Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Katariina Laakkonen

Contracts in Agile Software Development

Master's Thesis
Espoo, September 10, 2014

Supervisor: Professor Juha Laine
Instructor: Tomi Tuominen M.Sc. (Tech.)

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Katariina Laakkonen

Title:
Contracts in Agile Software Development

Date:	September 10, 2014	**Pages:**	83
Professorship:	Software Engineering and Business	**Code:**	T-76
Supervisor:	Professor Juha Laine		
Instructor:	Tomi Tuominen M.Sc. (Tech.)		

Agile software development is no longer a new innovation: it has been around for over 10 years and a study shows that it is now used in over half of the software companies in Finland. Using agile software development methods also means that contracts need to be defined differently. For example, the traditional approach where the customer defines requirements at the beginning of the project and then leaves the supplier to implement them by themselves is not possible in agile projects.

The interdisciplinary study conducted in this thesis shows the five most critical aspects that need to be taken into account in agile contracting: agile development practices, pricing, change management, early termination of the project, and warranties and liabilities. Traditional contracting, agile ideology, general contracting terms used in Finland, Sweden and Norway, as well as a case contract from a Finnish agile consulting company are evaluated based on each of these aspects.

Despite the popularity of agile development in Finland, the study shows that general contracting terms used are not very agile. This thesis shows the main discrepancies between the Finnish general contracting terms and agile ideology, as well as the work done with agile general contracting terms in Sweden and Norway. The clear conclusion is that the Finnish ICT industry needs proper agile general contracting terms so that each agile company does not have to make up their own contract from scratch.

Keywords: agile, contracts, IT2010, JIT 2007

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Katariina Laakkonen		
Työn nimi:	Sopimukset ketterissä ohjelmistokehitysmenetelmissä		
Päiväys:	10. syyskuuta 2014	**Sivumäärä:**	83
Professuuri:	Ohjelmistotuotanto ja liiketoiminta -	**Koodi:**	T-76
Valvoja:	Professori Juha Laine		
Ohjaaja:	Diplomi-insinööri Tomi Tuominen		

Ketterä ohjelmistokehitys ei ole enää uusi keksintö: ketterää kehitystä on tehty yli 10 vuotta ja tutkimus osoittaa, että yli puolet suomalaisista ohjelmistoyrityksistä käyttää ketteriä menetelmiä. Ketterien menetelmien käyttö tarkoittaa myös sitä, että sopimukset on määriteltävä eri tavalla. Esimerkiksi perinteinen tapa, jossa asiakas määrittelee ohjelmiston vaatimukset projektin alussa, minkä jälkeen toimittaja toteuttaa ohjelmiston itsenäisesti, ei ole mahdollista ketteriä menetelmiä käytettäessä.

Tässä diplomityössä toteutettu poikkitieteellinen tutkimus osoittaa viisi tärkeintä asiaa, jotka tulee huomioida kun tehdään sopimuksia ketteriin projekteihin: ketterien menetelmien vaatimat käytännöt, hinnoittelu, muutoksenhallinta, projektin ennenaikainen päättäminen sekä takuu ja vastuut. Näitä näkökulmia käyttäen työssä arvioidaan perinteistä sopimusoikeutta, ketterää ideologiaa, yleisiä sopimusehtoja Suomesta, Ruotsista ja Norjasta sekä esimerkkisopimusta eräältä suomalaiselta ketterältä konsulttiyritykseltä.

Vaikka ketterä kehitys on suosittua Suomessa, tutkimus osoittaa, että käytössä olevat yleiset sopimusehdot eivät ole kovin ketteriä. Tässä diplomityössä näytetään tärkeimmät eroavaisuudet ketterän ideologian ja suomalaisten yleisten sopimusehtojen välillä sekä osoitetaan, kuinka ruotsalaiset ja norjalaiset ketterät sopimusehdot ovat oikeasti kohtuullisen soveltuvia ketteriin projekteihin. Selvä loppupäätelmä on, että Suomen IT-ala tarvitsee kunnolliset yleiset sopimusehdot ketterään kehitykseen, jotta jokaisen ketterän yrityksen ei tarvitse kehittää omia sopimuksia alusta alkaen.

Asiasanat:	ketteryys, sopimukset, IT2010, JIT 2007
Kieli:	Englanti

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Juha Laine, for the useful comments, remarks and engagement through the learning process of this master thesis. Furthermore, I would like to thank my instructor Tomi Tuominen from Houston Inc. for the instructions and support on the way. I would also like to thank my family and friends for their understanding and support throughout the entire process. Finally, I would like to thank all my colleagues at Houston Inc. for their endless encouragement.

Espoo, September 10, 2014

Katariina Laakkonen

Contents

Chapter 1

Introduction

Agile software development methods are widely used nowadays and most ICT-professionals see them preferable over waterfall development – as we can see from the second section of this chapter. Still, the development of contractual models and general contracting terms have fallen behind the trend, and the models used for waterfall software development do not work well with the agile ideology. Usage of the old-fashioned contracting terms in an agile project can be difficult, but it can also prevent the project from efficiently using the agile practices and both customer and supplier may lose the benefits of agile methodology.

One of the reasons why agile contracting models and contracting terms are falling behind is because customers are used to waterfall contracts and they like the certainty a fixed-everything contract seemingly offers: they pay certain amount of money and get the software they needed. Luckily, also the customers are waking up to the harsh reality of software development: customers cannot know what they want before they see it, change to the requirements is inevitable, and ultimately the customer is the one that suffers when fixed-price project fails. Agile contracting models are needed to balance the risks between both parties and give both the incentive to work together towards the success of the joint project. It is time for cooperative contracts instead of confrontational ones.

1.1 Agile software development

In 2001, 17 software developers published the Manifesto for Agile Software Development (Beck et al., 2001) describing a lightweight software development method. The Agile Manifesto emphasizes the following values:

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

Agile development is an umbrella term for several lightweight agile implementations including Scrum, eXtreme programming (XP), and Kanban. Each has its own terminology and approach to development, but the basic values described in the agile manifesto apply to them all. There are six common features that all of the implementations have: collaboration, code reviews, small teams, short release schedules, time-boxing, and constant testing (Coram and Bohner, 2005). Instead of heavy long-term planning or comprehensive feature documentation, requirements are broken into small tasks, developed iteratively in small, cross-functional teams and integrated and tested continuously. After each iteration the customer gets high-quality, working and tested software.

The highest priority in agile development is to satisfy the customer by delivering valuable software early and continuously, making the working software the primary measure of progress. Deliveries of the software should be frequent, preferably every couple of weeks, and changes to the requirements in order to gain higher customer value are welcomed in any state of the development. (Beck et al., 2001) At the end of each release, the customer gets a working product that can be evaluated, and requirements can then be changed and prioritized for the following releases. Contrary to traditional development, where the set of features are fixed and the delivery date changes, in agile development the release schedule is fixed but the features can change. This time-boxing helps to focus the customer and reduces unnecessary gold-plating. (Coram and Bohner, 2005)

In agile development, rapid response to change is more important than strictly following a plan. The plan is always only as good as it was when initially written, and since in software projects things always change, the plan needs constant updating. Often the changes are so fast that the plan simply cannot be updated fast enough, thus dogmatically following it does not make sense. This does not mean that in agile the code is just a pile of hacks but instead the plan needs to be as lightweight and easily modifiable as possible. For example, Scrum uses post-it notes on a board for a sprint "plan". (Coram and Bohner, 2005)

All agile implementations are highly collaborative, and business representatives and developers should work together daily since the most efficient way of communication is face-to-face conversation (Beck et al., 2001). Informal communication is valued over comprehensive documentation when

constantly spreading information inside the team as well as to all the other stakeholders. Customer collaboration is crucial in agile development, and the customer representative should be "committed, knowledgeable, collaborative, representative, and empowered". The customer is encouraged to actively participate in the development process by giving them the product right from the beginning of the project and the opportunity to change their mind about it whenever necessary. (Coram and Bohner, 2005)

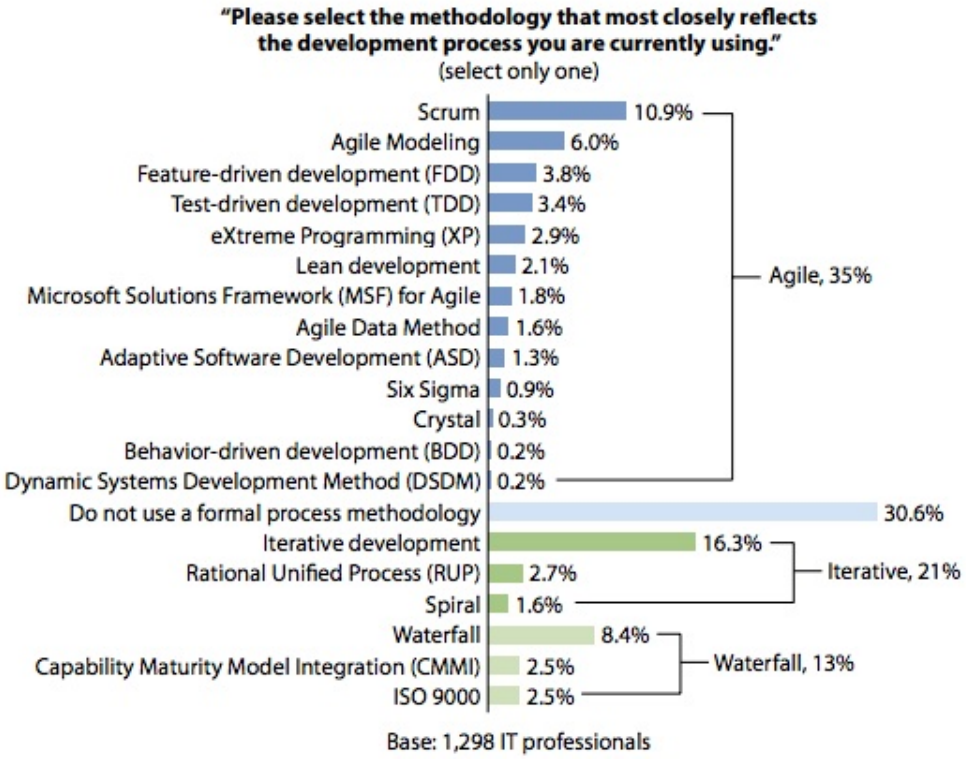
Takki (2002) describes traditional IT-contracting so that it can be assimilated with construction work where precise drawings are necessary right at the beginning of the project, and that the importance of thorough and comprehensive specifications cannot be overemphasized. It is a common Finnish saying that "well planned is already half done", but according to Takki (2002), in IT projects well planned is already almost completely done. He also states that the comprehension and quality of requirements in the contract annex are inversely proportional to the amount of problems that will be escalated during the implementation phase. The most common disagreements are related to whether the project outcome is equal to what was agreed. Another disagreement relates to, in fixed-price contracts, whether some work is part of the original agreement or additional work that is separately charged, and correspondingly in time and materials contract, whether the work has been taken into account in the cost estimate or not.

Takki (2002) thinks that the idea of finalizing plans during the implementation phase will expose the project to endless disagreements and problems. This base for traditional contracting has fundamental discrepancies with agile ideology which values "responding to change over following a plan" and "working software over comprehensive documentation" (Beck et al., 2001). This discrepancy causes challenges when contracting for agile projects in an environment used to doing contracts for traditional software development projects. This thesis will further explain these challenges and introduce possible solutions based on previous research.

1.2 Agile today

Agile ideology has been around for over 10 years and it is no longer a new or exceptional way of developing software but it is starting to be a mainstream development process. Figure 1.1 shows agile adaptation results from Forrester/Dr. Dobbs Global Developer Technographics Survey conducted in Q3 2009. 35 % of the respondents stated that they are using agile development processes, making agile more common than traditional waterfall (13 %) and iterative (21 %) development methods combined.

Figure 1.1: Agile adoption 2009 (West et al., 2010)



Rodríguez et al. (2012) conducted a similar survey in 2011 about agile and lean development usage in Finland using the membership registry of The Finnish Information Processing Association (FIPA) and collecting 408 responses from 200 software intensive organizations. 55 % of the responders reported that they used either only agile or agile and lean methods combined in their development, where 42 % reported that they use no agile or lean methods. About half of the companies had been using agile for 2 years or less, and only 7 % had used agile for more than 5 years. Lean methods were even newer for the companies: over 70 % had been using them for 2 years or less. For the organizations not using agile yet, only 16 % had clear plans to start using agile or lean methods in the near future.

Even more remarkable figures exist, although they are from a company offering e.g. training on agile methods, thus their results may not be completely reliable. The seventh annual "State of Agile Development" -survey conducted in 2012 and sponsored by VersionOne, including 4048 participants mainly from North America (60% of the participants) but also from Europe (27 %), shows that approximately 84% of companies are practicing agile development. This means four percentage point increase compared to 80 % adoption rate in the same survey conducted in 2011. Half of the companies had been practicing agile development 2 years or less and only 14 % had been using agile for over 5 years, and half of the companies used agile in at least half of their projects. The study also shows that agile momentum is up: 83 % of the participants said they are going to use agile in future projects, which is a huge increase compared to the same study in 2011 when only 59 % of the participants had plans to implement agile. (VersionOne Inc., 2013)

This data shows that agile is already a mainstream development process, though it is still quite new practice in most of the organizations. Even though agile emphasizes customer collaboration over contract negotiation, contracts are still needed when suppliers develop software for external customers with agile development methods. Still, general contracting terms at least in Finland are falling behind so companies have to make up their own contract terms for agile projects.

1.3 Case company: Houston Inc.

This thesis uses a Finnish consulting company called Houston Inc. and one of its contracts as a case sample about contracting in agile software development. Houston is a company specialized in agile and lean software development and offers its experienced personnel to implement customers' demanding software projects. Houston also offers consulting about agile project

management as well as agile coaching (e.g. certified scrum master, product owner, test-driven development).

Houston Inc. was founded in 2004 and it employs about 50 agile professionals mainly in Helsinki area. Houston is specialized in projects related to financial services (banking and insurance) but the customers vary from small to large companies in multiple different industries, e.g. telecommunications, journalism and gambling. Many of the projects are implemented using the latest Java EE technologies, but Houston's personnel have competence in wide variety of both back- and front-end technologies as well as mobile development.

For example, Houston has participated in the development of a global digitized invoice solution used around Europe, combining of the information systems of two large life insurance companies, and development of a buying and selling system for stock funds. One larger project included e.g. development of financial background systems, an online banking service, and an interface for mobile applications. One example of a non-financial system developed by Houston is a used cars section of one of the highest-profile classified ads services in Finland used by both private users as well as major car dealers.

The case contract used in this thesis is a quite well working agile contract between Houston Inc. and one of its longstanding customers. It was used in 2012 in a successful project where Houston developed a relatively large, custom, financial service for the customer. The contract tried to follow Houston's agile ideology as well as possible, and both client and supplier were very pleased with it.

1.4 Research questions and methodology

To summarize the content of this thesis, the aim is to answer to following research questions:

- What special issues should be considered when contracting for agile software development projects compared to traditional software development projects?
- How well the general contracting terms in Finland and other Nordic countries fit agile projects?
- What could be improved in the agile contracting of the case company and more generally in Finland?

The first question about special characteristics of agile contracting is answered by conducting a literature study. This study uses some of the industry's basic literature such as the book "Lean software development: an agile toolkit" from Poppendieck and Poppendieck (2003), as well as the most recent articles published about the specific topics of agile development. Also some more informal resources are used in order to find out about the state of the art practices used by agile professionals.

The general contracting terms and conditions for software development used in Finland (IT2010 and JIT 2007) are evaluated based on how well – or how bad – they work with the agile ideology and how they address the most important issues when negotiating contracts for agile projects. For comparison, Swedish and Norwegian terms designed for agile software development are evaluated based on the same special requirements. The parts irrelevant to the topic of the thesis, such as recruitment restrictions or confidentiality, are not covered in the study since the chosen software development methodology is not relevant when handling those kinds of issues in the contract. The intellectual property rights are covered only briefly: the important factor is that in agile projects all the rights to the software should belong to the customer.

Because many companies, including the case company used in this thesis, have found the general contracting terms in Finland insufficient for agile software development projects, they have been forced to create their own terms and conditions. In order to answer to the third research question, a case study is conducted in order to find out how an agile consulting company has handled contracting for agile projects, how it differs from the general contracting terms, and what could the company do better.

Because of the case company being a consultancy company delivering customized software for (mostly) private-sector customers, the analysis is limited to customized private-sector projects when necessary. For example, the parts evaluated from IT2010 and JIT 2007 terms are the ones applicable for delivery of customized software. Also, the complex tendering process and other bureaucracy needed in projects where the customer is part of the Finnish public administration are not discussed in this thesis.

1.5 Content of the thesis

The first chapter of this thesis, the introduction, gives a general idea of agile software development and the ideology behind it. It also shows that, while agile is still quite new approach in many companies, it has became the mainstream software development methodology and the agile momentum is definitely up.

The second chapter of the thesis describes the background of agile subcontracting and contracting in more general level. First, the importance of trust and risk sharing in all contracting and the difficulties and opportunities these two factors offer are addressed. The second part of the chapter explains the different types of agile subcontracting based on the asset specificity. Then the chapter introduces an orientation to contracting that has many agile-like ideas: proactive contracting. Also the general contracting terms and conditions used in Finland (IT2010 and JIT 2007 for projects made for public administration) are described, as well as some general agile contracting terms used in Sweden and Norway. The final section of the chapter introduces the case company of which contracting ideology and a sample contract are analyzed in this thesis.

The following chapters introduce the problems agile contracting has compared to contracting in traditional development models. Commonly used pricing models and their suitability for agile development are evaluated, and also some more extreme models are introduced. Agile also requires e.g. customer participation in the project, thus requiring that the contract somehow covers agile principles. In addition, agile ideology does not match traditional termination clauses used in contracts, and incremental deliveries provide additional challenges for defining warranties for products. For all of these issues, this thesis introduces the traditional perspective, agile viewpoint, how the general contracting terms handle the specific issue and how the issue is handled in the case company.

The results chapter collects all the agile contracting issues and compares the principles the case company uses in contracting with the general contracting terms, as well as with the traditional and agile viewpoints for the issues. Finally, concrete suggestions are given for how to improve the contracting in the case company.

Chapter 2

Background

Trust and risk sharing are the two most basic matters in contracting. Without any trust between the contracting parties, there is no point of starting the project in the first place. The amount of trust in the relationship also influences the distribution of the legal conditions and the ethical principles in contracting, as well as how the risk is shared between the parties. The first section of this chapter discusses trust and risks in contracting.

The second section remarks that agile principles can be applied to different kinds of projects. The degree of social complexity and how the project management responsibility is divided in the project divides agile software development for multiple different subcontracting project types. The third part of the chapter describes an orientation to contracting that has many agile-like ideas: proactive contracting.

The final two sections of this chapter introduce the general contracting terms evaluated in this thesis. The two commonly used general terms in Finland – IT2010 and JIT 2007 – are selected. As a comparison of how agile contracting is done in other Scandinavian countries, the thesis also evaluates the Swedish General regulations for agile projects and the Norwegian Computer Society's contract standards (PS2000). Finally this chapter introduces the case company's "Houston Way", i.e. the way the case company sees agile and project work with customers.

2.1 Trust and risks in contracting

In every transaction where simultaneous exchange is not possible, trust and risk are involved (Jeffries and Reed, 2000). These risks can be either financial or social, e.g. fear of losing reputation (Blomqvist et al., 2005), and both sort and long term risks are important in contracting. Traditionally, customers

think mainly about short-term risks that can be assigned to be the supplier's responsibility with fixed-price contracts. Usually these contracts do not take into account that maintenance of the (usually unsatisfying) system on a time and materials basis will cost a lot for the customer and, if the project is a failure, it will also affect the customers business, thus making the customer undertake the whole long-term risk. Ideally, both short and long term risks should be shared more equally so that both parties have the incentive to make the project a success. (Lichtenstein, 2004)

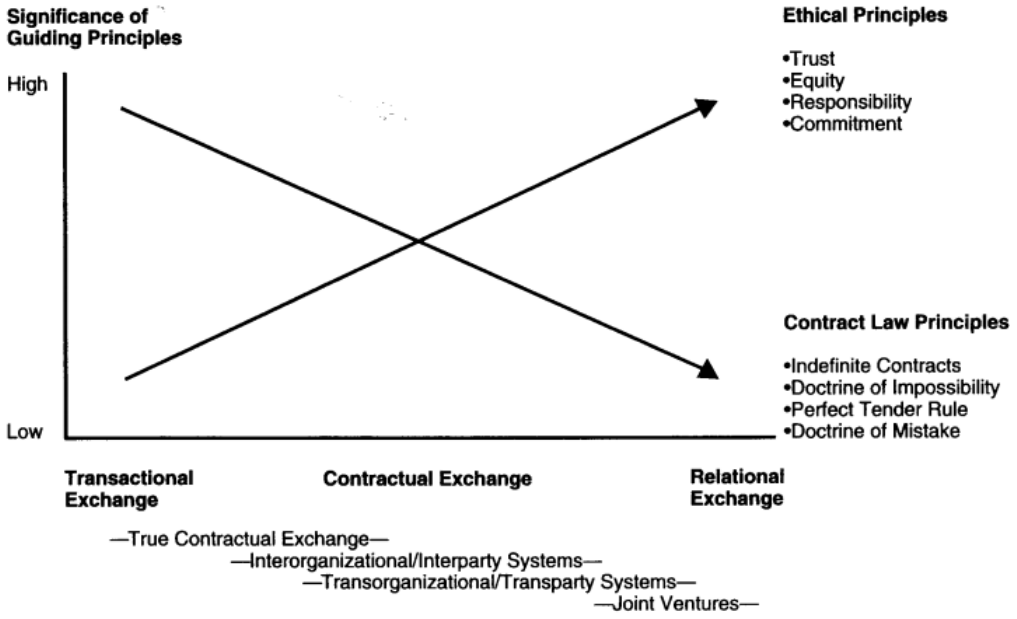
Trust is also often seen as the most important success factor for the business (e.g. Glover, 1994). Still, the conceptual definition of trust is difficult, even though in every-day sense trust is familiar to all. Everyone sees the meaning and role of trust differently and trust is always related to the context, thus making universal definition of trust impossible. (Blomqvist, 1997; Blomqvist et al., 2002) Luckily, some studies still exist that try to enlighten the concept of trust in the business context.

In the business context, an important trust factor is competence, i.e. skills and technical capabilities, which are the necessary base for the whole professional relationship. What comes to accepting a certain level of risk, signs of goodwill, i.e. positive intentions towards the other and moral responsibility, are also needed. Positive intentions also show as proactive behavior and effective cooperation in the partnership. (Blomqvist et al., 2002) With these two factors trust can be defined as "actor's expectation of the other party's competence and goodwill" (Blomqvist, 1997).

Trust is a difficult concept, but crucial in long-term relationships. All circumstances cannot be precisely covered in a written contract and short-term inequities are inevitable, implying that trust towards the other party is needed in all contracting. This also means that if there is no trust at all, it is not reasonable to even try contracting: it is very unlikely that contracting without any trust would lead to a (successful) partnership. On the other hand, if the negotiating partners do trust each other, they will be able to save time and effort in negotiating, agreeing and executing contracts. (Blomqvist et al., 2005) Economists also believe that trust can reduce transaction costs (e.g. Chiles and McMackin, 1996). Both the negative and the positive factors therefore enforce the idea that trust is essential in contracting.

Contracts can also have less importance if trust exists. Figure 2.1 from Gundlach and Murphy (1993) shows the relative significance of contract law and ethics in different types of contractual situations. The horizontal axis has the three exchange forms: transactional, contractual and relational. The vertical axis shows the significance of the guiding principle. The two arrows – one for ethical and the other for contract law principles – show that when moving towards relational exchange, the importance of ethics (e.g. trust and

Figure 2.1: Relative significance of ethical and contract law principles (Gundlach and Murphy, 1993)



commitment) increases where, on the other hand, the importance of contract law (e.g. indefinite contracts) decreases.

On the left side of the exchange forms -axis is transactional exchange i.e. simultaneous transfer of goods where precise rules guide the transaction. When both parties have accomplished their clearly agreed duties, prerequisites for the transaction are met and the exchange can be completed. (Gundlach and Murphy, 1993) This kind of exchange can be e.g. buying a ready made software from a store, where ethical principles like trust are less important.

The other extreme is relational exchange on the right side of the axis, that involves complex transactions linked together over a longer period of time. Even though a contract may exist, contract law is not sufficient to handle this kind of close, long-term relationship. Thus trust and other ethical principles are mandatory when administering this kind of relationship that can be – in its most simple form – e.g. a frequent stay program at a hotel. (Gundlach and Murphy, 1993)

In the middle are the types of contracts that fall somewhere between the two extremes: contractual exchange where both legal and ethical principles are very important. Legal conditions of the contract are the base of the exchange but relying too strictly on the contract may not lead to the best outcome. The formal contract cannot address all the aspects of these kinds of

exchanges that have obligations with complex duties, therefore ethical principles such as trust are equally important in these relationships. (Gundlach and Murphy, 1993)

2.2 Different types of agile projects

Agile software development has multiple different subcontracting project types depending on project management responsibility and the degree of social complexity. The depth of the relationship and level of dependency the customer has towards the supplier vary depending who has the most specific knowledge to give to the project. Different types of subcontracting project types based on degree of social complexity are introduced in figure 2.2, and the effect of asset specificity and allocation of management responsibility in them is described in figure 2.3.

The easiest project types are the two extremities: the customer or the subcontractor has the whole responsibility. If the customer is solely responsible for the project management, the subcontractors also follow the customer's process and are part of the customer's social system. They are practically like rental workers – Pyysiäinen et al. (2003) calls this body shopping. This type leaves the whole risk of the project to the customer, since the supplier has no responsibility for the outcome of the project. On the other hand, if the customer can terminate the contract any time, they also have very little dependency for the supplier. These types of projects are usually developing a software that is close to customer's business and the customer has all the domain knowledge, leaving the supplier's asset specificity low and dependent on the technical skills of the individual members of the development team. (Laine et al., 2011) If the problem is not technically very demanding or the supplier does not have any other asset specificity e.g. related to project management, this leaves little power for the supplier related to pricing.

On the other hand, if the subcontractor has all the responsibility, they usually also follow their own process and work in a "black box" from the point of view of the customer. The supplier is given clear, relatively unchanging requirements for an independent module that they develop using their own project management. The customer does not need to monitor the project continuously, and this type of project resembles more traditional development models where collaboration between the customer and the contractor is not that essential. (Pyysiäinen et al., 2003) This kind of project can be, for example, customization or integration of the supplier's or third party's product for the customer. The supplier has the special knowledge of the software and can take the responsibility of the technical solutions and customer

Figure 2.2: Classification of project types according to the degree of social complexity (Pyysiäinen et al., 2003)

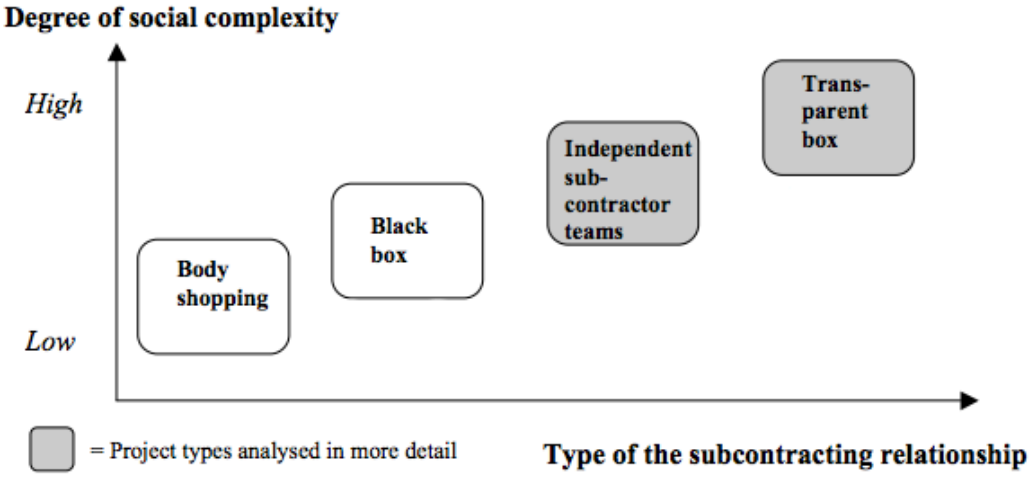
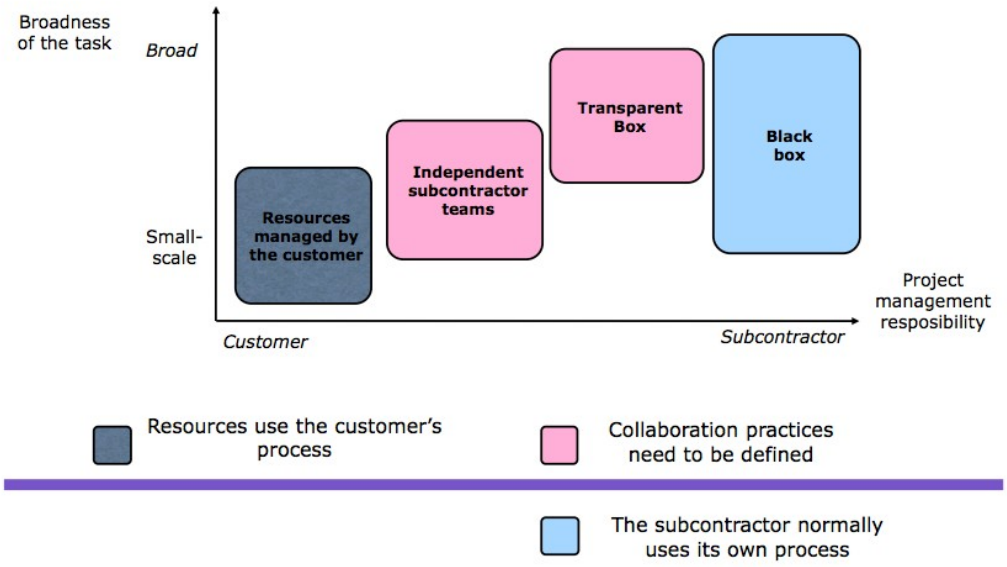


Figure 2.3: Project management responsibility in different project types (Lassenius, 2012)



is more dependent of the supplier. This also gives the supplier more power to raise their prices. (Laine et al., 2011)

The most interesting projects are the ones in the middle in figure 2.3 – independent subcontractor teams and transparent box – where social complexity is high. In socially complex projects collaboration is mandatory, resource management is a shared responsibility, and common processes need to be defined. These projects are also the main focus in this thesis since they offer the most challenges for contracting.

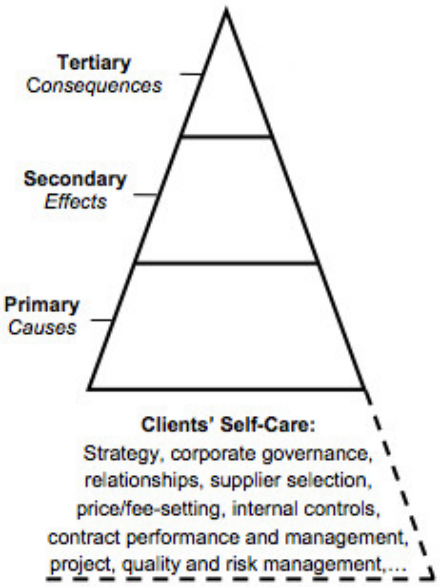
The independent subcontractor project type is a project where the subcontractor organizes its own teams and management for them and is given quite clearly specified, small(ish) tasks during the whole project. Usually, the development does not involve critical interfaces for other sites, reducing the need for active communication during the project. Still, it is important to share sufficient background knowledge about the requirements early in the project and standardize the clarification of the tasks, as well as make it easy to ask clarifying questions about the specifications. It is also important that the customer knows that the subcontractor has understood the requirements properly (instead of being reluctant to ask questions) and that the subcontractor is not left alone with vague specifications. (Pyysiäinen et al., 2003)

Even though this kind of project does not usually contain critical interfaces for other sites, reliability of deliveries between teams has a major impact on the satisfaction: inconsistencies can cause a feeling of risk and uncontrollability, thus reducing the level of trust. The aim in this kind of subcontracting is to build a long-term relationship, making open evaluation critical when seeking optimal work practices. The beginning of the project should contain active communication that lays foundation for the standardized mode of development used during the project, e.g. by introducing different sub-project teams to each other in order to build trust between (at least) the key communicators. (Pyysiäinen et al., 2003)

In the transparent box project type the subcontractor has more responsibility in the project, is involved right from the specification phase, and has its own planning responsibility. This kind of subcontracting is used when requirements are unclear, changes are expected and the customer needs specific knowledge from the subcontractor. The subcontractor has quite much power to decide the specific development solutions they use, and they should be committed to the goals of the whole project. The relationship between the customer and the subcontractor is close and communication is intensive – standardized development practices are defined only to support active communication. (Pyysiäinen et al., 2003)

This kind of subcontracting requires openness between the teams in the

Figure 2.4: Proactive Contracting Levels (Haapio, 2010)



network so that no-one has doubts about others' competence and all are willing and able to help other teams to solve sudden interface-problems. The teams cannot become attached to rigid rules but the development has to be continuously evaluated and continuous negotiations about the dependencies for other modules are inevitable. The frequent and reliable delivery process helps build trust between teams, which then helps the teams solve unpredictable events during the development. (Pyysiäinen et al., 2003)

2.3 Proactive contracting

Traditionally, contracts have been seen as simple one-shot transactions, and contract law has been based on enforcing of binding contracts instead of the idea of long-term cooperation. The goal has been on making one final, perfect contract that does not need flexibility, and if legal disagreements happen, the result is that one party is right (and wins) and the other party loses. In long-standing relationships this kind of mentality is not productive, instead both parties should be willing to aim towards win-win situations where both parties are satisfied – also when conflicts happen. (Pohjonen, 2006a)

Proactive contracting and proactive law are based on preventive law, which was started in 1950 in the United States by Louis Brown with his book Preventive Law. He was a pioneer of the field of preventive law: an

attorney that realized his clients could have prevented most of their problems with good advice from their layers and better planning. Proactive law shares the views of preventive law, but – instead of prevention of legal problems and lawsuits – the focus is on the client, multi-professional teamwork, and promoting successful business. (Nysten-Haarala, 2006) I.e. proactive law emphasizes self-reflection and responsibility: the focus is on how one can create the conditions for achieving goals without unnecessary problems instead of only reacting to something that is already happened or happening (Pohjonen, 2006b).

Proactive contracting and law are developed in interdisciplinary cooperation with lawyers and researchers (Pohjonen, 2006b). It is based on the real-life needs of businesses to start, manage and implement successful relationships and business actions without unnecessary problems. Contracts are only tools that are used to express and fulfill the will of the parties and should be linked to the corporate goals and strategy. The contracting should involve interaction of professionals with different special knowledge (e.g. legal and technical) in order to achieve the real will of each party. (Pohjonen, 2006a)

In proactive contracting, the rules and methods should be planned in a way that will prevent problems and will lead to successful partnership of the contracting parties. One of the main ideas is that problems are best to be solved where they are manifested: different actors such as project managers and developers should recognize the possible causes of problems and fix the reasons before conflicts happen. And if conflicts cannot be avoided, they should be handled in a way that does not need lawyers or court actions, and will keep the atmosphere for further cooperation as good as possible.

Proactive contracting can be described with a three level pyramid as seen in figure 2.4 from Haapio (2010). The primary level is the causes: proactive contracting aims to successful contractual relationships with good communication and coordination, thus eliminating possible causes for problems. The contracting process is done in a way that helps recognize, correct and manage the expectations of each party and form a contract based on these expectations. This makes it is possible to share responsibilities and balance risks and rewards in a clear and realistic way in the contract.

The secondary level is the effects, that aims to minimize risks and harmful consequences when problems do happen. This means that the contract is formed in a way that it is clear how to function when a risk is realized or some other problem occurs. Finally, the tertiary level is about consequences i.e. controlling the conflicts without going to court and minimizing the costs caused by conflicts. This helps resolve conflicts in a constructive way and the contractual relationship can last longer. (Haapio, 2010)

This pyramid can be interpreted so that proactive contracting is not an isolated function but needs to be part of the client's business processes such as corporate governance and project management. Another interpretation of the figure is that it is not enough for the lawyers to deal with the legal issues in the secondary and tertiary levels, since actions on those levels only minimize the harmful effects and the client does not profit from the remedies or damages. Without success in the primary level and the client's self-care, all other effort is wasted. Haapio (2010)

In practice, the aim of proactive contracting is to increase mutual understanding and produce better contracting methods (Pohjonen, 2006a). The best way to achieve this, according to Haapio (2010), is that proactive lawyers and business managers together "establish processes, practices, checklists, and templates that help the entire business team take care of the key issues in a systematic way". This way everyone knows what they need to do and how to do it from the first time. It is also impossible in big organizations for the lawyers to prepare all the contracts and deals, thus the sales personnel should have the tools and knowledge to recognize opportunities and prevent problems on a daily basis. For example, various contract templates, project models and interactive internet programs exist to show how the contracting and business process should proceed (Pohjonen, 2006a).

Siedel and Haapio (2010) describe a medical analogy to explain proactive law. The traditional reactive management sees lawyers as an emergency function, like a patient who seeks doctoral treatment after catching malaria. The preventive aspect of proactive law tries to prevent legal harm by adding e.g. risk allocation clauses in the contract, similarly to a person using pills and a mosquito net to prevent getting malaria. The promotive aspect of proactive law tries to change the legal concern into business concern or business opportunity – so instead of malaria medication and netting, the root cause i.e. the mosquitoes are eliminated by draining swamps. If the root cause is eliminated, also the contract negotiations can be moved from the risk management systems, such as the malaria medication and netting, to something more constructive.

As an example of proactive contracting in practice, Haapio (2010) gives Metso Contract Library and Standard Templates for Purchase Contracts. Metso has business operations in over 50 countries and hundreds of Metso employees around the world buy services and products for Metso. Metso's contract templates and conditions cover the most important issues that should be taken into account in contract negotiations, and they can also be used by people who seldom participate the contracting. Metso has also included contracting training as part of their procurement training.

Another example is the UK Treasury Public Private Partnership website

that has guidance and standard contracts for contracting for the public sector. The model terms and conditions with related guidance are free to download, and they have been used by e.g. the defense, health, and education sectors. The aim is to standardize the Private Finance Initiative (PFI) contracts in the UK, in order to achieve commercially balanced contracts and help the public sector to meet their requirements and get the best value for money. The guidance's main objectives are to promote common understanding of the main risks in a PFI project, enforce consistent approach and pricing in similar projects, and reduce the costs and time needed for negotiating similar contractual issues over and over again. (Haapio, 2010)

2.4 General contracting terms

In Finland, IT2010 terms and conditions are widely used general terms and conditions for contracting in traditional information technology projects. They have been prepared in cooperation between the Central Chamber of Commerce of Finland, the Finnish Software Entrepreneurs Association, the Finnish Association of Purchasing and Logistics LOGY, the Federation of Finnish Technology Industries and the Finnish Information Processing Association. Still, they can not be found suitable for agile development and are not commonly used in agile software development projects.

IT2010 terms and conditions are a draft of contract clauses that can be used in domestic deliveries between suppliers and customers in the field of information technology. They are prepared to make execution of agreements easier and to reduce contracting cost. The idea is not to remove the need for negotiating the content of the contract but to help parties focus on certain, important issues that should be considered in contracting.

IT2010 YSE general terms and conditions are meant to be the starting point of any contract dealing with the sale or licensing of information technology products or suppling information technology services. In addition, a suitable annex (EAP, EJT, ELH, ELT, EOY, ETP or EVT) of special terms should be selected. For an agile software development project the suitable annex would be EJT special terms and conditions for deliveries of data systems and customized software. If there are discrepancies between the EJT special terms and YSE general terms, the special terms and conditions are applied.

The Advisory Committee on Information Management in Public Administration (JUHTA) is a committee set up at the Ministry of Finance of Finland to plan cooperation in information management, make studies and reports, and give recommendations for the public administration. One of the

recommendations of JUHTA is that "the procurement of IT products and services by central government agencies; by state enterprises; by central government institutions and funds; and by municipalities and municipal federations" should use Terms and Conditions of Government IT Procurement i.e. JIT 2007. (JIT 2007, 1.1) In addition to the JIT 2007 General Terms and Conditions, a suitable annex is used when applicable – within the context of this thesis it would be Special Terms and Conditions for Procurement of Customized Applications i.e. JIT 2007 - Applications. The annex always takes precedence over the general terms if there are discrepancies between the two. (JIT 2007, 1.2)

This thesis will address the issues IT2010 terms and conditions would have if applied to agile projects. For comparison, also the similar problems found from JIT 2007 terms and conditions used in the procurement of IT products and services for the public administration are introduced. Also, a more suitable general terms for agile software development from Sweden and Norway are examined. Swedish General regulations for agile projects were published by the IT and telecommunication companies in 2012. The Norwegian Computer Society's contract standards i.e. PS2000 terms were developed under the research Project 2000 organized by NTNU (Norwegian University of Science and Technology) and SINTEF (Norwegian: Stiftelsen for industriell og teknisk forskning, the largest independent research organization in Scandinavia).

Both Swedish General regulations for agile projects as well as Norwegian Computer Society's contract standards (PS2000) are intended to be used with projects that have no pre-approved specifications but the project requires quickly response to changing conditions during the implementation. The planning, implementation and monitoring of the project is guided in cooperation with the supplier and the customer, and the success of the project is dependent on how well the parties interact and control the project jointly.

2.5 Agile in the case company

The base for Houston Inc.'s way of doing agile is that we cannot predict: people writing specifications to the system before the project starts do not know what the end-user will actually need. We ask about the customer's needs but – because of human nature – we only get rational answers that are not true. For example, Nokia had a touch screen mobile device a century ago but users told them that they do not want it and Nokia stopped the development. Another problem is that developers know too much about the product and make decisions based on that, leading to software doing wrong

things or right things in a wrong way. Developers really need feedback from the actual users – the sooner the better. (Teikari, 2012)

Clear concept for the new product is crucial for the project to succeed. Vague ideas need to be shaped into viable and technically feasible business concepts that have measurable goals. This is usually not easy to do alone but external expert can help by asking the questions one does not dare or know to ask. For this purpose, Houston offers Houston Accelerate -workshops to brighten the concept before the project. (Teikari, 2012)

Also agile projects need some specifications, which in traditional development methods can take months to define. Houston way of doing specifications is Boot Camp: a two day intensive workshop to specify and plan the project. During the Boot Camp, the project stakeholders get together and a trained professional will squeeze the requirements and their priorities out of them. The end result of the Boot Camp is a prioritized, work-estimated release plan for the project. This plan can be used as the base for the contract, but agile principles should not be forgotten: this is just the first guess about the future software and will become more precise and change when the development starts. (Teikari, 2012)

Chapter 3

Development practices in contracts

Traditionally, software development has had only minimal customer collaboration: customers have been needed only at the requirements elicitation phase at the beginning of the project and during the acceptance testing at the end of the project. On the contrary, agile development is more customer-centric. In agile one of the main priorities is to produce maximum amount of value for the customer as early as possible in a form of a working software. This kind of development requires an active customer during the whole project, including implementation phase, which adds new requirements for the contracting.

This chapter addresses the requirements for agile contracting by first giving an insight on how software development have been seen traditionally and then explaining the agile practices that must be taken into account in contracting. Then the general contracting terms are evaluated in order to find out whether they address these issues or not. Finally, the agile development practices the case company uses in contracts are introduced.

3.1 Traditional development

Traditionally, software development has been rationalized and engineering-based, grounded in the principles of hard systems thinking. The approach assumes that problems are fully specifiable and for every problem exists a predictable and optimal solution. The basis for predicting, measuring and controlling problems during the development is in comprehensive, in advance planning. (Nerur et al., 2005)

In traditional, process-centric software development the primary focus is on highly optimized and repeatable processes. By continuously measuring and refining the processes, sources of variations in the processes can be identi-

fied and eliminated. This kind of traditional software development is driven by control and planning accomplished by a command and control style of management. Project manager has the authority and a role of a planner and a controller. (Nerur et al., 2005)

Customers are, of course, important in traditional software development, but their main role is during the early and late phases of the development process. They participate specifically in requirements elicitation and analysis, budget and contract negotiations, and acceptance testing. (Grisham and Perry, 2005) What comes to other activities during the development phase the customer participation is minimal: it contains some limited interactions with the development team during the implementation phase.

The traditional development where emphasis is on comprehensive, in advance planning and customer involvement during the implementation is not needed gives the supplier free hands to execute the project using whatever working methods they prefer. No cooperation methods exceeding regular reporting of the project's state to the customer is needed to be stated in the contract. Also the contract has only minimal requirements for the customer to e.g. participate in the requirements elicitation before the implementation starts and perform the acceptance tests after the project has been delivered.

3.2 Agile principles

In agile software development projects active customer engagement is crucial for the project to succeed (Nerur et al., 2005). In agile projects, the system is developed iteratively based on what the customer needs, which means the customer has to dedicate some of their time to the this collaborative development. With uninterested, overly busy or otherwise unavailable customer the supplier cannot deliver a system that actually fulfills the customer's needs (and the requirements in the contract), thus agile contracts should issue the customer engagement to the project.

The customer's role in any software development is to decide "what to build". In agile, this means that the customer is integrated part of the development team and works on-site with the team. The customer will write the user stories and discuss them with the rest of the development team, is responsible for the business decisions (e.g. prioritizing the user stories), and regularly tests the software to confirm it works as expected. This means that the customer must know the domain well enough to be able to do the necessary decisions. (Martin et al., 2004)

If agile software development, the customers are expected to be "collaborative, representative, authorized, committed, and knowledgeable" (Boehm

and Turner, 2003). In addition, the customer must be able to handle multiple tasks at the time and have managerial support. This all combined to a one person means that a suitable on-site customer must be a valuable member of the customer's organization, and the organization may not be willing to donate such an asset to the software development team. The organization may see that using a valuable employee's time to a software development project perceives little business value, and requiring that may have negative effects on the relationship with the customer. One possible solution to that can be that a knowledgeable engineer or manager acts as the supplement of the customer who is available only part-time. (Grisham and Perry, 2005)

In Scrum, the key representative of the customer is called the product owner and he is responsible for communicating the customer needs to the development team and keeping the product backlog in shape during the project. The contract should require that the customer nominates a product owner who will develop, prioritize and maintain the product backlog as well as participate to the required meetings such as sprint plannings or review meetings. Since the product owner is crucial for the project, the contract should also require that the product owner dedicates a reasonable part of their time and effort to the project and is available to respond the development team's questions in a decent time. (Bird&Bird, 2012)

When using Scrum – or any other agile process model with short iterations – the contract should specify the agreed duration of each iteration (i.e. sprint). The duration of each sprint should be short (e.g. 2–4 weeks) and the duration of a individual sprint should not be extended under any circumstances. If all the agreed items in the sprint cannot be completed, the unfinished items should be put back to backlog to be implemented in the future sprints. (Bird&Bird, 2012)

Another key aspect in agile is to deliver working software to the customer after each iteration and have the software as the primary measure of progress. But how to determine this when the deliverables are small and missing lots of the features the final product needs to have? This can be achieved by defining an agreed "Definition of Done" – ideally already in the contract – that can contain e.g. items like: all code is reviewed and tested (acceptance tests as well as non-functional tests such as performance tests), all code follows agreed coding standards and have been refactored when needed, and all necessary documentation has been completed. (Bird&Bird, 2012)

Hoda et al. (2009) also gives a worst case option to use with customers who cannot be convinced to use agile technologies: do not tell the customer about the agile process. The supplier is then usually forced to work under a fixed price contract and the customer is unaware of the internal agile practices used by the development team. This is probably still better than using

waterfall methodology and the customer is arguably happy to see the project results during the project. Still, this does not guarantee that the customer is available during the project.

3.3 Practices in the general contracting terms

IT2010 terms and conditions as such do not state what kind of process should be used in the project: EJT special terms and conditions say that work related to the project should be performed using the supplier's working methods (IT2010 EJT, 5.7). So the supplier is free to use waterfall process or agile methodologies or anything they want, just some basic practices are defined in the terms. One practice is that based on the EJT special terms and conditions, the supplier has to report the progress of the project in writing at least once per month and, if the pricing is not a fixed price, also about the time used for the project (IT2010 EJT, 6.1). This kind of written, forced reporting does not fit well with agile ideology where active customer collaboration is enforced.

IT2010 EJT special terms and conditions also state that the project should have a steering group containing representatives from both parties which will organize the cooperation between the parties and supervise the implementation of the project. This group should meet at least after each delivery phase and whenever either of the parties requests a meeting. (IT2010 EJT, 5.1) This is actually a sound approach for both traditional and agile development projects.

Also JIT 2007 terms define a similar steering group for the project that manages the cooperation of the parties and supervises the project. The group meets whenever requested and at least after each delivery phase. The supplier's project manager is responsible for reporting to the steering group about the state of the project. In addition to that, both parties should have named contact person that monitors and supervises the fulfillment of the contract and communicates information regarding implementation of the contract to his own organization and to the other party. (JIT 2007, 4)

On the contrary to the two Finnish general terms, the Swedish general terms are based on iterative, agile development process where software is developed in consecutive time boxes (or in sprints if Scrum terminology is used). Parties should jointly agree how the time boxes form releases (e.g. is a time box equal to a release) in order to archive the goals stated for the project. Continuous cooperation is crucial in order to plan, implement and monitor the incremental work. This will take time from the customer, thus the terms state that the customer must e.g. ensure that decisions are

taken continuously during the implementation. The terms also require the customer's representative to be qualified and competent and have necessary knowledge about the client's business. (General Terms for Agile Projects, 3)

Before each iteration, content for the time box is agreed and the contents should not change during the time box. At the end of the time box, the supplier will deliver the work for the customer, who will then – without a delay and still inside the time box – verify that the contents correspond to the plan and agreement in general. This verification is the basis for planning the next time box: if some items are not accepted (or not implemented), they are moved to later time boxes. (General Terms for Agile Projects, 4)

Norwegian PS2000 terms are also based on Scrum process where the roles required by Scrum are combined with the organizational model used in traditional software development projects. Both parties have project managers, and project is steered by the steering group containing representatives from both parties (e.g. the project managers). In addition to the steering group, the contract should also specify a team-level cooperation between the parties. It is also possible to form specific work groups to evaluate, for example, specifications, prototypes and test results. (Laine et al., 2011)

An important factor in the PS2000 terms is the reasonable risk sharing between the customer and the supplier. Thus one of the customer's responsibilities is to perform a risk analysis as a part of the requirement analysis performed before the implementation phase. The supplier will then complete the analysis. The result of the analysis is an uncertainty matrix where each of the risks is bared by the party more capable of handling the risk factors and minimizing the damages. (Laine et al., 2011)

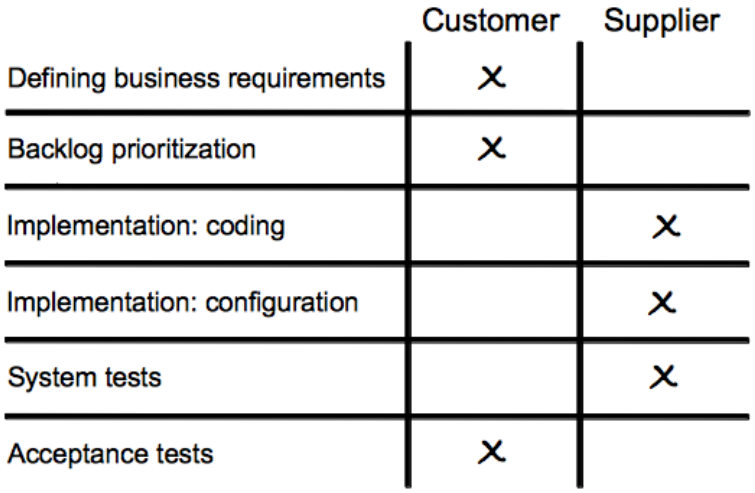
3.4 Practices in the case company

Agile development needs more engagement from the customer than traditional software development. In the case company, Houston Inc., the minimum requirement for the customer is to have full-time product owner for the project. The product owner has to have the required knowledge about the business and the customer's clients, and is e.g. responsible for prioritizing the requirements and accepting the features. (Teikari, 2012)

To prevent any misunderstandings, the responsibilities of each party should be clearly stated in the contract. The contract can e.g. have responsibility matrix that describes the responsibilities of both the customer and the supplier. Simplified example of this kind of responsibility matrix is shown in figure 3.1. (Teikari, 2012)

The development team and the individual members of it, as well as the

Figure 3.1: Example of a partial responsibility matrix



chemistry between supplier's and customer's personnel, are extremely important in agile development projects where the supplier works within the customer's premisses and with the customer's personnel. Thus a client's CEO tells in the book of Teikari (2012) that it is essential for the customer that the members of the development team can be replaced just by notifying the supplier.

The case company sees that every software project needs a business steering group that steers the project with business decisions. The group will guide the project in long-term so that project will stay in line with the customer's overall business. This group should contain members from both parties in order to keep the cooperation close and open. (Teikari, 2012)

The sample contract notes the importance of customer engagement and uses responsibility matrix to ensure the customer takes care of the duties necessary for the agile process. For example, the customer is responsible for having a reachable Product Owner, as well as for the business requirements in the product (and sprint) backlog and prioritization of them. The customer also conducts the acceptance tests for each delivery, and is obligated to nominate a member to the steering group.

The sample contract also notes the importance of good chemistry between the supplier's development team and the customer's personnel. Thus the sample contract gives the customer the right – with a reasonable cause – to request replacement of the suppliers resources in the project. The supplier must then immediately and without additional costs replace the person with another, equally competent one.

Chapter 4

Pricing models

In some industries, calculating a price for a product is easy: just calculate the costs (e.g. material and labor costs) and add some margin. Unfortunately, in software development calculating the costs (and therefore also the price) beforehand is not that easy. Usually the product requirements are not thoroughly known beforehand and they change during the development, which is also the base and motivator for agile software development.

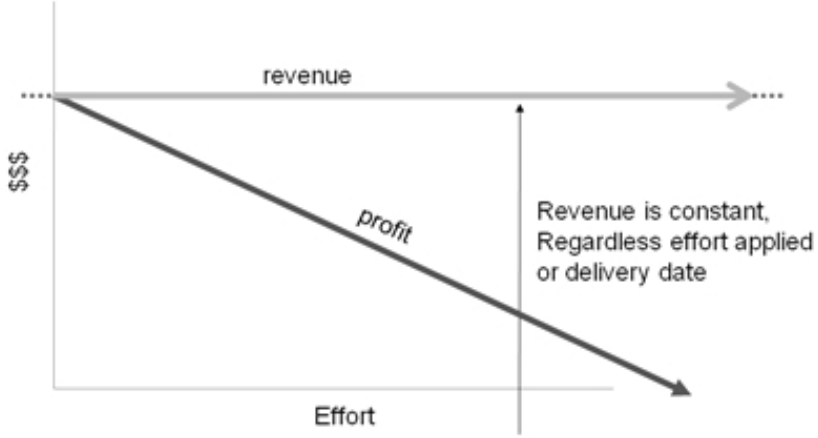
This chapter introduces three commonly used pricing models: fixed price, time&materials and target cost pricing. These can all be used in both traditional and agile software development projects – the second section of this chapter evaluates how well those models work with agile projects. The second chapter also introduces some more innovative pricing models that agile development enables.

The third section explains the pricing models used in the general contracting terms. IT2010 and JIT 2007 terms rely on using penalties where the Swedish and Norwegian agile terms both use some kind of shared pain and gain models. Similarly, the ideology of the case company as well as the clauses used in the case contract – both introduced in the final section of the chapter – rely strongly on the persuasion that bonuses work better than penalties.

4.1 Traditional pricing models

The three commonly used pricing models are fixed price, time&materials and target cost pricing. The first feels like low-risk model for the customer since the supplier bares the responsibility to finnish the project in time but may actually cause high long-term costs for the customer. In the second model the customer bares all the risks and, when applied to waterfall development,

Figure 4.1: Fixed Price, Fixed Scope (Stevens, 2009)



customer is tied to the supplier who has no incentive to keep the costs down. The third option is an attempt to share the pain and gain between the parties and is recommended e.g by Takki (2002).

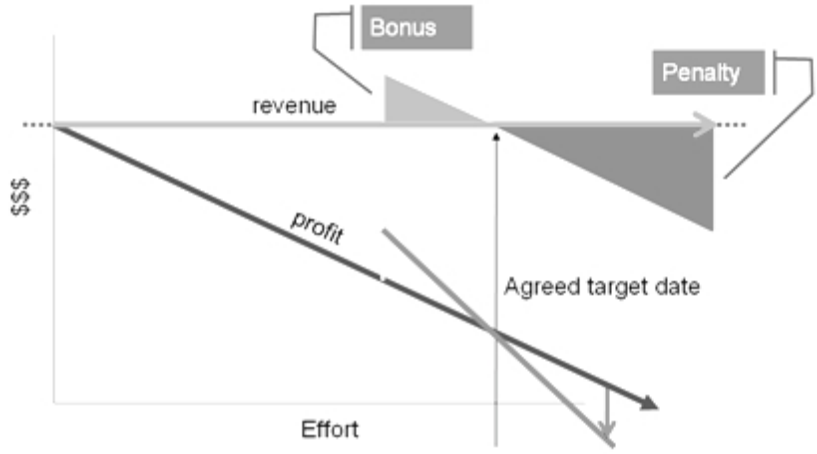
4.1.1 Fixed price

Customers are used to fixed price, scope and time -contracts that feel simple and safe: the customer pays a certain amount of money and gets the system they ordered on an agreed date. Unfortunately, software projects are seldom that simple, thus fixed price contracts are not popular among the agile practitioners. Also for traditional software development, Takki (2002) suggests that fixed price should be used only if the requirements are well known before the project or the requirements are first gathered using hourly rate and fixed price is agreed only for the implementation phase.

The customers also like fixed price contracts since they can move the risk to the vendor. Though this might work for the sort-term risks, in long-term, if the project fails, the customer is the one who will suffer. Instead of always trying to transfer the risk to the other party of the contract, it should be evaluated who is the most competent party to handle the risk. For technically complex but quite stabile problems the supplier is probably in a good position to manage the risk and should assume it. However, in many software development projects the problem is uncertain or changing, thus the customer has a better position to manage the risk and fixed price contracts should be avoided. (Poppendieck and Poppendieck, 2003)

As can be seen from the figure 4.1, fixed-price contracts involve high risk for the supplier since they have to estimate the cost of the system before

Figure 4.2: Bonus / Penalty Clauses (Stevens, 2009)

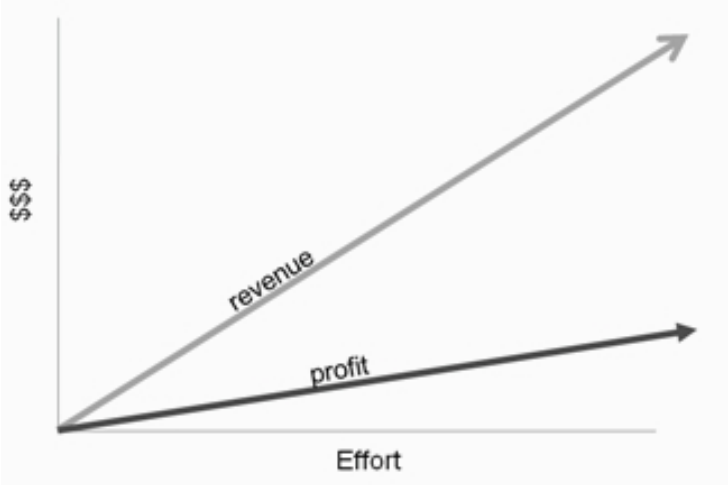


any work has been done and without complete understanding of the domain and the complexity of the problem. An experienced supplier will include this risk in the bid but less competent vendors are likely to underbid. Since the supplier bidding the lowest price is usually selected, the customer may end up selecting the supplier that has the lowest understanding of the complexity of the problem and who has the highest probability to fail the project. (Poppendieck and Poppendieck, 2003) Also, the requirements for the software almost always change during the project, which in fixed-price projects mean ordering additional work from the supplier. Low bidding in the fixed price part of the project can then be compensated with high prices for the extra features, possibly making the project very expensive. (Takki, 2002)

In order to stay within the price and the scope of the project, suppliers can save effort by reducing the quality of the system e.g. by writing lower quality code and doing less testing. Though this may help the project finnish in time, it will also increase the future costs for the customer since the maintenance of a low quality software with high technical debt is not cheap. (Arbogast et al., 2010) The supplier may also need to aggressively protect their interests by delivering as little as possible in order to keep the schedule and budget, causing the customer to get less than they really want (Poppendieck and Poppendieck, 2003). Of course, this can also be seen as a business model: the supplier knowingly delivers a system that does not meet the true needs of the customer and then takes the profit from changing the unsatisfactory system to an useful one (Arbogast et al., 2010).

If the customer has real incentive for early completion of the project, bonus and penalty clauses where the supplier gets a bonus if the project

Figure 4.3: Time and Materials (Stevens, 2009)



finishes early and will pay penalty if it is late can be used with almost any of the pricing models. This combined stick and carrot -model used with fixed price pricing is shown in the figure 4.2. This bonus can be e.g. 50 % of the hours that go under the target, and should also be tied to the project schedule so that the supplier will not get the bonus if the project is finished late with low amount of hours used (Takki, 2002). When using this kind of pricing, it should be clear and transparent that the customer will economically benefit from the early completion of the project, otherwise the customer may misuse the penalties to get a cheaper product the longer the project takes. (Stevens, 2009)

4.1.2 Time and Materials

Time and materials contract is a contract where the customer pays the supplier for the time used for the project and for other expenses. It has the lowest risk for the supplier but, on the other hand, it does not take away the risks but just transfers them to the customer. Time and materials contract is good for dealing with uncertainty and complex problems but it also offers less security for the supplier since they cannot be certain how long the contract will last. Still, these contracts are often considered to be good deals for the supplier as long as they last. (Poppendieck and Poppendieck, 2003) This supplier incentive for time and materials contract is illustrated in figure figure 4.3. For the customer, the incentive to use time and materials is that changes to the project are easy to make (Takki, 2002).

This kind of open ended contract may be hard to sell to the customers

and require lot of trust that the supplier will use the charged time properly. Actually, with time and materials contract the supplier has no reason to work efficiently since they make more money if the project takes more time (Poppendieck and Poppendieck, 2003). Eckfeldt et al. (2005) describes the mindset of many customers with a quote from a IT Manager: "T&M is favorable for relatively smaller budget projects or when working with a consultant in which you have a high degree of confidence. For larger projects or when working with unknown consultants we typically prefer fixed price, at least initially, to maintain budget certainty. Once we are confident in the consultant's ability to deliver as promised and we have a well-defined set of requirements we are more willing to shift to T&M."

Time and materials contract can increase contract transaction costs if the customer does not have sufficient trust towards the supplier. The customer may feel the need to have extensive control over the costs and go through every cost and define wether they are acceptable or not. (Poppendieck and Poppendieck, 2003) Even when using time and materials pricing in traditional software development, the contract often has an agreed budget in order to reduce the risk for the customer. If this budget is clearly exceeded, the customer can terminate the project, but naturally has to pay for the work already done. (Takki, 2002)

Time and material contracts can be used with traditional software development projects, but they cause a risk to the customer since the customer is tied to the supplier who has no incentive to keep the costs down. (Poppendieck and Poppendieck, 2003) Even thought the customer might have the option to terminate the project when the agreed budget is used, in practice this is not usually possible. If traditional software development project is not finished, the customer will end up having an incomplete software that they cannot use before the original supplier (or someone else) will finnish it.

4.1.3 Target-cost

Both fixed price and time and materials contracts reinforce the traditional thinking that customer-supplier relationship is adversarial. One of the parties is always protecting themselves as the other is encouraged to self-serving behavior, which limits the possibility to greater success for both parties. What is needed is in the middle of the two extremes: the project risk should be shared and both parties should have a reason to drive towards the joint success of the project.

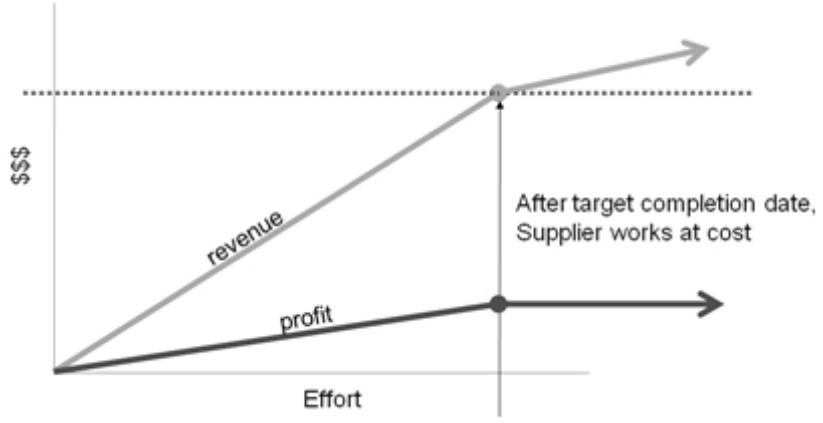
The base for a target-cost contract is a situation where both parties recognize that the project details are uncertain and defining them needs mutual work, but also meeting a target cost is found very important in the project.

Figure 4.4: Target-cost shared pain&gain (Arbogast et al., 2010)

Target cost	Target profit	Target customer payment	Actual supplier cost	Adjustment	Actual customer payment	Actual supplier profit
1,000,000	150,000	1,150,000	1,100,000	+60,000	1,210,000	110,000
1,000,000	150,000	1,150,000	900,000	−60,000	1,090,000	190,000

$$\begin{aligned} \text{Adjustment} &= (\text{ActualCost} - \text{TargetCost}) * \text{CustomerShare-\%} \\ \text{CustomerPayment} &= \text{TargetCost} + \text{TargetProfit} + \text{Adjustment} \end{aligned} \tag{4.1}$$

Figure 4.5: Target-cost fixed profit (Stevens, 2009)



This requires common effort from both the supplier's technical personnel as well as the users of the system, and both parties need to be truly committed to meet the target cost. To make this happen, in target-cost contracts the whole cost of the system is a joint responsibility of the customer and the supplier: if the target cost is exceeded both parties will pay more and, if the actual cost is less than the target, the benefits are also shared. (Poppendieck and Poppendieck, 2003)

In order to succeed, the contract must enforce the customer to stay in line with the the target cost with the feature requests and the supplier must have an incentive to work under the target cost. This can be achieved in various ways. Poppendieck and Poppendieck (2003) suggest that the customer incentive is a clause that if the actual cost is significantly higher than the target, new cost-sharing negotiations are held.

Poppendieck and Poppendieck (2003) also suggest two models for the supplier incentive: cost plus fixed fee or profit not to exceed. First means that the supplier's profit is a separate fee payed in addition to the costs after the project is completed, meaning that the supplier gets a higher profit margin if the project meets (or goes under) the target. Second one, also shown in the figure 4.5, means that the supplier's profit is included in the target cost, and the supplier will reduce their rates and work at cost if the target cost is exceeded. Takki (2002) suggests this kind of pricing for traditional software development projects: if the target-cost is exceeded, the supplier will charge e.g. 50 % of the normal hourly rate which will enforce the supplier to meet the target but will not bankrupt them.

The figure 4.4 from Arbogast et al. (2010) shows calculations done when using a model where price is adjusted using an adjustment percentage (which is agreed so that the customer carries 60% of any cost difference). The formulaes vary, but this simple example uses as simple formula as 4.1. If the costs are higher than estimated (firs row), the supplier gets less profit but the customer also pays more than the target cost. On the other hand, when the costs are less than estimated (second row), the supplier gets higher profit and the customer pays less than the original target payment.

Eckfeldt et al. (2005) describe a bit different approach to target-cost contracts. They first estimate the total hours needed with some contingency percentage (about 10% for existing customers and up to 25% for new customers) and then negotiate a daily or hourly rate based on the costs and a profit percentage of the target-cost. The profit is then set as a fixed amount for the project and the customer is billed according to the agreed hourly rate.

Increasings in the project scope are divided into three categories: fixes are changes that need to be done in order to meet the requirements in the existing stories, clarifications are changes that are found necessary for the

system to work correctly, and enhancements are additional feature requests. For fixes and clarifications the supplier will work at cost but for enhancements the original estimated hours are increased and the fixed profit is recalculated. Thus the profit also increases. (Eckfeldt et al., 2005)

4.2 Agile pricing

Pricing is particularly difficult in agile contracts since it is difficult to put a fixed price tag for a fixed set of features – which is usually what the client would want to have. Selling the agile ideology is quite easy in in-house development where the supplier is well trusted and the pricing is not that important factor. But for a consulting company, more work and knowledge regarding the pricing is needed in agile contracting.

Fixed price is the "old-school" pricing method commonly used in traditional software development, time and materials is probably the simplest and easiest pricing method for agile development, and time and materials is a good compromise between the two for agile projects. They are all commonly used in contracts – both traditional and agile projects – but agile enables other types of pricing models too.

4.2.1 Fixed price agile

Because of the complex nature of software projects and the inability to predict beforehand what the customer wants, agile ideology emphasizes response to change over following a plan. The previous section showed that it is obvious that fixed price, scope and time -contracts are not ideal for neither the supplier nor the customer since simply following the plan is not usually possible. Still many customers – especially government related – require using all three boundaries.

In agile projects the idea is to develop the software in short iterations implementing always the next most important features in each iteration. After each iteration the system has the features most important for the customer and the project can be terminated when the money runs out. However, when working under a fixed price contract, this approach can be risky for the supplier since it is hard to get the customer to agree that the project is finished when all the money is used and the project backlog still has unfinished requirements. This forces the supplier to protect themselves with detailed specifications and strict change control. (Poppendieck and Poppendieck, 2003) This also means that during the project there will be constant negotiations about whether some task is part of the contract or billable extra work (Takki,

2002). In practice, the customer has to commit to a scope too early in the project and when they realize their true needs, changes to the project will be expensive (Eckfeldt et al., 2005).

Franklin (2008) describes how they succeeded in an agile project using the full fixed price, fixed scope and fixed schedule expectations. They had a backlog list of items to be developed during a year and for each sprint they selected a part of those items to be developed, tested and delivered to the customer for approval. They did not receive any payment for work until the deliveries were approved by the customer and any errors in the deliverables were reason for disapproval. The probability of disapproval was high when the deliverable was a part of a functionality that was supposed to be finished in the future sprints, and it was highly important to document these cases carefully so that each sprint was approved by their own merits.

Fixed constraints required additional time and management resources for contract modification processes. Also all the elements of the done criteria (e.g. data validation, documentation, performance, usability) had to be taken into account in the schedule and estimates in order to keep the costs and schedule fixed. The scope included and the customer had to be aware of the non-tangible deliverables that do not come directly from the product backlog but were part of the full contract expectation. These items are, for example, training, support, reporting, meetings, maintenance, and administrative tasks (including the ones related to the contract modifications). (Franklin, 2008)

Franklin (2008) succeeded in "completing functionality faster but with less flexibility" with fixed constraints, which in their opinion can lead to increased maintenance effort but helped the customer to get the production software they needed. One of the key success factors they mention was the first hand collaboration with the users during the contracting phase in order to ensure good user story awareness to estimate the size of the themes in scope. Also the expectations had to be documented clearly and managed with open communication with the product owner. In order minimize scope creep, the feature complexity had to remain within "what we need and no more", and they also had "contingency buckets" for both scope and schedule changes to address the unknowns.

4.2.2 Time and materials in agile

Time and materials is a common pricing model in agile contracts. In time and materials contracts the customer has the fear that the supplier will not use the time properly and has no incentive to be efficient. Agile methodology helps the customer to trust that the supplier is using the money invested in

a proper way since after each iteration, the supplier has to deliver value for the money spent in a form of a working software. The customer evaluates the results and has the right to terminate the contract if the results are not what they should be. After the termination, the customer will still get the value for the investments up to that time. Thus time and material contracts can well be used with agile software development projects. (Poppendieck and Poppendieck, 2003)

4.2.3 Target-cost and agile

Agile development is easy to sell to the developers but sometimes hard to sell to the customers who are used to traditional development models. Customers usually appreciate the clear benefits of agile – such as getting a new version of the working software every couple of weeks – but agreeing on time and materials pricing might be more difficult. On the other hand, fixed price agile is not what the supplier will want to agree because of all the uncertainty it causes. Target-cost contracts offer the customer some assurance that the costs will stay within the budget without causing unfair risks to the supplier. Shared pain and gain also drive towards the common success of the project.

4.2.4 Fixed price per unit of work

One of the agile principles is that working software is the primary measure of progress (Beck et al., 2001). This is also the base for agile fixed price per unit of work (UoW) pricing model, where UoW is related to running, tested software features. These models go with various names and ways to estimate a UoW: e.g. price per story point, price per function point, and price per feature point. (Arbogast et al., 2010)

In theory, this kind of pricing follows closely the agile ideology by being delivery- and value-oriented: by defining UoW so that it is related to the value created to the customer the customer pays for the value received. However, usually e.g. a story point is more of an estimate of the size or effort rather than a value measure. Therefore, the pricing is not truly based on the value impact for the customer. (Arbogast et al., 2010)

At least in theory, it is possible to define a UoW in terms of business value impact: Gilb (2005) suggests a value-impact price model which uses impact estimation tables. Also function points are closer of being a value measure: they represent a logical piece of software and measure the amount of functionality the customer gets – in comparison to the story points that calculate the effort used. Function points are also more more standardized than story points: they have a set of rules and guidelines to support the

technique. Since function points do not measure effort, the project will have cheap and expensive function points (from the point of view of the supplier), which may cause disputes. (Radford and Lawrie, 1996)

Using function points for pricing reduces the risk for the customer since they know they get certain amount of logical pieces of software for a certain, fixed price. For the supplier, this offers security against the changing requirements: changes are automatically priced based on how they affect the function points. This is also efficient way of change management since the effect each change has to the budget is directly visible from the function points. (Forselius, 2013)

Arbogast et al. (2010) suggest two possible ways to determine the fixed price per point: 1) use average price from several previous projects, or 2) use customized amount specified just for the certain project. In the latter case, the customer pays the average point value for a few iterations (or some other pricing model like time&materials is used) and the costs are tracked during those iterations. Then, a specific price per point is determined based on those costs and some profit margin.

When using this kind of pricing in a contract it is essential to specify a clear and common framework for defining an unit. For example, story points (i.e. relative effort points) do not have any global meaning and their meaning need to be defined for the specific project. On the other hand, function points are relatively unambiguously defined and can be calculated with certified function-point analysts. Arbogast et al. (2010) Still, they are not generally well understood by the customers nor the suppliers, and usually an external, independent consultant is needed to e.g. size the project and resolve disputes (Radford and Lawrie, 1996).

4.3 Pricing in the general contracting terms

IT2010 EJT special terms and conditions do not state anything about pricing of the project but it is (loosely) defined in YSE general terms. They state that, if nothing else is agreed, the price is determined based on the the supplier's price list on date of the order (IT2010 YSE, 4.3). This does not really say much about the pricing and in practice something else – perhaps agile-compatible – should be agreed. On the other hand, the terms do have lots of specific regulations e.g. on how possible currency exchange should be handled using mid-rate quoted by the European Central Bank and how travel time should be charged if a necessary journey exceeds 60 kilometers (IT2010 YSE, 4.4, 4.5), which may be details the parties should be able to agree themselves if needed.

Instead of any bonuses or other positive reinforcement methods to keep the project in schedule, IT2010 EJT special terms define compensations for delays. If a part of the delivery is late from the agreed schedule, the amount of compensation for the damage is 0.5 percent of the price of that part for each beginning week. The maximum amount for the compensation is 7.5 percent of the price of the delayed part of the delivery. (IT2010 EJT, 9.1, 9.2) It is a very traditional way of thinking that penalties will help to keep the project in schedule. In practice, suppliers can calculate the original price in a way that takes the penalty into account making the delayed delivery the "default case" for the supplier.

Also JIT 2007 terms and conditions do not state anything about the pricing model that should be used but the pricing principles that are in the terms relate strongly to the supplier's general price list. For example, if the prices change so that the list price is lower on the date of delivery than it was when the tender or order was made, the price valid on the date of delivery should be used (JIT 2007, 13.2). These kind of pricing principles do not really seem relevant for modern software development where the customer orders customized software from the supplier and not a ready made product from the store.

In a similar fashion than IT2010, also JIT 2007 offers penalties instead of bonuses in order to keep the project in schedule. If the delivery is delayed because of the supplier, the supplier will pay a penalty of 0.5 % of the sales price of the delayed product for each beginning week the deadline is exceeded. The maximum amount for the compensation is 7.5 percent of the price of the delayed part of the delivery. (JIT 2007, 18.2) Actually, these delay terms are quite identical between the two general terms and conditions used in Finland, even though penalties are often seen an insufficient way of keeping the project in schedule.

The Swedish general regulations offer only basic pricing model if nothing else is agreed: the supplier is payed hourly fee based on the supplier's current price list. Also possibility to use fixed price for iteration is mentioned. (General Terms for Agile Projects, 14.1) Both are quite fine defaults in agile projects, although something else is probably usually agreed to make the contract more intriguing for the supplier.

The Swedish general regulations for agile projects also have an annex for the profits and risk sharing, which offers more interesting aspects to project's pricing model. Instead of using simple hourly pricing where the supplier as though rents workers to work under the customer's supervision, more complex model is used to share the risks and gains of the project. The risk of not meeting the requirements in the defined timeframe (or reaching the target price before the system is ready) is shared by defining a reduced

hourly rate for the time needed to fulfill the requirements. Naturally, this reduced price does not apply if the supplier can show that the requirements are not met because of the customer's actions – or lack of them. (General Terms for Agile Projects, Risk&profit annex 2.1)

In addition to the "stick approach" of the reduced price, the profits and risk sharing annex defines gain compensation model as a carrot for the supplier. The annex defines that if the customer terminates the project early and all the requirements are met at that point, the supplier is still entitled to a compensation for the full planned length of the project (or the target price, if so agreed). (General Terms for Agile Projects, Risk&profit annex 3.1) This can be significantly more effective for achieving success faster than the traditional model of using sanctions for delayed deliveries.

In the Norwegian PS2000 terms the project pricing is defined using target cost with lower and upper limits. If those limits are exceeded or undercut, the costs or savings are shared between the parties. Fixed price is still used in the final, approval phase. (Laine et al., 2011)

4.4 Pricing in the case company

Typically contracts include plenty of penalty clauses but very rarely any bonuses for outstanding work. Christ et al. (2012) conducted a study showing that in employment contracts workers responded better to bonuses than penalties: the employers getting a base pay plus a bonus used more effort and trusted their supervisor more, leading to increased productivity. On the other hand, those who were getting penalties had less trust towards their supervisor and used less effort for their work. In fact, a base pay plus a bonus indicates that after a certain (base) point, additional effort from the employee is valuable whereas a base pay minus a penalty indicates that after a certain point additional effort is not valuable (Luft, 1994).

The basis of pricing in the case company is to use not just penalties but rather bonuses in order to archive success fast. The idea is simple: the supplier offers a reduced hourly rate (e.g. -5 %) for the base price but gets an additional bonus when all the commonly agreed targets are reached. This approach is backed-up by a client's ICT purchasing manager who tells in the book of Teikari (2012) that they have found bonuses more efficient than penalties in important projects.

Pricing in the sample contract is defined as a hourly rate with additional bonus per invoiced hour if the supplier meets all the success criteria defined for the release. This evaluation is done by the steering committee after each release. They evaluate the success of the release agains the predefined

acceptance criteria of the backlog items and the functional scope, and if all of the criteria are met, the supplier is entitled to the bonus. So instead of the "stick approach" of penalties the contract relies on giving the supplier a positive incentive to finish the agreed items.

Chapter 5

Change management

Practically all software development projects will need changes during the implementation. Traditionally, these changes have been seen as a bad thing which is clearly contradictory to the agile ideology. Agile development emphasizes rapid change management and welcomes changes to the requirements even late in the development.

This chapter introduces the traditional way of handling changes and then points that even in agile projects some kind of change management procedures are necessary. This chapter also shows that all of the general contracting terms researched as well as the case contract use quite traditional ways of handling changes in the projects.

5.1 Traditional change management

In traditional contracting, the contract often specifies a steering group containing same amount of representatives from both parties and which is authorized to make changes to the project. The changes must be unanimous and they must all be documented as contract annexes, and all of them should contain evaluation of whether they impact the cost or the schedule of the project. The group may not have authority to make changes that affect the schedule or the cost of the project but those changes must be done by the customer's management. This may help keep the costs under tight control but also ultimately makes changes to the project more difficult. (Takki, 2002)

5.2 Agile change management

Agile ideology emphasizes "responding to change over following a plan", meaning that changes to the requirements are welcomed even late in the

development (Beck et al., 2001). Rapid response to change is build into the agile process since the development is done in small iterations and refactoring is a frequent practice (Eberlein and Leite, 2002). There are several reasons why using much time for the initial requirements analysis is seen as a waste in agile development: 1) the requirements are often unstable, 2) the business domain may be stable, but the technical details are unknown and can affect requirements (e.g. the high cost of a technically difficult feature may be a reason to drop the feature); and 3) the customer does not know clearly what they want before they see it (Ramesh et al., 2010).

In agile development, before the development starts, only high-level requirements analysis is done. During that analysis, the development team acquires high-level understanding of the critical features of the system. These requirements are not meant to be complete but they are just the base for planning the initial release cycle for the project. As the project goes on, more features are added, the existing ones are discussed in more detail and modified when needed, and some of them are completely removed. All the requirements are frequently prioritized based on the business value they will give for the customer. (Ramesh et al., 2010)

While this build-in response to change is definitely important, this does not mean that agile projects do not need any change management. Central thing to managing requirements is requirements traceability, meaning that there must be descriptions of the requirements as well as something to link the requirement to its source and the implementing code. Using only the code as the sole documentation of the process is not usually enough. (Eberlein and Leite, 2002) For example, simple techniques such as user stories can be used to describe the requirements in high-level, and more formal documentation is seldom created. Those user stories can then be used as a base for further discussion about the implementation details. (Ramesh et al., 2010)

In small-scale projects, a great product owner might be able to handle the agile project independently: first identify the right product (continuously identify and prioritize the right requirements from all stakeholders involved) and then governance the project towards maximizing the business value. But in large projects, even though the product owner is well familiar with the business domain, he cannot estimate correctly the business value of each and every feature and handle all the expectations of all the stakeholders. The product owner will end up being in trouble with prioritizing items in the backlog, thus needing further help to get better understanding of the stakeholders' needs. (Ktata and Lévesque, 2009)

A larger project will require a steering committee that supports the product owner by analyzing and prioritizing the needs of the stakeholders, and offering the product owner visibility on business expectations and sources of

value. This will help the product owner to understand the reasons behind the expectations and solve conflicts in them, track the value gained and make decisions based on the changing business context, reduce problems caused by miscommunication, and identify and report business opportunities back to the steering group. Having a steering group will also help sharing the same vision of the software (and business) between all the teams involved. (Ktata and Lévesque, 2009)

5.3 Change in the general contracting terms

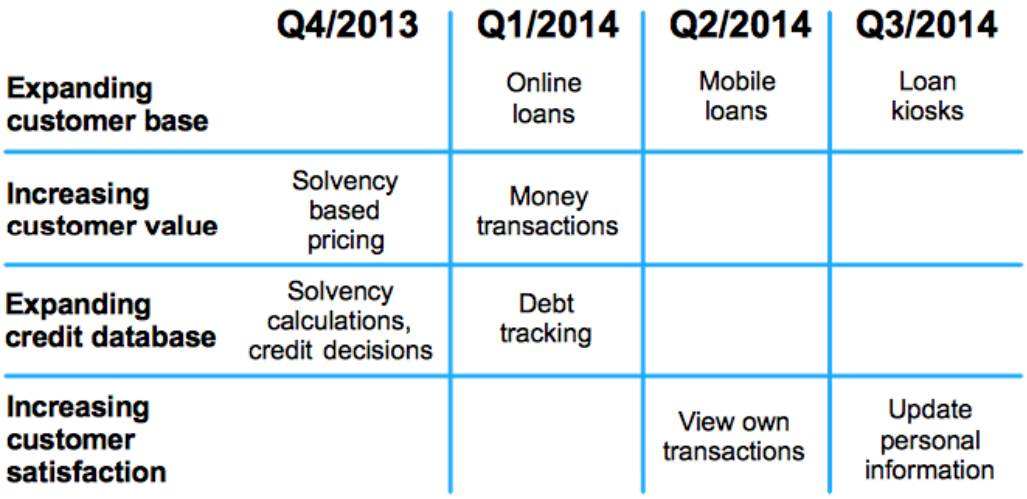
IT2010 EJT special terms and conditions state that the steering group is responsible for handling any change requests to the project. The group has to evaluate the possible effects the changes have on the schedule, price and other terms and conditions, and agree a common decision about the request. (IT2010 EJT, 5.9) All the changes and amendments must be documented in writing (IT2010 YSE, 15.3).

JIT2007 terms do not state any specific change management procedure but only require that all changes are agreed in writing using a mutually agreed procedure. Also the possible effects the changes have on the schedule or price must be documented. If the agreed changes require extra work from the supplier, the parties must agree beforehand, again in writing, that the supplier may charge the extra work from the customer. (JIT 2007, 12)

Changing requirements in the Swedish general agile terms are handled in the profits and risk sharing annex. If the customer wants to change the requirements, the parties will agree to adjust the requirements so that the scope does not increase. Another solution is to change the plan so that the new requirements fit into it or, if the contract uses target price, change the target price to take the new (or removed) requirements into account. (General Terms for Agile Projects, Risk&profit annex 4.3) The contract should state who has the authority to do the changes (General Terms for Agile Projects, 5.3).

In the Norwegian PS2000 terms, small changes that do not affect costs or schedule can be done without any additional bureaucracy. Bigger changes are gathered together and the project steering group discusses them in specific, beforehand agreed, checkpoints. Even though the steering group handles the changes, the customer has the final decision authority. After the decision, the decided changes can then be implemented during the next iteration. (Laine et al., 2011)

Figure 5.1: Roadmap for a loan system (freely translated from Teikari, 2012)



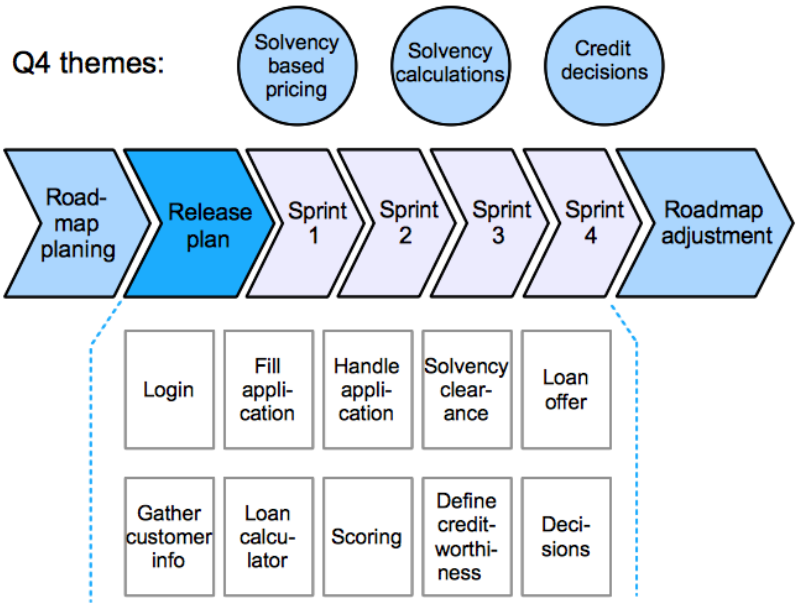
5.4 Change management in the case company

In the case company changes to the software projects are seen inevitable, since it is simply impossible to predict what the customer wants. In order to track the changes, the steering group uses a project roadmap to plan and steer the project. The roadmap – example shown in the figure 5.1 – is a strategic tool to see what themes are implemented in each quarter of a year. A year is a suitable time to be planned in the roadmap – there is no point to even try to predict the future beyond that especially in software projects. (Teikari, 2012)

Business management, the project's product owner and the development team should go through the roadmap after each sprint to check wether they are still in track with it. If not, changes to the roadmap need to be done by the steering group. The roadmap should not change all the time, since a roadmap that changes after every sprint does not steer the project anymore nor justify features with business reasons. Themes can, of course, be re-prioritized to be implemented in different quarters, but adding or removing them should be an exception. (Teikari, 2012)

More concrete tool for project management is the release plan – example shown in the figure 5.2 – which is done about every four sprints (2-3 months). It contains the features needed to implement the themes from the roadmap. The features are small enough to be easy to move and prioritize e.g. in the sprint planning meetings, and the product owner should have the authority

Figure 5.2: Release plan for a loan system (freely translated from Teikari, 2012)



to do the prioritization based on the business needs. The steering group's duty is to make sure the product owner knows how the business is evolving.

In the case contract, all the changes to the agreement are handled in the the steering group according to the change management procedure. The procedure requires that all the changes are recorded as written attachments to the contract and their scope as well as impact to the schedule and cost are evaluated. Minor changes to the product, i.e. changes that do not have significant effect on the scope, should be done by the supplier without additional fee upon customer's written request. Other additions and changes to the work requested by the customer are discussed in the steering group where their effect on the scope and schedule are evaluated early enough before the work is started. The customer can also decide to reduce the scope of the product, and the possible effects on the schedule and costs are again agreed in the steering group.

Chapter 6

Contract termination

A huge difference between traditional contracting and agile ideology is how the contract can be terminated. Traditionally contract termination has been seen as highly unwanted situation where one of the parties is responsible for breaching the contract, thus being guilty for the termination. This is contradictory to the agile ideology where early termination can just mean early success – or failing fast so that minimum time and effort is wasted on a project that is bound to fail anyway.

This chapter explains the strict traditional termination conditions that do not work well with agile projects. Then, the agile idea of an easy, early termination without a clause – and with or without an additional termination compensation – is introduced. The general contracting terms follow the two models in a quite expected way: the two Finnish terms use the traditional strict termination clauses where the Swedish and Norwegian terms follow the agile model of easy termination. Finally the chapter introduces the agile termination clauses used in the case contract.

6.1 Traditional termination conditions

Traditionally contract termination has been seen as the last resort after contract has been breached and when all other means have been used. Contract can be terminated only if the other party fundamentally breaches the contract, or it is obvious that fundamental breach of the contract will happen and the other party cannot give acceptable guarantee that the contract will be fulfilled. Sometimes it is also required that before the contract can be terminated, the claimant have to give a warning to the breaching party and contract can be terminated only if the breach continues (or reoccurs) after the warning. For example, if the delivery is delayed, the claimant may give

reasonable extra time when the contract needs to be fulfilled or the contract will be terminated. Lack of negligence is not sufficient reason to continue the contract after the breach but, on the other hand, clear negligence can widen the right to terminate the contract. (Hemmo, 2003)

The Finnish Sale of Goods Act §25.1 and §39.1 define the fundamental breach of a contract in a way that it is "of substantial importance to the buyer and the seller knew or ought to have known this". This is often connected to what would have happened if the other party would have known about the breach when signing the contract. The breach can be seen to be fundamental if the party would have not signed the contract at all if knowing about the breach, instead of being satisfied with e.g lower price or other compensation. (Hemmo, 2003) Hemmo (2003) also notes that this can lead to too wide termination right since often the mere inconvenience caused by the contract breach would have been enough for the claimant to abstain from the contract.

Requiring a fundamental breach of the contract is, as such, quite massive requirement for the contract termination, but in some situations the threshold for termination is even higher. For example, when the breaching party has high financial dependency to the claimant (e.g. in franchising) or is it not possible to return the partial performances already delivered (e.g. partially build building), the threshold for termination is high. It can also be seen that after the lowered price, compensation and/or other correction to the breach (for example fixing the broken product) the claimant does not have enough reason for terminating the contract. Also, if the overall financial goals are reached, some individual breaches may not justify termination of the contract. (Hemmo, 2003)

What happens after the contract is terminated is also extremely important in traditional software development where the customer cannot use the unfinished software, and the customized software has little or no value for the supplier since it is custom made for the needs of the specific customer. The contract can be seen in a way that it is either about the work done for the customer (usually time and materials contract) or about the produced product (fixed-price contract). The evaluation is based on the contract as whole – not solely on e.g. the contract heading or pricing model used – and is affected by the heading, pricing model, factual content, the customer's influence to the development process and management. For example, the more the customer influences the development of the product and the more rights the customer gets to the produced product, the closer the contract is to service contract (the first option). Also, if the customer is already using parts of the software, it usually leads to service contract -interpretation. (Takki, 2002)

The first contract option usually means that substitutes are not returned

after the termination. The customer will end up having nonworking software, and the supplier may have to reduce their prices but no big harm is done for their business. The second option, on the other hand, is an extreme pressure mean for the customer since the substitutes are returned when the contract is terminated. This means that customer gets their money back and the supplier will get nothing valuable in return. (Takki, 2002) Neither is a fair or a good solution, and fortunately agile projects do not have this problem. In agile projects the customer gets a working, valuable, high-quality software regardless of if or when the project is terminated, and can take on another supplier to finnish the product if they wish so.

Possibility to terminate the contract after any phase of the project is definitely the most optimized solution for the customer but not so good for the supplier since the reallocation of the resources is usually not possible without any delay. Takki (2002) suggests that the contract should contain a price for early termination of the project and that termination would not require any cause. This can be, for example, 50 % of the expected contribution margin of the remaining project since most of the costs can usually be avoided after the termination. This will encourage the customer to engage to the contract and gives the supplier a certain profit even if the contract is terminated early. Both parties will also avoid unpleasant legal showdown and neither will lose one's face or be marked as the guilty one.

In addition to termination of the project, traditional contracts often include contractual penalty which is a certain rule that justifies the customer to get compensation without having to show any damages caused by the breach. For example, if the delivery is delayed, the compensation can be some percentage of the total project cost for each day or week limited to some maximum percentage. This can be seen as incentive for the supplier to finnish the project in time but, on the other hand, it sets an upper limit for the responsibility and when the maximum compensation is reach, the supplier has no incentive to try to finnish as fast as possible. The supplier might also deliver poor quality product in order to avoid the contractual penalty since fixing the faults later may be cheaper. (Hemmo, 2005) This kind of clauses can also be used in agile contracts, but using a better pricing model that drives both parties to success early is more preferable way.

6.2 Agile termination principles

Agile ideology emphasizes change to the point where terminating the project early, at the the end of any iteration, should be possible. Since in agile projects the customer has a working system after each iteration, early ter-

mination does not necessarily mean the project has failed: it may have just reached success faster than initially estimated. At least from the customer's point of view this is indubitably a positive event. (Arbogast et al., 2010)

In agile projects, the customer's financial risks are reduced since the project can be cancelled quickly if it is noticed that the project is not what the customer wants or will not meet the budget (Maurer and Hellmann, 2013; Hoda et al., 2009). This is quite the opposite to traditional contracting where termination of the contract is extremely difficult and, if the termination is found groundless, it may turn the situation around and the terminating party has to pay damages to the other party (Hemmo, 2003). This is also a trust issue: the agile ease of early termination reduces the trust the customer has to have towards the supplier since they are not strictly tied together by the fear of costly contract violations. When starting a new project this is especially intriguing: in the worst case the customer loses one sprint and realizes that the project cannot be implemented at all, or has the opportunity to change the supplier if the initial choice turns out to be a wrong one.

For the customer the ideal termination model in an agile contract is naturally to allow the customer to stop the project at the end of any iteration without any penalties. For the supplier, on the other hand, this can be unappealing if they have dedicated people to work for the project for he next two years and the project is unexpectedly terminated much earlier. Because of the risk for the supplier, the termination clauses can contain early termination penalty for the customer. This penalty will reduce over each iteration. (Arbogast et al., 2010)

The key to negotiating termination clauses in agile contracts is the fact that after each iteration the customer will have a working system and both parties have constantly a clear vision of the state of the project. (Arbogast et al., 2010) This makes it possible for the supplier to offer the customer a contract with an early termination without any penalties, which can also be used when selling the agile ideology to a customer accustomed to more traditional project models. Possibility to terminate early easily is in fact a huge benefit compared to traditional models where early termination can be very difficult and expensive.

6.3 Termination in the general contracting terms

In IT2010 terms the termination of the project is handled in the YSE general terms. The terms state that early termination is possible only if the other

party breaches – or it is clear that the other party will breach – the terms of the agreement in a matter that has substantial importance. However, cancellation is only valid if the other party has not remedied the happened breach within a reasonable time-period (at least 30 days) or provided an acceptable guarantee of the fulfillment of the agreement. (IT2010 YSE, 11.4, 11.5) This approach follows quite closely the traditional contracting model and makes terminating projects early practically impossible. It is also clearly in discrepancy with agile ideology where it is desirable that the customer can terminate the project after any iteration, and often early termination means only that project has succeeded faster than initially estimated.

Another factor in IT2010 terms makes early termination of the project difficult: the intellectual property rights. IT2010 EJT special terms for customized software say that, unless otherwise agreed in writing, the intellectual property rights to the system will belong to the supplier and the customer will have a license to use the system. The customer does have the right to do changes to the system – or order them from a third party – by using the possibly developed open application programming interface (open API) or to any open source software or customized software included in the system. Still, the supplier gets a non-exclusive, paid-up and royalty-free license to use any of these changes made. (IT2010 EJT, 10) The supplier will deliver the source code of the customized software as well as API interface description when the project is delivered (IT2010 EJT, 7) but, because of the traditional waterfall development model used as the base of IT2010 terms, the customer does not have anything delivered when the early termination occurs.

Also JIT 2007 terms have strict, non-agile cancellation policies: termination of the project is possible only if fulfillment of the contract is delayed more than 4 months due to force majeure, usage of the software infringes the intellectual property rights of a third party, or one of the parties has materially breached his contractual obligations. In the last case, if the breach is reparable, the contract can only be terminated if the breach has not been corrected in a reasonable time after notification of the breach. Things that always constitute a material breach are e.g. that the software is so faulty that it is unusable for at least 30 days during the warranty period, delivery of the software is delayed because of the supplier for at least one third of the agreed time (or at least 14 days for short projects or 4 months for projects lasting over a year), or the customer's payment is delayed by more than 45 days. (JIT 2007, 22)

Also in JIT 2007 terms the intellectual property rights to the system will belong to the supplier. When the project is ended, the customer has the right to do changes (or order them from a third party) to the customized software, get control and user rights of the machine-language version and source code

of the customized application. (JIT 2007, Customized Applications 7) But, when the contract is terminated premature, the JIT 2007 terms say that both parties will return the substitutes they have already got during the project (JIT 2007, 22), meaning that customer gets its money back and supplier will get the probably useless software back in return.

On the other hand, the Swedish general regulations give the customer the right to terminate the contract after any iteration, before the next iteration begins, without a cause. (General Terms for Agile Projects, 7.1, 19.1) This is highly beneficial to the customer and truly follows the agile principles but may cause harm to the supplier who have reserved resources to the project. Thus, the general regulations also state that the supplier has the right for compensation for the resources reserved for the next iteration if they cannot be reallocated to other work. The customer can then decide whether these compensated resources should work for the project for the next iteration or not. (General Terms for Agile Projects, 7.2) Regardless of the early termination, if their payment obligations are fulfilled, the customer has the right to use the results the same way they would have been if the project had been fully completed. (General Terms for Agile Projects, 19.5)

Also in the Norwegian PS2000 the customer is entitled to terminate the contract after each iteration. When the project is terminated, the supplier gets compensation for the work done for the project as well as compensation for the direct costs the termination causes for the supplier. In addition to that, an additional compensation for the termination can be set. Based on the application guidelines of PS2000, this compensation is on average 4-6 % of the total value of the project. (Laine et al., 2011)

6.4 Contract termination in the case company

The case company's ideology as well as the sample contract follows closely the agile ideology what comes to termination of the project: the customer can terminate the project after any delivery phase without a cause. The customer is not liable to any sanctions relating to the termination. When the contract is terminated, the supplier is entitled to compensation for the work done before the termination but no additional compensations are payed. In return, the supplier will deliver all the material related to the product, e.g. source code and documentation, to the customer. The customer has the right to use the material however they want.

Chapter 7

Warranties and Liability

Agile ideology emphasizes "working software over comprehensive documentation" (Beck et al., 2001). This can be misinterpreted so that in agile development no documentation outside the source code is needed. Having no documentation than the source code will cause serious problems at least when defining warranties for the system, thus some kind of lightweight documentation is needed also in agile software development projects.

This chapter takes a look to the warranties and liabilities in traditional Finnish contracting law and practice. Next section points out what additional difficulties iterative development, where features can be accepted in separate phases, can cause to defining warranties. Then the warranties and liability clauses in general contracting terms are investigated, and finally the case contract's warranty terms are explained.

7.1 Warranties in traditional contracting

Based on the Finnish Sale of Goods Act §32, the reclamation period starts when the buyer should have noticed the fault in the product. It is also possible to define in the contract that the supplier will guarantee that the product will have certain features, which decreases the need to notice the faults, or define a period when the customer needs to complain about the found faults (Hemmo, 2005).

In practice, in traditional software development, the finished software is delivered to the customer who then tests it during the acceptance phase. During the acceptance tests the customer will report all bugs found from the system, usually refusing to accept the delivery before the program is perfect. This can make the project last forever, thus the contracts often include a term that "good enough" (e.g. ready for production use) software has to be

accepted and found faults are then fixed without additional cost during the warranty period. (Takki, 2002)

After the complaint the supplier has per se the right to correct the faults on their own expense if it does not cause significant harm to the customer. If the corrections have to be ordered from someone else, the costs can be charged from the supplier, and if the faults are significant and cannot be fixed, the customer has the right to terminate the contract. (Hemmo, 2003) Because of the nature of software programs, the contract can also include a clause that the supplier will do their best during the warranty period to fix all the bugs found during the acceptance tests as well as on the warranty period, but will not guarantee that the program will be completely bug-free. (Takki, 2002)

If there are major faults in the program after the warranty period, the customer can claim a rebate. The customer can also withhold part of the payment, e.g. 15 % of the price of the product, to be payed only after the errors are fixed during the warranty period. This will work as supplier incentive to fix the errors in order to get the customer to pay the last consideration. That is not usually mandatory since if the supplier acts like a scoundrel what comes to fixing the errors in the software, it will probably not stay in the business for long. (Takki, 2002)

The contractual division between the work done for the customer and the produced product that Takki (2002) discusses also affects the liabilities. Commitment to the result (produced product) requires that the exact, concrete outcome is accomplished, where the commitment to the work done requires that the work is performed in properly and carefully but the outcome is not thoroughly defined. If the supplier does not accomplish the outcome when the contract is about the produced product, the customer has the right to claim for a new accomplishment, fixing the error(s) or reduced price. On the other hand, if the contract is about the work done, previously listed consequences are only possible if the supplier has not been performing properly. (Hemmo, 2003)

Traditionally, the negligence (or lack of it) is not a factor when evaluating the insufficient fulfillment of the profit responsibility: it is not possible to avoid the responsibility by pleading to the fact that one has tried to fulfill their responsibilities. Instead, supervisory responsibility is used, meaning that the supplier can be released from the responsibility only if the contract breach is caused by an obstacle that they cannot impact or bypass and which they could not have known on the moment of contracting. (Hemmo, 2003)

In traditional software development projects the warranty for the system is usually defined so that the finished product matches the functional specification. The fact that the warranty does not mention the actual expectations

of the customer is meant to rule out the Sale of Goods Act §17 responsibility that the product must be suitable for the purpose those kind of products are usually used as well as the customer's special use of which the supplier has been aware. For example, the warranty period may exclude the right for discount. Using the warranty as limitation to liability is not legal in consumer business but usual in contracting between companies: in about 99 % of the cases the customer would be entitled to larger compensations without the warranty clauses. (Takki, 2002)

7.2 Agile warranties

Singe in agile projects comprehensive functional specification is not usually developed, this makes defining warranties trickier in agile projects. To help defining the scope of warranties in agile projects, each delivery should contain summarized user stories of the features included in the delivery. Together these stories will describe the overall functionality of the system and effectively work as a functional specification of the system. (Opelt et al., 2013) This document should, naturally, be developed in cooperation with the customer so that both parties will have the same understanding of the user stories and how they fulfill the product vision. In addition to being useful, low-effort overview of the product, this product description can also be used as a basis of the supplier warranty.

Iterative development also means that features are finished in different times which adds an extra challenge for defining the start of the warranty period. There is a possibility to start the warranty period of each deliverable after the iteration when the feature is accepted, use single starting point for the whole system's warranty or use combination of the two methods. Despite the chosen warranty specification, it should be certain that the supplier accepts the responsibility for the whole project. (Opelt et al., 2013) Takki (2002) notes that if not otherwise specified, no partial acceptance will relieve the supplier from the responsibility that the outcome of the project must correspond to what was originally agreed.

If combination of the two warranty styles is used, at the end of each iteration warranty is tied to the the incremental working deliverable. In addition, there is an overall warranty to the whole product, which starts after the final acceptance. (Arbogast et al., 2010) However, this can lead to complex situations if, for example, sprints 1-8 are accepted (and their warranty period has started), sprint 9 is rejected (thus no warranty) and then sprint 10 is again accepted. This would mean that warranty periods run for parts of the current system but not for the whole system. (Opelt

et al., 2013)

The simplest way of defining warranty in agile projects is, in fact, to define it similarly to traditional project models: a single start date of the warranty is right after the final acceptance of the project. This acceptance is separate from the acceptance of the final iteration and is performed against the product description which is developed during the iterations. (Opelt et al., 2013)

7.3 Warranties in the general contracting terms

In IT2010 general terms the warranty of the software is defined in the EJT special terms. After the supplier has tested the software and delivered it to the customer, the customer has 30 days to perform acceptance tests for it and notify the supplier about all errors found. If the system has an error that prevents the testing, the tests can be suspended until the error is corrected and the testing time is extended by the length of the delay caused by the corrections. (IT2010 EJT, 8.3) Errors that are not severe (i.e. the system can be taken or is taken into production use) will not prevent the acceptance of the system but the supplier does have to correct the errors without delay (IT2010 EJT, 8.4). The supplier will then correct all the reported errors at no cost during the warranty period, which lasts 6 months from the acceptance of the delivery (IT2010 EJT, 11.1). If the system is delivered in phases, the limitations above do not apply to the acceptance of a partial delivery if it has an error that could not have reasonably been noticed before to the acceptance tests of a later partial delivery (IT2010 EJT, 8.5).

JIT 2007 has a similar acceptance phase than IT2010. After the software is delivered to the customer, they have 30 days to perform an acceptance inspection to it and report all found errors in the software. If the software is delivered in phases, the customer will inspect each delivery separately within 7 days of the delivery, and the next phase cannot be started before the previous one is accepted. In an agile project, this would mean unbearable break between each short iteration. The acceptance of interim phases do not discharge the supplier from liability for errors found later in the development or in the final acceptance tests. Minor errors should not prevent the acceptance of the software but the supplier is required to fix those errors at their own cost and without unreasonable delay. Also, if the customer starts the production use of the software, the product is seen as accepted. (JIT 2007, 10)

JIT 2007 states that the ”product and end result of the service must be free of errors”. All the errors have to be fixed by the supplier at his own

expense without a delay, and the supplier has to pay a contractual penalty for the time used for removing the errors after the agreed delivery time. However, this penalty does not apply to situation where the error is a minor one or if the supplier repairs it immediately. If the error prevents the usage of the product, the customer can also withhold the payment of the part of the product that is affected by the error until the supplier fixes the problem. (JIT 2007, 16)

The warranty period of the software in JIT 2007 terms is defined separately in the Applications annex. It is 12 months from the date of the customer's acceptance of the application, and if the application is accepted in phases, the warranty of each part will not expire until 6 months have elapsed from the acceptance of the whole application. (JIT 2007, Applications 6.1) If the product is unusable during this period due to an error, the warranty period is exceeded by that period of time (but not beyond twice the original warranty period). If the supplier does not correct the errors in a reasonable time, the customer can order the repairs from a third party and charge the expenses from the supplier. (JIT 2007, 17)

The customer is entitled to compensation for direct damages (and not for indirect damages) caused by delay or other breach of the contract to the extent by which the damage exceeds the contractual penalty. If the breach is not due to negligence, the compensation should not be more than 7.5 % of the price of the product, but if the damage is due to negligence, the maximum compensation is as high as the total price of the product. Limitations on liability do not apply if the damage is caused deliberately or through gross negligence. (JIT 2007, 21)

Warranty for the system is defined in the Swedish general terms so that if the software has errors caused by the supplier not being skilled enough in their work, the supplier should, at his own expense, correct the errors. These errors should be highlighted by the customer no later than six months after the termination of the project (of three months after the error should have been discovered if it have been discoverable earlier). If the supplier is unable to fix the errors in a reasonable time, customer is entitled to compensation. (General Terms for Agile Projects, 9.1) Unless presence of intent or gross negligence, the amount of compensation is limited to 20 % of the total amount paid for the project, if not otherwise agreed (General Terms for Agile Projects, 10.2).

As said, in the Swedish contract terms, the liability for defects is related to the supplier's workmanship in performing their work. If the supplier's work is professional and skilled, errors found during the iterative development are handled similarly than any other requirements: they continuously change and are prioritized accordingly. In addition to that, the supplier is liable for

errors caused by regression i.e. errors that are caused by the new features not working together with the old ones. The fact that supplier is liable only for damages occurred due to their negligence in performing their work (General Terms for Agile Projects, 10.1) is a difference to traditional contracting where the lack of negligence does not redeem the supplier from the compensations, but in practice it might be hard for the supplier to prove that their work has been professional and skilled.

The Norwegian PS2000 terms take a more traditional approach to warranties: the supplier is responsible for the work done just like in traditional contracting. The terms are similar to the ones in IT2010 and JIT 2007 terms. After the supplier delivers the product, the customer will perform acceptance tests for it, and the project is ended in a common evaluation meeting where both parties can present their possible financial claims towards the other. (Laine et al., 2011)

7.4 Warranties in the case company

The case contract defines warranty period to be 6 months after the go live of the system, and it assures that the work done corresponds to what has been defined. Even though the software is developed and delivered in short iterations, the acceptance of the partial deliveries do not affect the supplier's overall responsibility of the product. The whole product is accepted after the whole system has been delivered and acceptance tests for it have been performed.

During the warranty period, the supplier will fix all the errors found from the system at their own cost. After the warranty period, the supplier is still obligated to fix at their own cost all critical errors found; critical error meaning a error that prevents using the software or parts of it or makes the usability of the software significantly worse. If the supplier does not correct the errors in a reasonable time or if similar errors are continuous despite the supplier's actions, the customer can order the repairs from a third party and charge the expenses from the supplier.

The customer is also entitled to compensation for direct damages (and not for indirect damages) caused by delay or other breach of the contract to the extent by which the damage exceeds the contractual penalty. If the breach is not due to negligence, the compensation should not be more than 7.5 % of the price of the product, but if the damage is due to negligence, the maximum compensation is as high as the total price of the product. Limitations on liability do not apply if the damage is caused deliberately or through gross negligence.

Chapter 8

Results

This chapter wraps up the results from previous chapters by first summarizing the most important findings in a comparison table. This clearly shows how agile-compatible the general terms actually are and what are the weak points when applying these terms to agile projects. The second part of this chapter evaluates how well do the general terms follow the ideology of proactive contracting, where the main point is to form successful business relationships without confrontation. Finally, the last section gives some concrete improvement suggestions that the case company should consider when contracting for new agile projects.

8.1 Summary of the findings

Chapters 3–7 addressed five different aspects important to take into account when contracting for agile projects: development practices, pricing, change management, (early) termination of the project, and warranties and liabilities. For all of those it was researched how the issue is handled in traditional contracting, what is the agile way to deal with it, and how each of the general contracting terms as well as the case contract handle it. These results are summarized in the table 8.1.

Table 8.1: Comparison of the general contracting terms

	Pricing	**Practices**	**Change**	**Termination**	**Warranties**
Traditional	Fixed-price, time& materials, target-cost	Minimal customer collaboration	Strict change management procedures	Difficult, last resort	For the whole product
Agile	Any (preferably not fixed-price)	Customer collaboration neccessary	Rapid respond to change, still steering needed	Easy at any point of the development, may include compensation for supplier	For the whole product, based on summarized user stories
IT2010	Supplier's general price list	Supplier's methods, no need for active customer	Handled by steering group, documented in writing	Only when contract has been breached	6 mo after acceptance tests
JIT 2007	Supplier's general price list	No need for active customer	Must be documented in writing	Only when contract has been breached	12 mo after acceptance tests, or at least 6 mo after final acceptance if accepted in phases
PS2000	Target-cost with upper and lower limits, fixed price for approval phase	Based on Scrum	Small changes: just do it, larger ones agreed in the steering group	After any iteration, compensation for 4-6 % of the total value can be set	Traditional approach
Swedish general terms	Bonus/loss (reduced hourly rate/ compensation for full project)	Development timeboxes, active customer needed	Adjust other requirements (no scope change) or change the plan/ target-price	After any iteration, supplier gets compensation for the next iteration	6 mo after termination, supplier only liable for errors caused by negligence
Case Contract	Bonus (increased hourly rate when success criteria is met for a release)	Agile methods, active customer, responsibility matrix	Handled by steering group, documented in writing	After any iteration, no compensations	6 mo after the go-live

Traditionally software development has been like building a house: first gather the requirements from the customer, then plan the structure and features, build and finally inspect. This kind of development needs minimal customer collaboration and only at the requirements elicitation phase and during the acceptance testing at the end of the project. Changes to the project are hard (but possible) to make, and the contract is strictly not meant to be terminated under any circumstances (unless one of the parties breaches it). Pricing models can vary but fixed-price is probably the customers' favorite: they pay a fixed amount and get a fixed set of features at the agreed date. Defining warranties is trivial: the end product has to contain the features agreed in the contract.

Unfortunately software development is seldom that straightforward, thus agile software development was invented. In agile development responding to change as well as producing maximum amount of value for the customer as early as possible in a form of a working software are the main priorities. This kind of customer-centric development requires an active customer during the whole project, including the implementation phase. This also enables lighter change management procedures, and termination of the project is possible at any stage of the development since all the work already done is in a working, tested software. Almost any kind of pricing model can be used – with the exception that fixed-price contracts are not preferable since they do not react well to rapidly changing requirements. Warranties need to be carefully defined: even though the agile ideology sees code as the primary documentation of the system some kind of other feature documentation is needed in order to base the warranties on something concrete.

The Finnish IT2010 general terms and conditions say that the project is executed using the supplier's working methods but, since no activity from customer (e.g. an active product owner) is not required, using agile methods without additional terms that define the responsibilities of each party would be difficult. Also the pricing of the project would need to be defined in the contract in a way that would drive both parties towards the joint success of the project. Having a steering group that decides the important changes is a solid approach for agile projects too, but the separately defined responsibilities would have to contain the right for the product owner to do small changes e.g. changing priority of the features. Still, probably the biggest issue with IT2010 terms is the termination: the terms only allow termination if one of the parties have breached or will breach the contract.

The IT2010 are clearly meant for traditional software development and not for agile projects. Same goes with the JIT2007 terms used by the public administration: early termination is practically impossible, no customer collaboration is required during the implementation and the pricing uses

only penalties for late deliveries and no incentives for the supplier to finnish early. The companies building software for the public administration might be forced to use the JIT 2007 terms, but it is apparent why the agile software companies working for the private sector do not want to use the IT2010 terms.

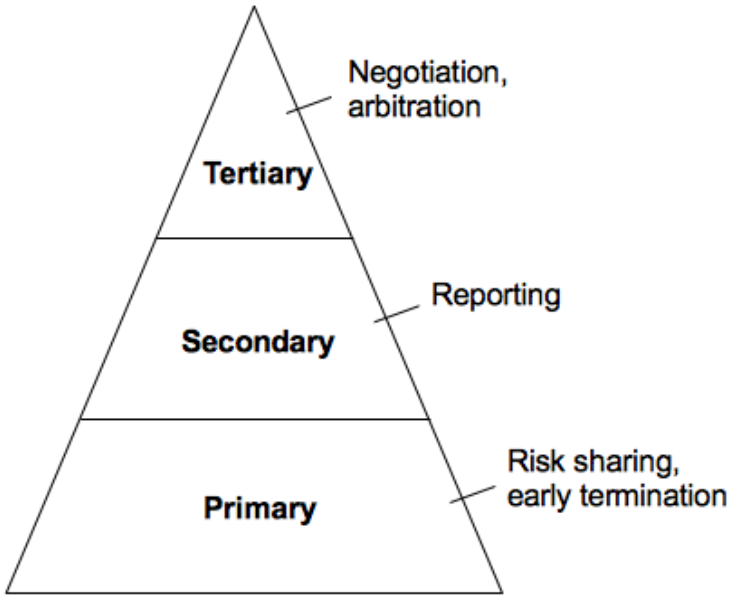
The Norwegian PS2000 terms are clearly more agile than the two terms used in Finland. They are based on the agile Scrum development model and use target-cost pricing with upper and lower limits so that the costs or savings are fairly shared between the parties. For the final, approval phase the terms still use fixed price. Changes to the project are easy to made: small changes can just be done without any additional process and bigger ones are agreed in the steering group. Termination of the project is also easy and can be done after any iteration, but it may contain compensation for 4-6 % of the total value of the project for the supplier.

Clearly the most agile of all the general terms are the Swedish terms developed for agile software development. They present an iterative development model where development is done in time boxes similar to Scrum sprints. Project uses bonus/loss pricing model where the supplier will work under reduced hourly rate if the target is not reached and gets a compensation for the whole project if the project is finished early. When some of the requirements change, the others are adjusted so that the total scope of the project does not change or, if that is not possible, the plan or target-cost is adjusted accordingly. The contract should state who has the authority to do the changes to the project.

In the Swedish general terms the termination of the project is easy and can be done after any iteration. In order to this to be fair for the supplier, the customer will pay compensation for the next iteration for the resources the supplier cannot immediately reallocate. A difference between Swedish agile terms and all the other general contracting terms is that in the Swedish terms the supplier's liability for errors is related to his workmanship in the project. Other general terms use the traditional approach where the liability is related to the produced product instead of the work done for the customer like in the Swedish terms.

The case contract is also made for agile software development thus it requires active customer collaboration and states the responsibilities of each party in a responsibility matrix. The pricing is also bonus/loss in a way that the supplier uses lower hourly rate than normally and after each release the steering group checks if the supplier has reached all the defined success criteria for the release and thus deserves a bonus. Change management is done in the steering group but the product owner has the authority to reprioritize features. Project can be terminated after any iteration and there

Figure 8.1: Proactive Contracting Levels



are no compensations for the supplier.

8.2 How proactive the general terms are?

Proactive contracting, introduced in the chapter 2, is a fine, agile-like ideology aiming towards win-win situations where both parties are satisfied. Still, it is lacking concreteness and concrete examples of how to actually follow the proactive ideology in contracting. The figure 8.1 tries to illustrate how the proactive ideas are implemented in the general contracting terms analyzed in this thesis.

The tertiary level of proactive contracting (consequences) is generally well handled in Finland. Hemmo (2005) describes that also in traditional contracting going to court to settle disputes is an unwanted situation that usually destroys the business relationship. On the other hand, when the stakes are high the parties are afraid to settle to a certain compensation or waive their rights to higher rebate without a legal process. The negotiations can be made easier by using an external arbitrator or specialist, and usage of some kind of negotiation process before a lawsuit can be required in the contract.

Also all the general contracting terms are proactive what comes to the tertiary level. In the Swedish general terms as well as in the IT2010 terms the disputes are settled by arbitration or in the district court if the dispute is only about monetary charges (IT2010 YSE; General Terms for Agile Projects). In the JIT 2007 terms the disputes should primary be handled through negotiation, and, if that fails, in a local court of law or by arbitration (JIT 2007). Also in the case case contract disputes are settled by arbitration.

The secondary level of proactive contracting (effects) is about minimizing harmful consequences when problems do occur. All the general terms as well as the case contract require that the supplier will inform the customer about the status of the project regularly, and both parties have the duty to inform the other party immediately about any matters related to the implementation of the agreement (IT2010 EJT; JIT 2007; General Terms for Agile Projects). This way the problems can be handled immediately and not only at the end of the project. In the Swedish agile terms and in the case contract that both use sprints, this means also reporting immediately if the sprint cannot be fully completed.

So all the general contracting terms are quite proactive what comes to the tertiary and secondary levels. The differences appear on the primary level of the proactive contracting (causes) that aims towards eliminating possible causes for problems before they happen. For example, in the Norwegian PS2000 terms the customer will perform a risk analysis and produce an uncertainty matrix (completed by the supplier). Each of the risks is bared by the party more capable of handling the risk factors and minimizing the damages. (Laine et al., 2011) The case contract has a similar approach: the contract has responsibility matrix that clearly states who is responsible of what. Also the Swedish terms state carefully the responsibilities of each of the parties (General Terms for Agile Projects). On the other hand, the Finnish general contracting terms (IT2010 and JIT 2007) have no similar risk sharing procedures.

Another aspect in the primary level of proactive contracting is the agile easy early termination of a project: possibility to terminate the project easily reduces the trust needed towards the supplier and reduces customer's financial risks. This is not something that would traditionally be possible in contracting – instead termination of the project is extremely difficult and requires a fundamental breach of the contract. Also the Finnish general contracting terms make early termination practically impossible, making them quite non-proactive what comes to the primary level of proactive contracting. On the other hand, both Swedish and Norwegian general contracting terms as well as the case contract are proactive also in this aspect making early termination easy by requiring no cause for it and defining at most small

compensations for the supplier.

8.3 Improvement suggestions

As a result of the research done in this thesis, the following suggestions should be taken into account in the case company when preparing contracts for future agile projects:

- Lighter change management for small changes
- Easy early termination but small penalty to lower the supplier's risk (e.g. the cost of one sprint)
- Extreme idea: warranties and liabilities related to the work done for the customer instead of the produced product

One of the main ideas in agile is the rapid change management. Still, even agile projects cannot afford to not to have any change management procedures, and larger agile projects do need a steering committee to handle the business requirements as well as larger changes to the project. It should always be stated in the contract who has the authority to do changes in the project, but for small changes (that do not affect cost or schedule) this could be someone else than the steering committee, e.g. the product owner. For example, in the Norwegian terms and conditions, this kind of small changes do not need to be handled by the steering group.

In the case company, customers can terminate the project without a cause and without any penalties after any iteration. This is definitely a good practice in agile projects as easy termination of projects is important in the agile ideology. This can also be used when selling agile to customers more accustomed to traditional development methods: they can easily try it out and terminate the project the minute they feel uncomfortable with it. Still, this is a risk for a supplier who ties resources to a project and those resources usually cannot be reallocated immediately after a project is suddenly terminated. Thus, the company should consider using a similar penalty than the Swedish terms use: the supplier gets compensation for the next iteration for the resources that cannot be immediately reallocated.

Most of the general contracting terms as well as the case contract follow the model where warranties and liabilities are related to the produced product. The Swedish general terms and conditions are the exception: they define liabilities in a way that they are related to the work performed for the customer. The supplier is only liable for errors if the work is not done

properly and professionally. Because of the nature of software products, a completely bug free software is an unicorn even when the supplier is skilled and works professionally. This kind of idea of liabilities being related to supplier's workmanship might be hard to sell for the customer but would be really fair for the supplier who performs their work properly.

Chapter 9

Discussion

As can be clearly seen from the results chapter in this thesis, the Finnish general contracting terms (IT2010 and JIT 2007 terms) are definitely old-fashioned and meant primarily for traditional software development projects. Both of the terms fit poorly for the agile ideology, making them difficult or impossible to be used in agile development projects. On the other hand, in other Nordic countries (at least in Sweden and Norway) the usage of special terms and conditions for agile software development is already customary.

Since increasing number of companies in Finland (55 % in 2011) are using agile methods in their software development, this also means that more and more companies are forced to define their own agile contracting terms. This takes time and effort from the company having to prepare the terms (probably with a lawyer), as well as makes the actual contracting process slower. Having a commonly agreed terms and conditions for the whole industry would significantly help the contracting process since both parties would already know the terms. The general terms would also be evaluated to be fair for both of the parties – a factor that the parties now have to carefully evaluate themselves.

In a way it is understandable that the Finnish general contracting terms are old-fashioned: they are both quite old. For a rapidly developing ICT industry, it is too slow and non-agile to have general contracting terms that are update once every 10 years: the predecessor of IT2010 terms were IT2000 terms published in 1999 and the predecessor of JIT 2007 terms were General terms and conditions of government procurement of information technology published in 1998 (VYSE 1998). On the contrary, for example in Norway, the PS2000 general terms and conditions were published in 2000, after which The Norwegian Computer Society has been actively maintaining and further developing the terms.

Unexpectedly, the public administration is waking up to follow the agile

movement faster than the private sector: The Advisory Committee on Information Management in Public Administration (JUHTA) is going to publish new JIT 2014 terms that contain special terms for agile development projects. Those terms are based on iterative development conducted in the customer's premisses. The customer is responsible for participating actively in the development, the constant prioritizing of requirements and the acceptance of each iterative delivery. Minor changes (e.g. prioritization or schedule changes within the overall project schedule) can be done without change management procedures, larger changes should use the mutually agreed procedure. (JIT 2014)

Even though the draft of the JIT 2014 terms are much more agile than the previous JIT 2007 terms, they are not quite as agile as one would hope so. For example, the early termination of the project is still extremely difficult, practically impossible. Likewise the intellectual property rights for the system will still belong to the supplier, making it hard for the customer to switch the supplier in the middle of the project. Also compelling and motivating pricing models are not included in the general terms but the terms continue to rely more on giving penalties for delayed deliveries. (JIT 2014)

The many non-agile aspects in the JIT 2014 agile terms might partly exists because of the strict Act on Public Contracts in Finland. The stiff and formal public sector may also be a reason not to update the general terms constantly. On the other hand, the parties preparing those terms for the private sector, e.g. the Federation of Finnish Technology Industries and the Finnish Information Processing Association, have no excuse. Hopefully, they will soon follow the lead and prepare a proper general contracting terms to be used in agile development projects. Those terms should also be updated frequently – once every 10 years is definitely not often enough for the ICT industry.

Chapter 10

Conclusions

"Customer collaboration over contract negotiation" does not mean that agile software development projects do not need contracts. Contracts are vitally important also in agile projects but the whole view to contracting is different, more proactive. Agile practitioners see that collaboration, rather than confrontation, produces better products and more successful, win-win business relationships. This also adds new challenges to contracting and makes usage of the old general contracting terms developed for waterfall development difficult.

The literature study conducted in this thesis showed that the main issues to take into account in agile contracting are agile development practices, pricing, change management, early termination of the project, and warranties and liabilities. Unlike in traditional development where customer collaboration is minimal, agile development needs a contract that clearly states the activities needed from the customer. The customer's representative should have the authority to do at least small changes to the project, such as prioritizing, and the responsibility to be available for the development team.

In agile projects, the customer may find the product to be ready even if not all of the originally planned features are implemented, thus early termination can be a positive event. The contract should allow that, possibly introducing a small compensation for the supplier when the project is ended prematurely. Also the used pricing model should drive the parties towards the joint success, sharing both the pain and the gain – something that fixed-price pricing does not do very well. The warranties should be carefully defined in a way that takes into account the partial deliveries: a newly added feature may reveal bugs in the previously accepted releases.

These aspects also show the visible problems the Finnish general contracting terms (IT2010 and JIT 2007) have when used in agile development. They are clearly not meant for agile projects, thus both are lacking all the

practices necessary for agile development such as active customer collaboration and iterative development. Since e.g. the product owner role is not defined, all the changes must go through the steering group. Also early termination of the project is practically impossible, and neither of the terms offer any intriguing pricing models that would aim towards the joint success. The new JIT 2014 terms (yet to be published) are a step towards more agile contracting in the public sector but they are still lacking some of the main aspects of agile development such as easy early termination.

On the other hand, the Swedish and Norwegian agile terms are well suitable for agile development. Both are based on iterative development and require an active customer representative who has the authority to do necessary changes to the project (at least prioritization). Termination of the project is easy, containing only reasonable compensations for the supplier, and pricing models share the risks fairly between both of the parties. The Swedish terms define warranties a bit differently than the others: unlike in the other terms researched, they define warranties to be related to the supplier's workmanship instead of the produced product. The Finnish associations developing the general contracting terms have a lot to learn from the Swedish and Norwegian agile terms.

The case company is already doing quite well with the agile contracting: the pricing model (bonus paid when release meets its success criteria) offers reasonable incentive to the supplier, responsibilities of each party are defined in the responsibility matrix and the project can be easily terminated after any iteration. The case company should consider introducing a small compensation when the project is terminated early, e.g. for the next sprint for the resources that cannot be immediately reallocated, like in the Swedish general terms. Also the change management could be a bit lighter for the small changes; now everything but prioritization needs to be handled by the steering group. Additionally, the Swedish way of defining warranties based on the work done for the customer should be considered, although the customers may not be too keen on taking that practice into use.

In the future, when the new JIT 2014 terms are published, they should be more carefully evaluated against the agile principles. If they are still lacking some of the basic agile aspects – like the form they currently are – it would be interesting to know what are the reasons to conduct only almost agile terms for agile development. Meanwhile, it would be interesting to study how the current agile projects made for public administration have overcome the inadequacies the JIT2007 terms have when used with agile projects. Since the public administration usually requires the usage of JIT terms, there should definitely be agile projects done using JIT2007 terms.

It would also be beneficial to conduct a larger study in order to research

how other Finnish agile companies have solved the agile contracting problems and is any of them trying to use the IT2010 general contracting terms. This could also include the viewpoint of the customer: how do they see the general contracting terms and agile contracting. Also a study about the usage of different pricing models in the Finnish agile consulting business would be interesting. For example, all the agile practitioners say they hate fixed-price contracts, but how often they are still used in agile projects?

Chapter 11

Evaluation

The aim for this thesis was to find out the special characteristics in agile contracting compared to traditional contracting, research how well the general contracting terms used in Finland and other Nordic countries fit to agile development, and give concrete improvement suggestions about agile contracting primary for the case company but also generally for the Finnish ICT industry. Generally, the thesis succeed to fill its purpose well. The main aspects about agile development that affect contracting were investigated and compared to traditional contracting in Finland. They were also used to evaluate how well Finnish IT2010 and JIT 2007 terms, Swedish agile contracting terms and Norwegian PS2000 terms fit agile for agile development projects. The case company was given three concrete improvement suggestions based on the literature study and the contracting terms used in Sweden and Norway.

The main contribution of this thesis was the interdisciplinary approach to contracting for agile software development projects. There are plenty of studies about agile software development as well as contracting in general but not so many large-scale studies about agile contracting. There are some papers written about it (especially about the pricing in agile contracts) and plenty of power-point presentations on the topic, but this is probably the largest multidisciplinary study about agile contracting that also contains the evaluation of general contracting terms.

Despite the fact that this a quite comprehensive study about the topic, the literature study conducted did not follow any systematic literature study process thus it may not handle all the aspects completely. For the purpose of this thesis, using only the newest possible research as well as some principal books was found sufficient enough to demonstrate the problems agile development can cause to contracting. Furthermore, this kind of interdisciplinary study would have required several separate systematic literature

studies which was out of the scope of this thesis. Still, there may also be other agile aspects that have an effect to contracting missing from the evaluation since this thesis focused on the most important and effective ones.

The main focus in this thesis was on Finnish agile contracting and from Finland both of the general contracting terms used are fully evaluated. At the same time, this thesis is lacking a complete view for the whole Nordic ICT industry's way of handling agile in their general contracting terms. The two samples found are from Sweden and Norway, and the Norwegian terms had to be evaluated based on second-hand research because the terms are unfortunately not available for academic work. Still, as a result, the thesis shows clearly the problems the Finnish general contracting terms have when applied to agile projects and gives some examples from Sweden and Norway on how to make the contracting terms more agile.

The case study in this thesis covered only one company, which does not show full image of the industry-wide conventions. On the other hand, the general contracting terms used in Finland should represent the common consensus in the field: the terms are agreed together and they should demonstrate what the industry sees as reasonable in contracting. Still, a single contract studied is enough to clearly show that the general contracting terms cannot be (and are not being) used in agile contracting. The general contracting terms are evidently outdated.

Bibliography

Tom Arbogast, Craig Larman, and Bas Vodde. *Practices for scaling lean & agile development: large, multisite, and offshore product development with large-scale scrum*. Pearson Education, 2010.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. The agile manifesto. *The agile alliance*, 200(1), 2001.

Bird&Bird. Contracting for agile software development projects, 2012.

Kirsimarja Blomqvist. The many faces of trust. *Scandinavian journal of management*, 13(3):271–286, 1997.

Kirsimarja Blomqvist, Pia Hurmelinna, and Risto Seppänen. Playing the collaboration game right?balancing trust and contracting. *Technovation*, 25(5):497–504, 2005.

Kirsimarja Blomqvist et al. *Partnering in the dynamic environment: The role of trust in asymmetric technology partnership formation*. Lappeenranta University of Technology, 2002.

Barry Boehm and Richard Turner. *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional, 2003.

Todd H Chiles and John F McMackin. Integrating variable risk preferences, trust, and transaction cost economics. *Academy of Management Review*, 21(1):73–99, 1996.

Margaret H Christ, Karen L Sedatole, and Kristy L Towry. Sticks and carrots: The effect of contract frame on effort in incomplete contracts. *The Accounting Review*, 87(6):1913–1938, 2012.

Michael Coram and Shawn Bohner. The impact of agile methods on software project management. In *Engineering of Computer-Based Systems,*

2005. ECBS'05. 12th IEEE International Conference and Workshops on the, pages 363–370. IEEE, 2005.

Armin Eberlein and JCSP Leite. Agile requirements definition: A view from requirements engineering. In *Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE?02)*, pages 4–8, 2002.

Bruce Eckfeldt, Rex Madden, and John Horowitz. Selling agile: target-cost contracts. In *Agile Conference, 2005. Proceedings*, pages 160–166. IEEE, 2005.

Pekka Forselius. *Onnistunut tietojärjestelmän hankinta*. Talentum, 2013.

Teresa Franklin. Adventures in agile contracting: evolving from time and materials to fixed price, fixed scope contracts. In *Agile, 2008. AGILE'08. Conference*, pages 269–273. IEEE, 2008.

General Terms for Agile Projects. It & telecom companies in sweden.

Tom Gilb. *Competitive engineering: a handbook for systems engineering, requirements engineering, and software engineering using Planguage*. Butterworth-Heinemann, 2005.

J Glover. Profiting through trust. *International Management*, pages 38–40, 1994.

Paul S Grisham and Dewayne E Perry. Customer relationships and extreme programming. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–6. ACM, 2005.

Gregory T Gundlach and Patrick E Murphy. Ethical and legal foundations of relational marketing exchanges. *The Journal of Marketing*, pages 35–46, 1993.

Helena Haapio. Business success and problem prevention through proactive contracting. *Stockholm Institute for Scandinavian Law*, 1999, 2010.

Mika Hemmo. Sopimusoikeus 2. *Talentum, Helsinki*, 2003.

Mika Hemmo. Sopimusoikeus 3. *Talentum, Helsinki*, 2005.

Rashina Hoda, James Noble, and Stuart Marshall. Negotiating contracts for agile projects: A practical perspective. In *Agile Processes in Software Engineering and Extreme Programming*, pages 186–191. Springer, 2009.

IT2010 EJT. Special terms and conditions for deliveries of data systems and customized software.

IT2010 YSE. General terms and conditions.

Frank L Jeffries and Richard Reed. Trust and adaptation in relational contracting. *Academy of Management Review*, 25(4):873–882, 2000.

JIT 2007. Terms and conditions of government it procurement.

JIT 2014. Terms and conditions of government it procurement, draft. `http://www.jhs-suositukset.fi/web/guest/jhs/projects/palautepyynto-jhs166`, fetched 20.4.2014.

Oualid Ktata and Ghislain Lévesque. Agile development: Issues and avenues requiring a substantial enhancement of the business perspective in large projects. In *proceedings of the 2nd Canadian conference on computer science and software engineering*, pages 59–66. ACM, 2009.

Juha Laine, Erika Leinonen, and Mikko Parkkola. Ketterien tietojärjestelmäprojektien sopimuksellinen hallinta. *Edilex*, 17, 2011.

Casper Lassenius. Bespoke it system acquisition: A software engineering perspective -lecture, 2012.

Yossi Lichtenstein. Puzzles in software development contracting. *Commun. ACM*, 47(2):61–65, February 2004. ISSN 0001-0782. doi: 10.1145/966389.966391. URL `http://doi.acm.org/10.1145/966389.966391`.

Joan Luft. Bonus and penalty incentives contract choice by employees. *Journal of Accounting and Economics*, 18(2):181–206, 1994.

Angela Martin, Robert Biddle, and James Noble. The xp customer role in practice: Three studies. In *Agile Development Conference, 2004*, pages 42–54. IEEE, 2004.

Frank Maurer and Theodore D Hellmann. People-centered software development: An overview of agile methodologies. In *Software Engineering*, pages 185–215. Springer, 2013.

Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5): 72–78, 2005.

Soili Nysten-Haarala. Contract law and everyday contracting. *A Proactive Approach. Stockholm Institute for Scandinavian Law, Scandinavian Studies in Law*, 49:263–283, 2006.

Andreas Opelt, Boris Gloger, Wolfgang Pfarl, and Ralf Mittermayr. *Agile Contracts: Creating and Managing Successful Projects with Scrum*. Wiley. com, 2013.

Soile Pohjonen. Proactive contracting: In contracts between businesses. *Ius Gentium*, 12:147–194, 2006a.

Soile Pohjonen. Proactive law in the field of law. *A Proactive Approach. Scandinavian Studies in Law, Stockholm Institute for Scandinavian Law*, 49:53–70, 2006b.

Mary Poppendieck and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley Professional, 2003.

J. Pyysiäinen, Maria Paasivaara, and Casper Lassenius. Coping with social complexity in distributed software development projects. In *Proceedings of the 10th International Product Development Management Conference*, Bryssels, Belgium, 2003. Helsinki University of Technology.

Paul Radford and Robyn Lawrie. The role of function points in software development contracts. In *Australian Conference on Software Measurement*, 1996.

Balasubramaniam Ramesh, Lan Cao, and Richard Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010.

Pilar Rodríguez, Jouni Markkula, Markku Oivo, and Kimmo Turula. Survey on agile and lean usage in finnish software industry. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, pages 139–148, New York, NY, USA, 2012. ACM.

Sale of Goods Act. 27.3.1987/355. `http://www.finlex.fi/fi/laki/ajantasa/1987/19870355`, fetched 15.2.2014.

George J Siedel and Helena Haapio. Using proactive law for competitive advantage. *American Business Law Journal*, 47(4):641–686, 2010.

Peter Stevens. 10 contracts for your next agile software project, April 2009. `http://agilesoftwaredevelopment.com/blog/peterstev/10-agile-contracts`, fetched 15.2.2014.

Pekka Takki. *IT-sopimukset: käytännön käsikirja*. Talentum, 2002.

Vesa Teikari. *Ketterän kehityksen ostajan opas*. 2012.

VersionOne Inc. State of agile development. 2013.

Dave West, Tom Grant, M Gerush, and D D'Silva. Agile development: Mainstream adoption has changed agility. *Forrester Research*, 2010.