 9]. The fourth, and the most recent type is 'turbulent fields'. This type implies that significant variances arise from the field itself, not simply from the interaction of the component organizations. Three trends contribute to the emergence of these dynamic field forces: i) the growth to meet type-three conditions, ii) the deepening interdependence between the economic and other facets of the society, and iii) the increasing reliance on research and development to achieve the capacity to meet competitive challenge. A change gradient is continuously present in the field [ibid., p. 10].

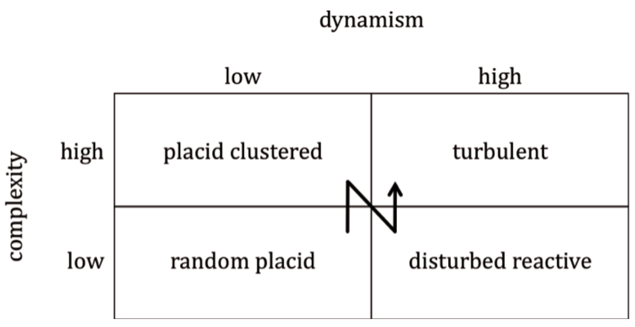


Figure 2 Types of organizational environments

This interplay, inherent in turbulent organizational environments, has been further studied, leading to the development of the concept of **organizational ecology** [6]. It is particularly relevant to organizations operating in complex and unstable domains. Viewing the organizational environment as an ecosystem means that it is considered to be an open system as opposed to a closed one, organizational borders are permeable, and organizations relate dynamically to other organizations in the same field.

Developing the ecology concept further, Trist describes three classes of organizations within the turbulent environment. In a Class 1 system member organizations are linked to a key organization among them. The key organization acts as a central referent organization, doing so even though many of them are only partially under its control, or linked to it only through interface relations. Interface relations are as basic to systems of organizational ecology as superior-subordinate relations are to bureaucratic organizations. Interface relations require negotiation as distinct from compliance – a basic difference between the two types of system. In a Class 2 system the referent organization is of a different kind. It is a new organization brought into being and controlled by the member organizations rather than being one of the key constituents. A Class 3 system has no referent organization at all.

Technocratic bureaucracies have been the natural organizational form for disturbed-reactive environments (up to the 1960's or so), and this is a form that has been applied to many software engineering organizations. However, this type of system fails to adapt to conditions of persistent and pervasive environmental turbulence, mostly because it is constructed and optimized to work well in stable environments. This leads to the emergence of the new ecologically oriented systems, which show clear differences in that they promote self-regulation (as opposed to centralized control), and that they have a networked character (as opposed to segregated organizations). According to Trist [ibid. p. 172], such systems, lacking formal structure, exist through the use of technology. Further, they also need shared values. Trist used the example of the 60/70's youth-culture that had a set of distinct (political) values. A more appropriate example from a business perspective is a shared value in growth and profit.

Unlike the micro-level (the single organization) and the macro-level systems (society), the intermediate level systems (organizational ecosystems) are hard to see, understand, and describe due to their weak structuring. They are also the most recent type, so there is less experience with them. This relates especially to software engineering ecosystems, which is a new but rapidly advancing concept [1, 2] despite decades of development of software engineering as a practice and business. This approach is driven by the Internet as a rich and speedy collaborative platform (the technology), and a common interest in the product line (the shared value).

2.2 Software ecosystems

Software ecosystems is a more recent term, that refers to a networked community of organizations, which base their relations to each other on a common interest in a central software technology. Some other definitions of this emerging concept have been proposed, for example by Jansen et al. [8]: "*a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them.*" (p. 2). Another definition by Bosch [1], focusing more on the common interest in the software and its use, is: *"the set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions."* (p 2).

Well-known examples of communities that may be seen as software ecosystems are Apples iPhone/Appstore platform, and the open-source development environment Eclipse. The first is an example of a partially closed and controlled ecosystem, and the latter is an example of an open ecosystem allowing more flexibility in use and development. This simply illustrates that the ecosystem concept may refer to a wide range of configurations. Yet, they all involve two fundamental concepts: 1) a network of organizations and, 2) a common interest in central software technology. These organizations may have different relations to the central software technology, and for this reason, different roles in the ecosystem. There are three key role types. First, one organization (or a small group) acts as the *keystone organization,* and is in some way leading the development of the central software technology. The second key organizational role is the *end-users* of the central technology, who need it to carry out their business, whatever that might be. The third key role is *third party organizations* that use the central technology as a platform for producing related solutions or services. In addition to these key roles, various other related roles might be part of the ecosystem [9], for example standardization organizations, resellers, and operators.

A fundamental property of the central software technology is that it is *extensible beyond the keystone organization* [10]. Extensibility can be achieved in various ways, for example by providing an application programming interface (API) or a software development kit (SDK), by supporting exchange of open data formats, or by offering parts of the technology as open source. Opening up the technology in these, and potentially other ways, enables external organizations to use the central software technology as a *platform* where existing services or data can be used and extended. Bosch proposed a Software Ecosystem Taxonomy [1] that identifies nine potential classes of the central software technology, according to classification within two dimensions. The first one is *the category dimension*, which is ranging from operating system to application, and to end-user programming. The second one is *the platform dimension*, ranging from desktop to web, and to mobile. The case discussed in this paper is an application-web type.

The keystone organization has a special position in the ecosystem as it controls, strictly or loosely, the evolution of the central software technology. This may include various responsibilities, for example typical software product development activities such as strategic planning, R&D, and operational support. These responsibilities come in addition to activities specific to ecosystems such as enabling efficient external extensibility, provision of insight into planning and development, and supporting ecosystem partners in various other ways.

One potential benefit of being a member of a software ecosystem is the opportunity to exploit open innovation [11], an approach derived from open source software processes where actors openly collaborate to achieve local and global benefits. External actors and the effort they put into the ecosystem may result in innovations being beneficial not only to themselves (and their clients) but also to the keystone organization, as this may be a very efficient way of extending and improving the central software technology as well as increasing the number of users. Closer relationships between the keystone organization and the other actors may drive both an outside-in process as well as an inside-out process, as described by Enkel et al. [12]. Also, the proximity between the organizations in an ecosystem may enable active engagement of various stakeholders in the development of the central software technology [13].

The ultimate objective for investing in and working towards an ecosystem is that all members will gain more benefits from being a part of it, as compared to the more traditional approach for software product development with segregated roles, a low level of collaboration, and closed processes. A well functioning ecosystem is, in summary, a complex configuration with collaboration across traditionally closed organizational borders. Such multi-organizations are probably not established as a deliberate, planned effort. Rather, they emerge as a result of many congruent factors such as technology development, globalization, new collaborative patterns, and clients becoming more and more accustomed to participating in the shaping of the technology they use.

3 Study method

The case study reported in this paper is the last in a series of four consecutive studies of the software product line organization CSoft, constituting a longitudinal study started in 2004, and now covering five years of the organization's history [13-16]. In addition, two earlier papers written by other authors provide background information on the historical development of the organization [17, 18]. Together, this provides a valuable insight into the longitudinal development of a software product line organization.

The name of the case organization and its product is kept anonymous due to a non-disclosure agreement that has been signed by the author.

3.1 Study type

The study can be classified as a longitudinal single case study. We have applied a set of principles for interpretative field studies defined by Klein and Myers [19]:

Table 1 – Application of Klein and Myers seven principles of interpretive field research.

Principles (from Klein and Myers [16], p. 72)	Practiced in the case study
1. The Principle of the Hermeneutic Circle	Data are collected through repeated interviews with actors playing various roles. The data collection is supported by observations and as collection of relevant documentation. The growing knowledge of the case has guided the data collection.
2. The Principle of Contextualization	The study of the case is conducted from two viewpoints – the present organization and its activities, and how this organization has emerged over time.
3. The Principle of Interaction Between the Researchers and the Subjects	A large part of the collected data is based on semi-structured interviews [17] that followed open interview guidelines to ensure a balance between thematic focus and room for reflection, correction, and discussions. This allows for unplanned but relevant topics to be addressed.
4. The Principle of Abstraction and Generalization	Findings are related to the concept of organizational ecology [5], derived from socio-technical theory. Key principles from this theoretical background are applied to the studied case. Some are adopted, some are adjusted, and some are added.
5. The Principle of Dialogical Reasoning	The theory applied to the case was *not* used to plan and guide the data collection. The applicability of the theory became evident through the analysis after the data had been collected.
6. The Principle of Multiple Interpretations	This principle is followed by collecting data from both external actors and people with various roles in the product line organization.
7. The Principle of Suspicion	The data have been collected and analyzed by the author, who is external to the organization, having no formal responsibilities, interests or agenda, except to create an unbiased view of the organization and its development.

3.2 Data sources

During 2008, 2009 and 2010 new data were collected to investigate the agile software product line organization, its processes, and in particular how they relate to external actors. Data are of three types: interviews, observations, and collected documents. Table 2 shows the list of content for each type.

Table 2 - Collected data

Interviews	R&D manager (semi-structured interview) Manager of Professional Services (semi-structured interview) Product Strategy Group manager (semi-structured interview) Product Strategy Group members (3 semi-structured interviews) Technical Account Manager (semi-structured interview) Team leader (semi-structured interview) Team member/developer (semi-structured interview) Product Strategy Group manager (follow-up interview after observation of the review meeting) 2 (of 3) members from the Architecture Team (group interview)
Observations	Product conference (various presentations and ad-hoc conversations) Customer review meeting (one R&D team + customer team + sales) Webinar presentation of the new API
Documents	Component A-E project roadmaps Chief Strategy Officers keynote at a product conference Chief Executive Officers keynote at a product conference Vice President Product marketing – presentation at a product conference Customer's presentation at a product conference

3.3 Sampling and collection

The focus of this study has been to investigate how CSoft relates to external actors such as customers and third party organizations. This has guided the sampling of interview respondents, selection of events for observation and documents to be collected. Interview respondents have been asked to recommend other respondents, based on their understanding of the study (snowball sampling). A single-person interview lasted approximately 30-40 minutes. Group interviews lasted up to 3 hours.

All data have been collected and stored in a database for later analysis. Interviews were recorded using a digital voice recorder and then transcribed.

3.4 Analysis

Data has been analyzed in two steps:

Step1 – All data were first examined to produce an intermediate analysis report, which documents the development process in terms of roles, activities, and artifacts, in addition to high-level concepts, necessary to understand how product planning and development is conducted. This analysis created a structure by grouping information coming from the various data sources. Examples of such concepts are teamwork, planning, and innovation. The objective of this report was to establish a broad understanding of the context, i.e. the organizational set-up and its processes. The report has been used in the description of the study context (3.5), as well as a preparation for step 2.

Step 2 – All data, in textual format, were analyzed using a tool for qualitative data analysis, NVivo™. Data were coded, meaning that fragments of text, for example statements, facts, comments, concerns, and ideas, were tagged with nodes describing the data fragment. Examples of such nodes, which emerged from the analysis, are '*co-creation*', '*finding the right stakeholders*', '*learning of business processes and domain*' etc. These are detailed in chapter 4 (Results). This way of analyzing the data develops a meaning and an interpretation of the data, and relates fragments from different locations in the data material to concepts, which may be grouped into categories. These results can then be used as the basis for formulating a theory explaining some of the findings. This approach resembles 'grounded theory' in that a theory is developed, and that it is grounded on data [20]. The theory being developed may be new, but it can also be related to an already established theory. In the case of the CSoft study, the analysis is related to the organizational ecology concept explained in section 2.1, and it seeks to apply this theory in a software product-engineering context. Implications for theory are discussed in section 5.1.

3.5 Study context

3.5.1 The organization, processes and the product line

The organization. CSoft is a medium-size software company that develops, maintains, and markets a single product line under the same name. They have now become the market leader in the high-end segment of the market. Currently CSoft employs about 260 people, including more than 60 developers. The main office is located in Oslo, which houses the main section of the development department as well as top management and various support services such as operations, technical support, sales, training and others. The rest of the organization is distributed internationally with development departments, sales and other support services in Eurasia and in the USA.

The development department is organized as a set of teams, each responsible for one of the main modules in the product line. A team is mostly a fixed group of 4-6 developers and a team leader, who is experienced in the domain and the technology. The teams share some supportive services such as the Chief Technical Officer (CTO), a system architecture team, and QA-services.

Being a product line organization means that strategic planning is a natural and important activity. This used to be a side activity of some of the supportive roles but has now developed into a fulltime prioritized internal service. A Product Strategy Group (PSG), consisting of five product managers, is responsible for developing a product roadmap for each of the main modules, and supports their development.

The processes. The overall SPLE process at CSoft can be described as three interacting main processes, each with a different time horizon [13]. First, the PSG drives a continuous *long-term (1-2 years) strategic process*, creating product roadmaps based on input from nearly all parts of the organization as well as several external sources. These roadmaps are high-level plans, or a vision, for the product line looking one to two years ahead. They typically present business cases, key stakeholders to participate in development, and prioritized product qualities, instead of functional requirements or feature descriptions. The main content of these plans is made visible externally to the organization through various meetings with customers and partners, at conferences, and through other channels. Roadmaps do not describe specific design decisions but rather high-level guidelines, which are elaborated when detailed plans are laid out for the development projects. In some cases, customers or related third parties visit the R&D department to have close meetings directly with one or more of the development teams to elaborate ideas and discuss needs.

The second main process is the *agile development process* Evo [21], which the R&D department follows to manage the approximately one-year long development projects, leading towards the next main release of the product line (all components are released at the same time). Each component team runs an Evo-project, meaning that development is done in fortnightly iterations, and that each iteration delivers new working software. Each iteration ideally starts with a meeting with an external stakeholder to explain and discuss needs and requirements. At the end of the iteration the team meets with the stakeholder again to get feedback on the outcome (new or improved software) from the iteration. Customers come from all over the world, using a web meeting solution (WebEx™) to communicate as effectively and closely as possible. This is a radical change compared with the previous waterfall approach where feedback was rare.

The third and last process is *the operational process*, which encompasses the day-to-day operations such as support, training, sales and marketing, and high-level maintenance. Apart from being common functions in a product organization they are also highly valuable sources of input to both the strategic process and the Evo development processes as they represent a wide, diverse, and continuous interface with customers.

The product line consists of five main modules, which together support the core business operation of the customers: a value chain of planning, data collection, analysis, and reporting of results. The composition and use of the modules varies according to customer and case. Some modules can be used in any configuration, while the use of others depends on the situation. The software comes with a set of predefined configurations for the most common usage scenarios.

The product line offers an Application Programming Interface (API), which is implemented as standard web-services. Most modules offer an API, which enables clients to integrate the product with other systems, and which is extensively used by other third-party organizations to offer additional software solutions and/or services. More than 60 such partners now base their business partly or completely on using the CSoft product line as a platform through these APIs.

3.5.2 From creative chaos to an emerging software ecosystem

CSoft was established in 1996 and has grown continuously since then. This development has gone through three phases, and has now entered a fourth. This section presents a summary of these phases of development, and it indicates some important milestones in the development of the organization.

The timeline in figure 3 shows the main events [22] in the development of the organization, the approximate increase in staff, and the studies of the organization.

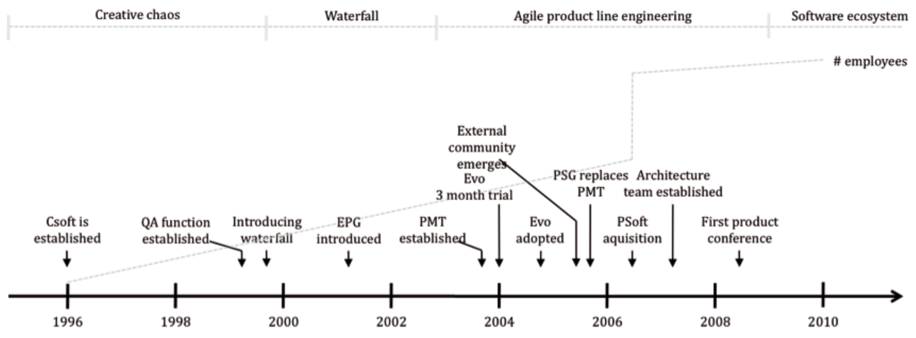


Figure 3 - Timeline of the development of the case organization

1996-1999: "Creative chaos"

The company initially grew out of a small business providing manual services to very few clients. A simple homemade software tool grew into a solution that could be sold as a stand-alone software product. The main focus of the company changed, and the development of this product became the main objective. At the start, in 1996, there were only a few employees providing the product to a handful of clients. The process can best be described as ad-hoc since the main drivers were almost daily interactions with and feedback from customers. A customer request was literally routed directly to the developers. This start-up phase was a 'creative chaos' – that is, it had nearly no plans and no control, but it was undoubtedly extremely creative and productive. The product grew rapidly, not only in terms of features and functionality, but also in terms of defects and complexity. Work became stressful, with little control, and a lot of overtime.

1999-2003: Waterfall

As the number of customers increased the organization formalized the development process, mostly according to the principles of the waterfall model [23]. This somewhat disreputable approach to organizing software development emphasizes upfront detailed planning of requirements, design, and development. The development is divided in consecutive phases, where requirements are developed into a design, and the design is developed into a software system, which is tested close to deployment. Prior to this, the R&D department was extended with a QA-manager. This structured approach established a certain level of control, and helped the organization in the continuing development of their product, which grew alongside the customer base. After a few years, several problems arose, clearly related to the waterfall approach [17]. Testing and verification was postponed to the final stages leading to late identification of problems, which in turn caused much rework. Also, requirements were almost solely focused on functionality, leaving out the quality perspective. To support a growing R&D department, and to spread knowledge about the formalized development process, an electronic process guide (EPG) was developed, and made available via the intranet [18]. The product expanded and eventually became a product line, capable of serving various usage scenarios. To manage this increasing complexity a Product Management Team (PMT) was formed – a group of experienced employees with other main responsibilities, who were supposed to spend part of their time in strategic product planning. In addition various specialized functions, beyond software development, were introduced such as Technical Account Managers (TAM), the operations department, and training services.

2004 – approx. 2009: Agile product line engineering
Due to a critically declining process performance, management of R&D was looking for a way to improve the situation. At a software engineering conference, a few representatives from the company learned about evolutionary development and the Evo method [21]. As it seemed to address some of their concerns they initiated a three-month testing period of this radically different development approach in one of the release projects. Instead of an extensively prepared process adoption, they started out with a few principles, focusing on requirements management, where functional requirements were replaced by explicit expression and evaluation of product qualities, preferably stated by customers involved in the development process [15]. Early experience showed that the number of issues near release was reduced, and that the delivered product matched customer expectations better than before. After this initial process trial, Evo was adopted on a permanent basis [13, 14]. Alongside the growth in the organization and the product line the PMT group was re-established as a full-time Product Strategy Group managed by a Chief Strategy Officer (CSO). Another supportive service, the architecture team, was established, originally with three full time members. Their task was to handle the excessive level of system entropy [16], and to support R&D in architectural issues. In 2006 CSoft acquired a former competitor (PSoft), and boosted the number of employees to 260. Adding new offices, for both R&D and marketing, was a considerable challenge. Through extensive internal training in the following year, the new organization was mostly using Evo as the development process.

4 Results

This section structures and summarizes the results obtained from the recent study of CSoft. First, we look at how the present product line organization relates to its clients. Then we describe the recent emergence of a community of third party organizations. Together, these results show how CSoft relates to its external environment, constituting a software ecosystem.

4.1 Engaging customers

An important aspect of the continuous change over the past years is how CSoft now relate to their customers. The shift from a plan-driven approach to an agile approach has included an increased proximity to the customers. The initial experience from collaborating with customers as external stakeholders in development projects [14, 15] showed positive effects such as better management of requirements, and improved motivation among developers. It also showed that the relationships with the stakeholders were fragile, and that it takes continuous and careful management to maintain their motivation to participate. These initial lessons inspired CSoft to further develop and actively exploit close relationships with their customers.

The main motivation for customers to spend time participating in the development projects is the ability to affect development: no payment or any other compensation is provided. One of the product managers explained:

> *"…they see their wishes or their requirements or whatever in the product at the end. And then you get very nice feedback like 'I can see that I said this and that, and in the next release you did it'".*

This collaboration forms a self-regulating system where the supplier and the stakeholders mutually adapt to each other through their shared interest in developing the software product line. This usually works well, but there is always a risk of having external stakeholders, which do not provide the necessary input, as explained by the manager of the PSG team:

> *"Everybody has busy jobs and projects that need to be on time etc. It happens quite often that we have to cancel these meetings or that they haven't done anything since the last time. Then we can only show them what we've done and get some ad-hoc feedback..."*

The PSG manager also explained that it is relatively easy to discuss ideas, but that it is more of a challenge when they are included in the development process:

> *"There's no problem to get them to discuss high-level plans, but it varies when it comes to the development process"*

Maintaining the motivation for participating is an important task for the PSG.

Interestingly, large and leading customers tend to expect and demand to be more and more involved at both planning and development stages. One example is a product conference keynote given by the VP from one of the large customers. He stated several 'requirements' (this was the word he used) for being involved, for example:

> *"Regular meetings with product development teams", "To work as a stakeholder on new software developments that are key to us.", "Help to guide product strategy."*, and others.

Finding the "right" stakeholders for participation in the development projects is not done through a formal and structured process, but is mostly based on the collective knowledge about the customers. The PSG manager says,

> *"We don't have a formal process for selecting stakeholders. We have internal discussions, listen to sales people etc. We know which customers have asked for certain features or improvements or those that are heavy users of a particular type of functionality."*

In addition, experience from previous participation is also useful as explained by the R&D manager:

> *"We have become better at selecting [external stakeholders]. Those that have disappointed us are never asked again. You end up with a pool of persons that you know you can trust."*

An important part of recruiting customers as external stakeholders is to communicate to them clearly the opportunity, which is given to them. The PSG manager explains:

> *"..quite often we talk about our development process. We do it in sales situations because it tells that our goal is to solve business problems for our customers. This level of interaction and the way we try to listen has been well received, and now some customers insist on being involved in development."*

In the very start when Evo and collaboration with external stakeholders were at an experimental stage it took quite some effort to recruit and stakeholders to the development projects and to keep them active [14]. After some releases where collaboration with external stakeholders have become an integrated part of the development process the situation is turned upside-down. When asked to explain this relationship, one of the product managers told us:

> *"It's almost a problem because as soon as you offer the capability of being a stakeholder, the hardest part is rejecting people, turning them away from actively participate. So people are very keen on participating."*

Co-creating the product line is one of the most significant effects of engaging and communicating with external stakeholders. The rationale is simple, CSoft have the most up to date knowledge of the technology, and the ability to make use of it in the development. Likewise, customers hold the most up to date knowledge of their own business domain, and how it seems to develop. These two pools of knowledge and competence are joined in several ways. One important arena for sharing and gaining knowledge is the product conference where management, strategists, developers and other internal actors get to meet externals from various customers and third parties. Equally important – customers can meet other customers, third parties can meet customers or internals etc. This shows the networked character of the ecosystem that is shaped around the product line. Some examples from the product conference in 2008:

> *A former customer of a competing solution was seeking experienced customers to discuss the product line and share experience.*
>
> *Several providers of third party products and services were having stands at the conference, communicating with both existing and potential customers and developers from CSoft.*

Another major event, which is more directly focused on the development of the product line, is the annual Advisory Board meeting. Top management from some of the largest and most demanding clients meet with the PSG and other actors who are involved in the shaping of the product strategy. A PSG member explains that they meet to:

> *"…discuss high level product strategy and how the demands of their companies and the market are developing." Bringing together major competitors like this was a daring thing to do according to the PSG manager: "The first time we did this it was a bit exciting – would they discuss issues openly, and would they open up? It turned out that they did very fast. They have many concurrent needs, and even though they are competitors they see the value of doing this."*

From a practical viewpoint, we see that tools and infrastructure for collaboration are important enablers for co-creating the product line. Especially the Webex online meeting solution lowers the threshold for having frequent and detailed meetings with stakeholders:

> *From our observation of one of the customer review meetings we saw a lot of very detailed discussions that were made possible by on-the-fly demonstration of the software through the screen sharing solution. This sparked detailed discussions both on the customer side and among the development team. The meeting resulted in a list of clear actions points to be addressed in the next development iteration.*

Close corrective feedback in the Evo development projects is another approach to co-creating the product line, but on a tactical level. One of the developers describes the meeting with the external stakeholder at the end of the two-week Evo iteration:

> *"What you get during a meeting is often very valuable. Especially when you are about to move in the wrong direction, which you can adjust. We get feedback saying that our solution is not quite what they had in mind or what they need."*

This demonstrates one important function of the agile process; the development teams get nearly immediate (within two weeks) and detailed feedback. This closeness to a few selected customers means that CSoft must also consider the needs of other customers, as they are the referent organization, which always has the last word in the development of the product line. This is partly achieved through Evo's focus on product qualities instead of product features, which are typically emphasized in plan-driven development methods. This is a useful abstraction, and it turns the focus from predefined design (features) to effect and impact (qualities). Both the product roadmaps and the evaluation meetings at the end of the Evo iterations evaluate the product qualities. This means that both the development teams and the external stakeholders have to consider *why* something is needed, leaving the *how* to the developers. The PSG manager explains:

> *"..we take one step back, and try to think about why our stakeholder needs this, and then rethink other ways of solving their problem. It is in this type of process that the smart things can turn up – that your thinking is totally new and that you come up with a solution which may be a totally different way of doing it, maybe faster...".*

Catching and following up on customer ideas on an ad-hoc basis is equally important as involving customers in regular processes such as roadmapping and the Evo development projects. At the product conference:

> *A customer representative told about a case where his company gave input to CSoft on some changes they would have liked to see. This led CSoft to invite a delegation from the (abroad) customer to the R&D department in Oslo. Ten CSoft people spent the whole day discussing the solution with four representatives of the customer. This was perceived very positively, and in the end actually affected the software.*

Some of the largest customers may also request dedicated workshops to discuss needs and ideas. The PSG manager talks about this:

> *"...for some of our largest customers, mostly by their initiative, we organize workshops once a year, usually on a strategic level. They want to know what the roadmaps may bring for the next couple of years, and talk a lot about their needs etc.".*

The close contact with customers is also a valuable source of learning about competing solutions. One of the team leaders talks about customers visiting the R&D department:

> *"In these meetings they demonstrated the solution they used today, and actually demonstrated how they used the competing solutions – what worked well and what needed improvements, as well as ideas they might have. These meetings gave the team a wealth of details, and it was quite clear what to deliver to the stakeholder."*

The PSG manager also explains the value of learning of the use of competing solutions:

> *"…alternatively they do it using other tools today when not using our solution. The option to work more closely with them and to get that knowledge made us more capable to meet their needs better than before when the development was more of the black-box type".*

A phrase from one of the roadmaps illustrates the business impact this may have:

> *"Through a client we have been given a thorough demonstration of the competing solution NN, and by implementing support for [some advanced functionality] and a couple of small features, the X-module will by far exceed their corresponding functionality. These improvements alone will ensure we win one [sales] deal, and have also been brought up by several other clients/prospects."*

Learning the business processes and domain is another valuable outcome from the direct contact with selected stakeholders. One of the developers talks about one of the stakeholder meetings in an Evo project:

> *"…we have tried to solve a task in a way we believed would be reasonable, but to people who actually use this it is obvious that we have misunderstood the process. This gives us guidance as early as possible."*

A PSG member tells about another case:

> *"They [the customer] were here in a workshop for two days. We presented the roadmap [for module X] and they presented their wishes and their business, what they are doing."*

This illustrates the shared interest that the customers and the supplier have – customers want to learn about the product line and its development. Correspondingly, CSoft learns about the business that their product line is supporting.

4.2 An emerging third party community

At present around 60 external organizations base their business completely or partly on CSoft as a platform. This can be value-adding solutions or products, related services, and consulting. Examples are solutions for data visualization or voice data capture technology, assistance in using various components in the product line, and training. This networked community [24] has not been planned and deliberately established by CSoft, it has emerged spontaneously over the past years. This emergence is mostly driven by customers' need for additional features and services on one hand, and the opportunity to extend and use the product line as a platform on the other hand. Also, building solutions and providing services based on the product line means that external organizations get immediate access to a large group of established users of the product line.

Providers of third-party solutions are considered to be important external stakeholders, and are included in the development of the product line in very much the same way as customers.

> *During a product conference, a representative from a third party company, delivering an integrated product, explained that when they needed to improve the integration with the CSoft platform they took on the role of an Evo stakeholder. Communication was mostly done by phone, supported by web meetings with screen sharing.*

Offering an efficient integration technology enables a third party community. Over the past few years a set of simple APIs have been offered to enable external actors to make extensions to the product line. The development of these APIs have followed the development of the product line, where each new release has improved existing and offered new APIs due to requests from external actors. This means that there is a long (a year) connection time between a request for an interface and its actual release. As more and more externals have made use of this connection point to the software it has been given increasingly higher priority in the development of the product line. An excerpt from one of the roadmaps exemplifies this:

> *"We are in dialogue with some clients/prospects who are building their portal in a Content Management System, and need to integrate content from [module X] into it. Some competitors seem to have APIs that are easier to use than our SOAP[1] based APIs, making it easier to integrate with other portals/communities. It is therefore an ambition to provide an easier API for including module X content into an external portal."*

Due to the extensive use of the API's by externals and their increasing demand for integration with the product line it became clear that the simple web-service based interface had become obsolete. This has led CSoft to develop and offer a new API called FlexibilityFramework (FF), which enables a closer integration to core services in the product line than the previous (and still existing) simple messaging-based APIs offer. A recent webcast, where the CSO presents FF explains further the motivation for this improved interface:

> *"CSoft is like a supertanker. It is large, can take huge loads, travel far, and take heavy weather. These are all very positive things, on the other hand, the consequence of that approach is that we are quite careful at looking after the supertanker. That means various procedures, on policy, on quality assurance and so forth. And that means that we get less nimble than we would like. The question we posed ourselves is how can we behave like a speedboat while having all the benefits of the supertanker? I'd like you to think of FF as the speedboat. The tanker is still there. It will still take heavy loads and perform extremely well, but in order to be nimble we can build a few speedboats. And they have independent lives from the supertanker and can run on different development schedules."*

The last argument is worth a comment; with this new interface to the product line external actors are disconnected from the long release cycles of the product line, and can develop value-adding solutions independently. This is likely to further drive the growth of the third party community.

[1] Simple Object Access Protocol, http://www.w3.org/TR/soap12-part1/

Actively supporting the community has become a regular activity in addition to the continuous development of the product line. As this community has emerged and grown, CSoft have seen its value, and started to actively support it. In 2007 a dedicated web-portal was launched to make this community visible and each partner is listed and presented. There are five types of partners, those offering technology that is integrated with the product line, those offering value adding services, some can prepare the use of the product line, some can use it on behalf of clients and some offer consultancy services.

5 Discussion

We have now described the CSoft case, emphasizing the present organizational set-up, its processes and the product line, the development timeline of the organization (section 3.5.2), and how the present organization relates to its external environment (section 4). Using this insight we now seek to provide answers to our research question: *Why and how is software product line engineering developing towards a software ecosystem?*

First of all, the initial motivation for changing the waterfall-like development process by adopting Evo in 2004/2005 was that CSoft struggled with unstable requirements incurring high costs due to little flexibility in the process. Much emphasis was given to extensive and thorough requirements engineering upfront, but with little effect [15]. The immediate experience from involving stakeholders in the short Evo development iterations was that developers felt more comfortable and secure by having this close and continuous dialogue on requirements and results [ibid]. However, in the first release projects using Evo, it became a considerable challenge to maintain the motivation of the external stakeholders throughout the project. The new process was fragile [14].

(Change 1) From the recent study we see that this has clearly changed; now external stakeholders are keen to participate – CSoft actually have to turn down candidates. This change is the result of a learning process that has progressed during the first years of using Evo – customers have gotten to know of this practice and some have gained experience as stakeholders. The engagement of customers and users is generally considered to be an important success factor in any kind of software development [25, 26].

(Change 2) We can also observe another change that took place internally at CSoft. The first experimentation with Evo was only done as an R&D-internal matter, like a kitchen experiment. However, as this turned out to be an improvement of the development practice, this way of working eventually became adopted in the rest of the organization. Now, all parts of the organization, from operational support to the top management, are supporting this practice. An example is the CEO explaining the software development process Evo and its strategic importance in his keynote at a large product conference. Another example is the strengthening of the PSG, which has a liaison function between customers and development teams. This tells us that changing a product line organization takes effort and time, and that both internal and external actors need to learn from practice to accept this opening of the organization and its work processes.

(Change 3) Another change we can see from the results is an increasingly higher external visibility of plans and strategies. Initially this kind of information was kept internal, but it is now more and more openly communicated through various channels. It has turned out that doing this does not introduce the presumed risk of leaking vital information to competitors, but that it is rather an advantage as external actors see what might be coming, they can relate it to their own business, and potentially respond to it.

(Change 4) Another related change is the opening of the product line at the technical level, first with the SOAP-based APIs and now the recent and more efficient Flexibility Framework. Initially this represented a minimal and very limited opportunity for extending the product line, but it quickly grew to a considerable extent as it represented tangible business value. This aspect has eventually been given more attention, and has been designated as strategically important in some of the roadmaps. We see several benefits from allowing externals to use the product line as a platform. Firstly, it increases the variability of the product line – it can be used in more specialized ways, serving more needs. Secondly, existing users represent a great opportunity to the third parties (being the second component of a symbiosis-like relationship). Thirdly, letting externals deal with specialization and minor extensions enables the product line organization itself to maintain focus on developing the core product line. This may be the most important effect [27].

To recap the research question - this change and the organization it has resulted in explains *why and how* software product line engineering at CSoft has developed towards a software ecosystem.

Why	1) Customers expect and have learned to value to be involved in development and in product strategy making. 2) A plan-based development approach is unfit when serving a volatile domain where the product line is under continuous and extensive development. 3) The total demands and requirements from customers can become too high for one product line organization to manage alone.
How	1) CSoft learns about the business it serves through active collaboration with customers and third parties. 2) CSoft makes strategy and plans visible externally. 3) The technical interface of the product line is opened. 4) Both customers and value-adding third parties are considered as external stakeholders. 5) CSoft actively support and assist the community of third parties.

5.1 Implications for theory

Software ecosystems, as a concept, have the potential of becoming an important field of practice and research in the years to come. We propose to shape a theoretical platform for this research. Just like the taxonomy suggested by Bosch [1], a theory of software ecosystems is valuable and useful to generalize the concept and bring together results from more empirical studies. It may over time develop towards a unified and empirically justified understanding of the concept.

Fortunately, the theory of organizational ecology [6], briefly presented in the background section, seems to fit well as a starting point. It concerns organizations operating in complex and unstable domains, in principle a suitable description of software ecosystems – and certainly of CSoft. Using this general theory of *organizational ecology*, we derive a set of theoretical propositions suitable to *software ecosystems*:

1. Member organizations in a software ecosystem are linked to a key organization among them, which acts as **a central referent organization**, doing so even though many of them are only partially under its control or linked to it only through interface relations. (This is the Class 1 system according to Trist's classification). CSoft is an example of such a referent organization. None of the external organizations are formally controlled by CSoft. However, all activity in the ecosystem is related to the product line, which is controlled by CSoft.

2. Software ecosystems promote **self-regulation**. Our study of CSoft show that the collaborative approach can be seen as a self-regulating system in that the referent organization to a large degree adapts to its external environment, and that the external environment adapts to the referent organization. This is in contrast to the previously centralized control that was applied in the development of the product line.

3. Software ecosystems have a **networked character**. CSoft's and its external environment constitutes a network of customers and third party organizations. Even competitors may be considered a part of this network, although this aspect has not been studied in particular here.

4. Software ecosystems **exist through the use of technology**. The ecosystem, which CSoft is a part of, relies on the use of technology to enable collaboration. Examples are web-meetings, web-casting, and the software-as-a-service deployment model.

5. Software ecosystems have **shared values**. In the CSoft ecosystem the software (product line) is this shared value. For CSoft, the value is revenue from licenses and services, for the customers the value is improved business operations, and for the third parties the value is revenue from sales of value-adding solutions. This common interest in the shared value creates motivation to collaboratively care for the shared value.

These five propositions constitute a start of a theory for software ecosystems. In addition to these principles adopted from Trist's work [ibid.] we also propose two extensions:

6. **The shared value of a software ecosystem is *both* the software product and the business domain**. Through the increased proximity to the external environment, CSoft have an interest in both the product line *and* the business it serves. Likewise, customers have an interest both in their business *and* the technology they use to drive it.

7. As a software ecosystem emerges, **control moves from the supplier of the software to its users**. An opening of the product line process, with external stakeholders participating in development and with external visibility of plans and strategies, means that some of the control move towards the users. Users in this case are both customers and third-parties. This affects the motivation to collaborate, and is of benefit to all members of the ecosystem.

5.2 Implications for practice

From the analysis of our findings we derive a set of implications for practice, relevant to other software product line organizations similar to CSoft.

- Support the external environment by sharing information on plans and strategies – this opens a channel for valuable input and enables collaboration with externals.
- If appropriate, encourage and support a third party community; it can be a valuable extension to the normal development of the product line.
- Establishing and benefiting from a software ecosystem takes time. A successful development relies on repeated cycles of experimentation and learning. This learning process needs to involve all types of actors.

Based on our findings we can derive a strategy that can be of practical value to other product line organizations. The shared interest in the product line (the shared value) is a key enabler for driving the collaboration between the actors in the ecosystem. Thus, a viable strategy would be to 1) make the product line supplier more involved in the development of the business domain and 2) make external actors more engaged in the development of the technology.

5.3 Limitations

The case study of CSoft is subject to three limitations.

Firstly, this is a single case study, which naturally affects the generalizability of the conclusions. Yet there are good reasons for selecting such an approach. First of all, the number of relevant cases is still low. In addition, focusing on a single case means that the study can be more thorough than a study of multiple cases, with respect to available resources. Yin discusses the single case study design [28] (p. 38-41) and presents several arguments in favor of choosing such a design. One of these is particularly applicable to CSoft, namely that it is a unique case, at least for the researcher who conducted the study. According to Yin, such a study may act as a prelude to further studies of a relatively new topic, such as software ecosystems in this case.

The second limitation concerns the completeness of the study. Only a subset of the employees was contacted. Likewise, relatively few samples of all available documentation were collected and analyzed. This is naturally due to the relatively long duration of the study.

The third limitation concerns the applicability of the findings and conclusions of this study. The organization investigated is a medium-size product line organization and a web/application type of ecosystem (according to the taxonomy proposed by Bosch [1]). Thus, results do not necessarily apply to all other types of software ecosystems.

6 Conclusions

Over a period of approximately five years we have studied a software product line organization and its external environment, showing and explaining an emerging software ecosystem.

We conclude that the development that has occurred has produced effects both internally in the product line organization and in its external environment. The change has led to an increase in collaboration across (previously closed) organizational borders, and it has developed a shared value consisting of two components: the technology (the product line) and the business it supports. Opening up both the technical interface of the product and the organizational interfaces are key enablers of such a change

We propose two directions for further research. First, like Jansen et al. also point out [9], we need to see more empirical studies of various types of software ecosystems, how they develop and what effects they produce. Secondly, more studies should contribute to the shaping of a theory of software ecosystems.

7 Acknowledgements

We are grateful to the people at CSoft for sharing their precious time, and providing highly valuable input to this study. This work was done as a part of the EVISOFT project (grant 156701/220) and the Agile project (grant 179851/I40), both partly funded by the Research Council of Norway.

8 References

1. Bosch, J., *From Software Product Lines to Software Ecosystems*, in *13th International Software Product Line Conference (SPLC'09)*. 2009, IEEE Computer Society: San Fransisco, USA.
2. Messerschmitt, D.G. and C. Szyperski, *Software Ecosystems, Understanding an Indespensable Technology and Industry*. 2003, Cambridge, Massachusetts, USA: The MIT Press. 424.
3. Jansen, S., et al., *Introduction to the Proceedings of the First Workshop on Software Ecosystems*, in *First International Workshop on Software Ecosystems*. 2009, CEUR-WS.
4. Qualman, E., *Socialnomics: how social media transforms the way we live and do business*. 2009: John Wiley & Sons.
5. Emery, F.E. and E.L. Trist, *The Causal Texture of Organizational Environments.* Human Relations, 1965. **18**: p. 21-32.
6. Trist, E.L., *A Concept of Organizational Ecology.* Australian Journal of Management, 1977. **2**(2): p. 161-175.
7. Trist, E.L. and K.W. Bamforth, *Some social and psychological consequences of the longwall method of coal-getting.* Human Relations, 1951. **4**(1): p. 3-38.
8. Jansen, S., A. Finkelstein, and S. Brinkkemper. *A sense of community: A research agenda for software ecosystems*. in *31st International Conference on Software Engineering (ICSE'09)*. 2009. Vancouver, Canada: IEEE Computer Society.
9. Jansen, S., S. Brinkkemper, and A. Finkelstein. *Business Network Management as a Survival Strategy: A Tale of Two Software Ecosystems*. in *First Workshop on Software Ecosystems*. 2009.
10. Alspaugh, T.A., U.A. Hazeline, and W. Scacchi, *The Role of Software Licenses in Open Architecture Ecosystems*, in *First International Workshop on Software Ecosystems*, S. Jansen, et al., Editors. 2009, CEUR-WS.
11. Chesbrough, H., *Open Innovation: A New Paradigm for Understanding Industrial Innovation*, in *Open Innovation: Researching a New Paradigm*, H. Chesbrough, W. Vanhaverbeke, and J. West, Editors. 2006, Oxford University Press: Oxford. p. 1.
12. Enkel, E., O. Gassman, and H. Chesbrough, *Open R&D and open innovation: exploring the phenomenon.* R&D Management, 2009. **39**(4): p. 311-316.
13. Hanssen, G.K. and T.E. Fægri, *Process Fusion - Agile Product Line Engineering: an Industrial Case Study.* Journal of Systems and Software, 2008. **81**: p. 843-854.
14. Fægri, T.E. and G.K. Hanssen, *Collaboration and process fragility in evolutionarily product development.* IEEE Software, 2007. **24**(3): p. 96-104.
15. Hanssen, G.K. and T.E. Fægri. *Agile Customer Engagement: a Longitudinal Qualitative Case Study*. in *5th International Symposium on Empirical Software Engineering (ISESE'06)*. 2006. Rio de Janeiro, Brazil: IEEE Computer Society.
16. Hanssen, G.K., et al. *Software entropy in agile product evolution*. in *43d Hawaiian International Conference on System Sciences (HICSS'10)*. 2010. Hawaii, USA: IEEE Computer Society.

17. Johansen, T. *Using evolutionary project management (Evo) to create faster, more userfriendly and more productive software. Experience report from FIRM AS*. in *6th International Conference on Product Focused Software Process Improvement (PROFES'05)*. 2005. Oulu, Finland: Springer Verlag.
18. Moe, N.B., et al., *Process guides as software process improvement in a small company*, in *EuroSPI*. 2002: Nuremberg, Germany.
19. Klein, H.K. and M.D. Myers, *A set of principles for conducting and evaluating interpretive field studies in information systems.* MIS Quarterly, 1999. **23**(1): p. 67 - 93.
20. Glaser, B.G. and A.L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. 1967, New York: Aldine Transaction. 271.
21. Gilb, T., *Competitive Engineering: A handbook for systems engineering, requirements engineering, and software engineering using Planguage*. 2005: Elsevier Butterworth-Heinemann. 480 pages.
22. Pettigrew, A.M., *Longitudinal Field Research on Change: Theory and Practice* Organization Science, 1990. **1**(3): p. 267-292.
23. Royce, W.W. *Managing the development of large software systems*. in *IEEE WESCON*. 1970.
24. Fricker, S., *Specification and Analysis of Requirements Negotiation Strategy in Software Ecosystems*, in *First International Workshop on Software Ecosystems*, S. Jansen, et al., Editors. 2009, CEUR-WS.
25. Chiasson, M.W. and L.W. Green, *Questioning the IT artefact: user practices that can, could, and cannot be supported in packaged-software designs.* European Journal of Information Systems, 2007(16): p. 542-554.
26. Keil, M. and E. Carmel, *Customer-Developer Links in Software Development.* Communications of the ACM, 1995. **38**(5): p. 33-44.
27. Zook, C., *Profit from the core: a return to growth in turbulent times*. 2010: Bain & Company, Inc.
28. Yin, R. and D.T. Campbell, *Case Study Research*. 2002: Sage Publications Inc.

Geir Kjetil Hanssen works as a researcher on software process improvement and methodologies at SINTEF ICT, Norway's largest independent research institution. He has a M.Sc. degree in informatics from the University of Trondheim.

Declarations on co-author consensus

Papers P1, P2 and P3: Tor Erlend Fægri
Paper P4: Aiko Fallas Yamashita, Reidar Conradi and Leon Moonen

To Whom It May Concern

Statement of authorship on joint publications to be used in Geir Kjetil Hanssens PhD-thesis

(Cf. NTNU PhD-regulation § 7.4, section 4 and dr.philos regulation § 3, section 5)

As co-author on the following joint publications in Geir Kjetil Hanssens PhD-thesis:

1. *T. E. Fægri and G.K. Hanssen, Collaboration and process fragility in evolutionarily product development. IEEE Software, 2007. 24(3): p. 96-104.*

2. *G. K. Hanssen and T.E. Fægri. Agile Customer Engagement: a Longitudinal Qualitative Case Study. In proceedings of 5th International Symposium on Empirical Software Engineering (ISESE'06). 2006. Rio de Janeiro, Brazil: IEEE Computer Society.*

3. *G. K. Hanssen and T.E. Fægri, Process Fusion - Agile Product Line Engineering: an Industrial Case Study. Journal of Systems and Software, 2008. 81: p. 843-854.*

I declare that the candidate's contribution to this work is correctly identified and that I consent that this work is to be used as part of the thesis.

August 2, 2010

[illegible]

Tor Erlend Fægri

To Whom It May Concern

Statement of authorship on joint publications to be used in Geir Kjetil Hanssens PhD-thesis

(Cf. NTNU PhD-regulation § 7.4, section 4 and dr.philos regulation § 3, section 5)

As co-author on the following joint publication in Geir Kjetil Hanssens PhD-thesis:

> *Hanssen, G.K., Yamashita, A.F., Conradi, R., Moonen, L., Software entropy in agile product evolution. In proceedings of 43d Hawaiian International Conference on System Sciences (HICSS'10). 2010. Hawaii, USA: IEEE Computer Society.*

I declare that the candidate's contribution to this work is correctly identified and that I consent that this work is to be used as part of the thesis.

Friday, June 25, 2010

[illegible]

Aiko Fallas Yamashita

To Whom It May Concern

Statement of authorship on joint publications to be used in Geir Kjetil Hanssens PhD-thesis

(Cf. NTNU PhD-regulation § 7.4, section 4 and dr.philos regulation § 3, section 5)

As co-author on the following joint publication in Geir Kjetil Hanssens PhD-thesis:

> *Hanssen, G.K., Yamashita, A.F., Conradi, R., Moonen, L., Software entropy in agile product evolution. In proceedings of 43d Hawaiian International Conference on System Sciences (HICSS'10). 2010. Hawaii, USA: IEEE Computer Society.*

I declare that the candidate's contribution to this work is correctly identified and that I consent that this work is to be used as part of the thesis.

R. Conradi 24.8.2010

Reidar Conradi

To Whom It May Concern

Statement of authorship on joint publications to be used in Geir Kjetil Hanssens PhD-thesis

(Cf. NTNU PhD-regulation § 7.4, section 4 and dr.philos regulation § 3, section 5)

As co-author on the following joint publication in Geir Kjetil Hanssens PhD-thesis:

> *Hanssen, G.K., Yamashita, A.F., Conradi, R., Moonen, L., Software entropy in agile product evolution. In proceedings of 43d Hawaiian International Conference on System Sciences (HICSS'10). 2010. Hawaii, USA: IEEE Computer Society.*

I declare that the candidate's contribution to this work is correctly identified and that I consent that this work is to be used as part of the thesis.

25-6-2010

[illegible]

Leon Moonen

All publications

Hanssen, G. K., Yamashita, A. F., Conradi, R. and Moonen, L. (2010). Software Entropy in Agile Product Evolution. In proceedings of 43d Hawaiian International Conference on System Sciences (HICSS'10), Hawaii, USA, 4-7 January. 1-10.

Hanssen, G. K. (2010), Agile Software Product Line Engineering: Enabling Factors, accepted (minor revision) for publication in Software: Practice and Experience (Wiley).

Hanssen, G. K. (2010). Opening up Software Product Line Engineering. In proceedings of 1st International Workshop on Product Line Approaches in Software Engineering, in conjunction with the 32'nd International Conference on Software Engineering (ICSE), Cape Town, 2 May. 1-7.

Hanssen, G. K., Yamshita, A. F., Conradi, R. and Moonen, L. (2009). Maintenance and agile development: challenges, opportunities and future directions. In proceedings of 25th International Conference on Software Maintenance (ICSM'09), Edmonton, Canada, 20-24 September. 487-490.

Hanssen, G. K. and Haugset, B. (2009). Automated Acceptance Testing Using Fit. In proceedings of 42d Hawaiian International Conference on System Sciences (HICSS'09), Hawaii, USA, 5-8 January. 1-8.

Stålhane, T. and Hanssen, G. K. (2008). The application of ISO 9001 to agile software development. In proceedings of Product Focused Software Process Improvement (PROFES 2008), Frascati, Italy, 23-25 June. 371-385.

Haugset, B. and Hanssen, G. K. (2008). Automated Acceptance Testing: a Literature Review and an Industrial Case Study. In proceedings of Agile Conference, Toronto, Canada, 4-8 August. 27-38.

Hanssen, G. K. and Haugset, B. (2008). Automated Acceptance Testing Using Fit. In proceedings of EuroSPI 2008, Dublin, Ireland, 3-5 September.

Hanssen, G. K. and Fægri, T. E. (2008). Process Fusion - Agile Product Line Engineering: an Industrial Case Study. Journal of Systems and Software 81: 843-854.

Hanssen, G., Bjørnson, F. and Westerheim, H. (2007). Tailoring and Introduction of the Rational Unified Process. In proceedings of European Systems & Software Process

Improvement and Innovation (EuroSPI 2007), Potsdam, Germany, 26-28 September. 7-18.

Fægri, T. E. and Hanssen, G. K. (2007). Collaboration and Process Fragility in Evolutionarily Product Development. IEEE Software 24(3): 96-104.

Dybå, T., Dingsoyr, T. and Hanssen, G. K. (2007). Applying Systematic Reviews to Diverse Study Types: An Experience Report. In proceedings of Proceedings of International Symposium on Empirical Software Engineering and Measurement (ESEM), Madrid, Spain, 20-21 September. 10.

Westerheim, H. and Hanssen, G. K. (2006). Extending the Rational Unified Process with a User Experience Discipline: a Case Study. In proceedings of European Systems & Software Process Improvement and Innovation (EuroSPI 2006), Joensuu, Finland, 11-13 October. 3.11-13.19.

Hanssen, G. K. and Fægri, T. E. (2006). Agile Customer Engagement: a Longitudinal Qualitative Case Study. In proceedings of 5th International Symposium on Empirical Software Engineering (ISESE'06), Rio de Janeiro, Brazil, 21-22 September. 164-173

Dingsøyr, T., Hanssen, G. K., Dybå, T., Anker, G. and Nygaard, J. O. (2006). Developing Software with Scrum in a Small Cross-Organizational Project. In proceedings of European Systems & Software Process Improvement and Innovation (EuroSPI 2006), Joensuu, Finland, 11-13 October. 2-12.

Westerheim, H., Hanssen, G. K. (2005). The Introduction and Use of a Tailored Unified Process - A Case Study. In proceedings of 31st EUROMICRO Conference on Software Engineering and Advanced Applications (Euromicro 2005), Porto, Portugal, 31 August - 2 September. 196-205.

Hanssen, G. K., Westerheim, H., Bjørnson, F. O. (2005). Using Rational Unified Process in an SME - A Case Study. In proceedings of European Systems & Software Process Improvement and Innovation (EuroSPI 2005), Budapest, Hungary, 9-11 November. 142-150.

Hanssen, G. K., Westerheim, H., Bjørnson, F. O. (2005). Tailoring RUP to a defined project type: A case study. In proceedings of Product Focused Software Process Improvement (PROFES 2005), Oulo, Finland, 13-15 June. 314-327.

Hanssen, G. K., Dingsøyr, T. (2003). A Comparison of Automated and Manual Functional Testing of a Web-Application. In proceedings of European Systems & Software Process Improvement and Innovation (EuroSPI 2003), Graz, AUstria, 10-12 February. 1-10.

Hanssen, G. K., Westerheim, H. (2003). Extending Lightweight Postmortem Analyses for Use in Software Process Improvement. In proceedings of 2nd Workshop in Workshop Series on Empirical Software Engineering, Rome.

Westerheim, H., Dingsøyr, T. and Hanssen, G. K. (2002). Studying the User Experience Discipline extension of the Rational Unified Process and its effects on Usability - The design of a case study. In proceedings of First International Workshop on Empirical Studies in Software Engineering, Rovaniemi, Finland. 69 - 74.

Hanssen, G. K., Dybå, T., Stålhane, T., Westerheim, H. (2002). SPI - easy in theory, hard in practice. In proceedings of European Systems & Software Process Improvement and Innovation (EuroSPI 2002), Nuremberg, Germany. 327-336.

Dingsøyr, T., Hanssen, G. K. (2002). Extending Agile Methods: Postmortem Reviews as Extended Feedback. In proceedings of Workshop on Learning Software Organizations (LSO 2002), Chicago, 6 August. 4-12.

Stålhane, T., Hanssen, G. K., Westerheim, H. (2002). Improving the Software Estimation Process. In proceedings of Quality Week Europe, Brussels, Belgium, 11-15 March.

Stålhane, T., Dingsøyr, T., Hanssen, G. K. (2001). Post Mortem - An Assessment of Two Approaches. In proceedings of European Systems & Software Process Improvement and Innovation (EuroSPI 2001), Limmerick, Ireland, 10-12 October. 129-141.