Internal Report 2012-2013-05 June 2013

Universiteit Leiden

Opleiding Informatica

Contracting in Agile Software Projects:

Current Challenges and New Possible Solutions

Shi Hao Zijdemans (s1028790)

supervised by J.C. Stettina

and Prof. Dr. B.R. Katzy

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract. Making successful formal agreements has proved to be challenging for Agile Software Development. One of the biggest advantages of Agile project management is the incremental delivery. Because the software is developed in cycles an opportunity for payment in cycles has also emerged. In our exploratory research we investigated current challenges in software contracting and the implications of doing the payments per cycle. We conducted a workshop at an Agile-centered event, and we conducted five semi-structured interviews at different organizations. In this thesis we explain current challenges in software contracting and we discuss the implications of doing the payments per cycle. Finally we explain how the suggested solutions for Agile Contracts could be applied in practice in the near future, and we explain what is currently withholding us from implementing this.

Keywords: Agile project management, Contracts

Table of Contents

1. Introduction

1.1 Problem Outline

Agile project management has become hugely popular in the world of software development (Dyba et al. 2008). Although the Agile approach brings a lot of advantages to both the software supplier and the customer, it can also bring along some difficulties. Some of these difficulties are caused by the fact that Agile software development hasn't been this popular for a very long time yet. In the rise of its popularity, there still hasn't been found a generally applicable solution to a particular set of challenges. Major challenges are often posed in implementing Agile processes in traditional development organizations (Boehm et al. 2005), and there are challenges in finding common ground with the customer in Agile projects (Book et al. 2012). The latter challenge makes for a complicated contract negotiation. Contract negotiation is a crucial part of a software development project, because it sets the tone for the entire project. If the software development company hasn't had any previous projects or collaborations with the customer yet, then a bad contract agreement for their first project together can very well cause it to be their last project together. Challenges with contracts for Agile projects will be the subject of this bachelor thesis.

Contract negotiation has always been a part of the project that requires special attention (Book et al. 2012). In Agile software development it currently is a way harder task to form a mutually beneficial contract than in traditional projects. One of the possible reasons why contracting in Agile software development (Agile contracting) in particular is more troublesome, is because there isn't a widely known contract form that perfectly suits agile processes yet. Currently, Agile projects make use of contract types that have been around before the emergence of SCRUM (the most widely used Agile approach). These contract types are a good fit for waterfall based projects, but some big complications arise when they're applied on Agile projects. A factor that plays a major role in that part, is the fact that the Agile approach embraces change. Negotiating a fixed price or a fixed scope for the project limits the amount of Agility that can be applied in an Agile project. However, good estimations on the price, scope and deadline are the standard for a lot of currently popular contract types. Like this, there are more factors in currently used contract types that make Agile contracting difficult, and therefore also more reasons why there should be a solution to this problem.

There have been many research articles that propose a way to modify existing contract types as to make them work in an Agile project. These suggestions try to fix the symptoms of the problem, not the cause. This problem is currently receiving growing attention. A badly engineered contract can easily result in low software quality, disturbance in the progress of the project, a decrease in Return on Investments, and dissatisfied customers. With these consequences continuously being at stake, you have to expect that Agile project managers around the world are eagerly waiting for a new contract form that has been designed to complement the Agile approach.

1.2 Research Goals

In Agile software development the software system is delivered in cycles (sprints). Agreeing to make the payments in cycles too, seems like a good way to allow more flexibility (Agility) in Agile software projects. We hereby imagine the possibility of buying an amount of initial sprint to give the customer a 'trial' option, and the possibility of buying extra sprints as long as the customer wants more functionality. However this method of making contracts more Agile (Agile Contracting) hasn't been adopted into the Agile community yet. The fact that Agile Contracting hasn't been implemented yet combined with the fact that the current status in contracting for agile projects is problematic, implies that Agile project managers are currently facing some challenges with regard to contracting. In our research we try to find out what these implications are and how they can possibly be reduced. This is an *exploratory* research, where the current challenges and possible solutions are explored by interviewing (Agile) project managers from different companies in The Netherlands. The research question for this bachelor thesis is:

RQ: *Agile projects are developed in cycles which means that developers also can be paid per cycle. What are possible solutions to make contracts more Agile?*

Research work began with studying publications on agile processes, contracting in agile projects, and related work on contracting, which will reviewed in section 2. In section 3, we explain in detail how we have done our research, and the results will be covered in section 4. We leave room for discussion of our research in section 5, and in section 6 we will explain the conclusions that can be drawn from the research.

2. Theoretical Framework

Before doing this research, we had a general (theoretical) understanding of what agile processes look like, and that current contracting in agile projects was problematic. We also stumbled upon a paper from Mathias Book et al. (2012), in which a fair pricing model for Agile Contracting is proposed. The paper stated that it had been tested in one large German software development company, indicating that this way of agile contracting is still very new. But these facts alone are far from sufficient knowledge on our research domain to be able to solve our problem effectively. That's why it's important to first build a theoretical framework, upon which we will build our own field research.

The quality of our research data depends greatly on the quality of our interview questions. In order to form great and effective interview questions we first needed to develop a thorough understanding of current practices in Agile software development and contracting. We gained a lot of knowledge on Agile processes, current most used contract types with Agile projects, Agile contracting, and looked into some Contract Theory. We had to fully understand how Agile processes work and what kind of advantages and disadvantages they have. We found multiple

papers that present an approach to improve the compatibility between current popular contract types and Agile projects. There was very little material on Agile Contracting, which can be explained by the fact that it's still a very new topic. Since we are looking into the financial aspect of software engineering and flexible ways of contracting, we thought that literature research on Contract Theory and Incomplete Contracts could bring an interesting new perspective on Agile Contracting.

2.1 Traditional vs. Agile Software Development

All current software development process models can roughly be divided into two groups: Traditional and Agile (Awad, 2005). The traditional approach is also often called 'plan-driven', 'heavyweight', or 'the Waterfall Model' (Royce, 1970). In this approach to software development the project is divided in a few phases, illustrated in Figure 1:

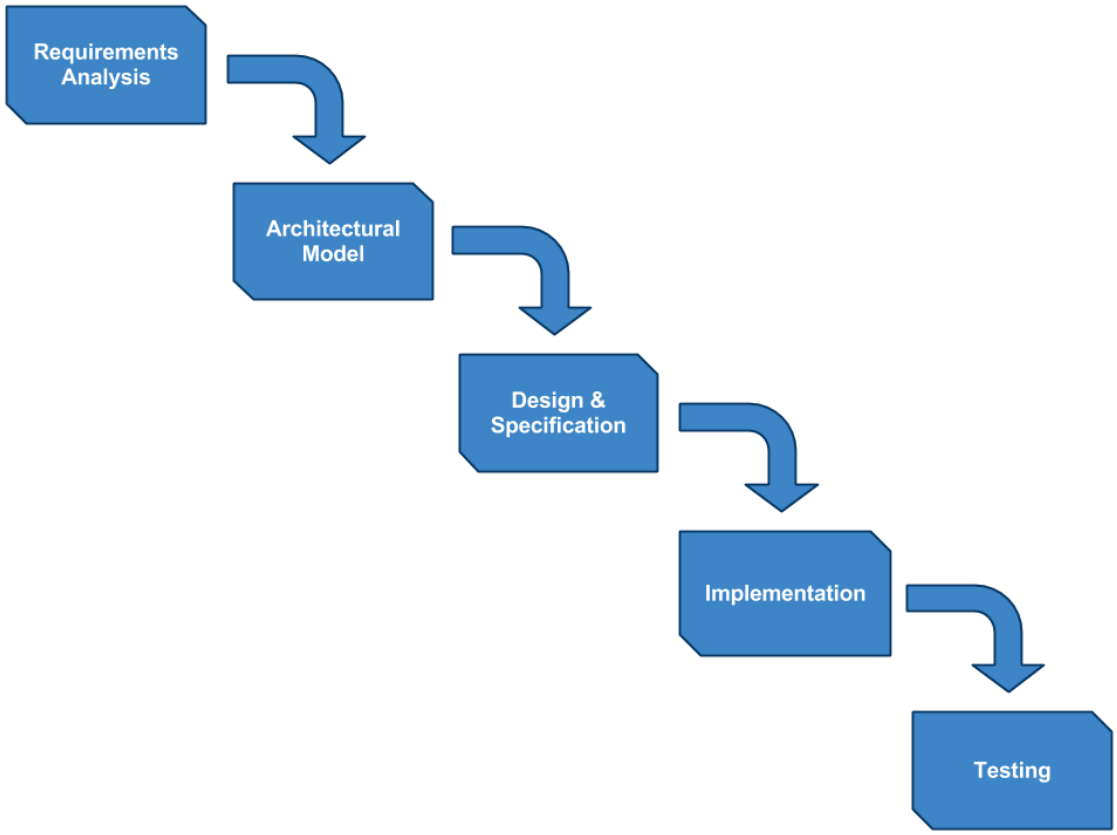

Fig. 1: The main phases in a Waterfall project.

Before implementation begins, the entire project is planned and the system is specified completely. Like all phases, these steps must result in a pile of detailed documentation, which is an important part in traditional software development. Because the different phases are executed by different teams, the documentation is the main source of communication between the different teams. One of the strong suits of the traditional approach is the fact that the initial phase will

result in a detailed plan and design of the entire project. The detailed upfront specification and phased approach is very suitable for projects, for which the requirements can be communicated more easily (e.g. the requirements for a house can be communicated very effectively by a design and a blueprint).

However, the usage of the traditional approach on software projects, often results in software systems that do not satisfy the customers' expectations. One important reason is the complexity of software development, which makes it very challenging to capture all the requirements and expectations from the customer correctly. Another important reason is the fact that in traditional software development, the customer does not see a working system until the very end of the project. Although there is a *validation* and *verification* between the supplier and customer after every phase, the artifacts of a traditional software project (e.g. the design of the system) often do not provide the customer with a representative presentation of the real end result. Also, change requests during the project are difficult to process, because of the detailed upfront specification and the difficulty of communication between the different teams. Therefore, the traditional approach is easier to execute correctly on small software projects of which the requirements and expectations are not very complex.

The Agile approach to software development (also called 'iterative' or 'lightweight'), has quickly gained immense popularity. There are multiple different process models within the methodological school of Agile Software Development, but the base for all these models was created in 2001, when 17 creators and supporters of the lightweight methodology gathered in Snowbird, Utah. Their gathering resulted in a clear set of statements of common values within all Agile process models, called the 'Manifesto for Agile Software Development' (Beck et al. 2001):

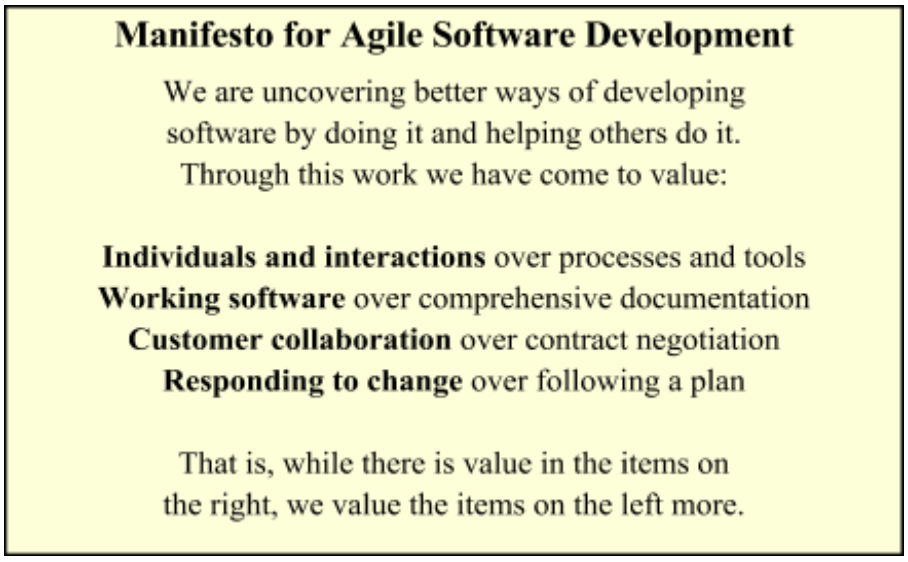

Fig. 2: The Agile Manifesto (Beck et al. 2001)

This gathering also resulted in 12 clearly defined principles behind the Agile Manifesto. Recent study has provided an insight into the Agile community's perception on different Agile principles (Williams, 2012). The results show a ranking of the Agile principles that are essential for a team to be considered Agile. We will shortly highlight the top five principles.

1. ***Continuous integration***: All the new components of the system that are developed during the project, must be immediately integrated with the rest of the system that has been built so far.
2. ***Short iterations (30 days or less)***.
3. ***"Done" criteria:*** the criteria that have to be met in order for the customer to accept the system that is demonstrated must be defined in advance.
4. ***Automated tests run with each build***.
5. ***Automated unit testing***.

There are multiple Agile process models all of which implement Agile principles in their own interpretation (Krebs, 2008):
Extreme Programming or **XP** (developed by Kent Beck, Ward Cunningham, and Ron Jeffries) is one of the most widely used Agile method. XP's most notable contribution to the Agile practices are 'pair programming' and its test-driven development.
Scrum (developed by Jeff Sutherland and Ken Schwaber) is a currently very popular method, which has popularized/introduced Agile principles as 'Daily stand-up meeting', 'Product Backlog' and referring to cycles as 'Sprints' (Schwaber, 1995). Figure 3 shows the process flow in Scrum as presented by Boehm et al (2005).

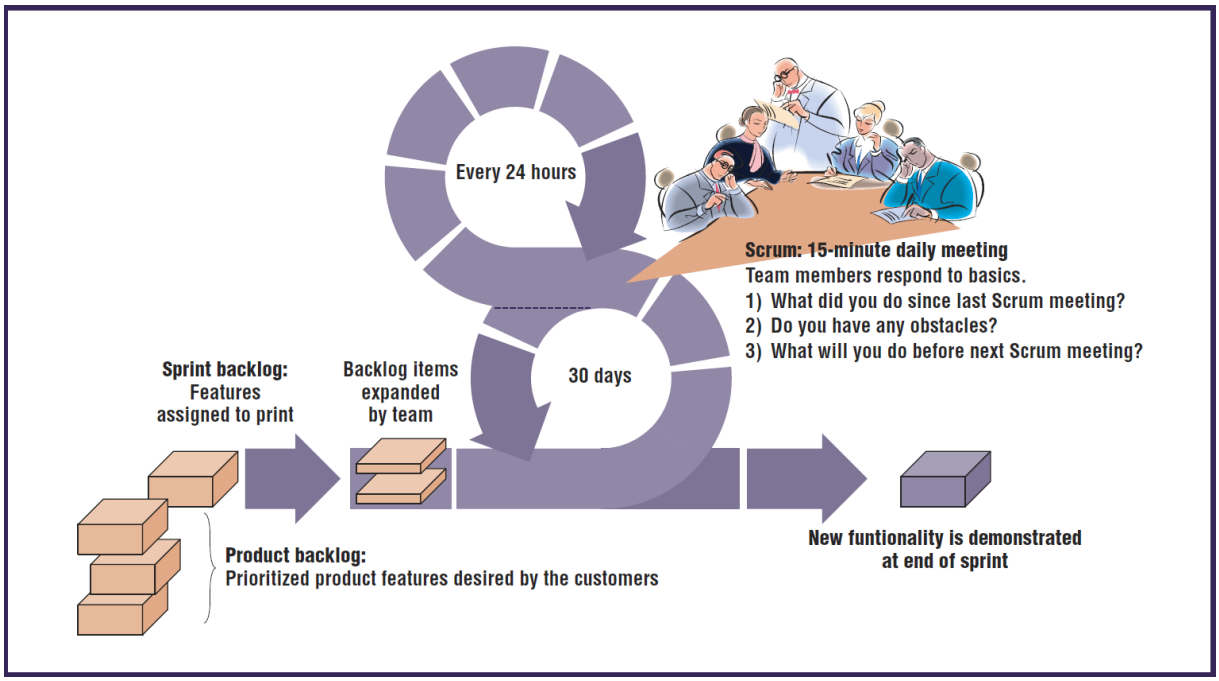

Fig. 3: The process flow in Scrum. (Boehm et al. 2005)

Dynamic Systems Development is a method originating from Rapid Application Development (RAD), which uses prototyping after every iteration to improve customer involvement.
Lean Software Development (developed by Mary- and Tom Poppendieck) originates from lean manufacturing. It does not run the project in iterations, but divides the project according to the individual features of the system to be implemented.
Crystal Clear (developed by Alistair Cockburn) is a process that focuses primarily on the people involved in the process. Crystal Clear focuses on fine-tuning the software development method itself and the product at the beginning and the middle of each iteration.

The main differences between traditional and agile is that agile focuses on *incremental delivery* unlike traditional, where there isn't a working system yet until the very end (Schwaber, 2004). The incremental approach reduces the need to plan the entire project upfront, since the details can be refined during the course of the project. This minimizes risk, because it enables the parties to postpone important decisions to when more knowledge is available. At the end of every iteration a working piece of software should be presented to the stakeholders. This approach welcomes feedback from the stakeholders, which contributes to the final product being more conform to the customer's wishes. Table 1 provides an overview of comparison between the Traditional approach and the Agile approach (Nerur et al. 2007).

	Traditional View of Design	Emergent Metaphor of Design
Design process	Deliberate and formal, linear sequence of steps, separate formulation and implementation, rule-driven	Emergent, iterative and exploratory, knowing and action inseparable, beyond formal rules
Goal	Optimization	Adaptation, flexibility, responsiveness
Problem-solving approach	Selection of best means to accomplish a given end through well-planned, formalized activities	Learning through experimentation and introspection, constantly reframing the problem and its solution
View of the environment	Stable, predictable	Turbulent, difficult to predict
Type of learning	Single-loop/adaptive	Double-loop/generative
Key characteristics	Control and direction Avoids conflict Formalizes innovation Manager is controller Design precedes implementation	Collaboration and communication – integrates weltanschauungs, or worldviews Embraces conflict and dialectics Encourages exploration and creativity and is opportunistic Manager is facilitator Design and implementation are inseparable and evolve iteratively
Rationality	Technical/functional	Substantial
Theoretical and/or philosophical roots	Logical positivism, scientific method	Action learning theory, Dewey's pragmatism, phenomenology

Table 1: Traditional vs Agile (Nerur et al. 2007)

2.1.1 Traditional Planning Approaches

Planning a schedule is essential for projects in all industries. Planning is especially challenging in information technology (IT) projects. Because of the complex nature, and the frequently changing requirements, software projects tend to overrun their cost estimates. Reliable estimates are crucial in traditional projects, since the total price must often be agreed upon at the beginning of these projects (these are called fixed-price contracts, which will be explained in section 2.2.1). Overrunning the cost estimate will result in decreased profits for the software company. In traditional projects there are three artifacts that are mainly used to guide the planning process:

- Work-breakdown structures
- Gantt Charts
- Critical Path Analyses

A *Work-Breakdown Structure* (WBS) is a hierarchical decomposition of all the work items that have to be done in the project. All the activities and tasks of the project are broken down into more detailed items from the top down, so the lower levels are subtasks of the higher levels. When the lowest level of the hierarchy has been defined, an estimation can be made of the required effort and the duration of the project. A problem with this approach is that any work that is not listed in the WBS won't be estimated, and any work performed outside the scope of the WBS will lead to cost overruns. Work-Breakdown Structures are difficult to update, so that is hardly ever done in practice.

Gantt Charts are widely used in traditional projects to show the project schedule. An example is illustrated in Figure 4. These charts display the different tasks in relation to the total time of the project. The main benefit of a Gantt Chart is the ability to give a clear visualization of the duration and order of all the tasks in the project. The problem is that the estimations for tasks later in the project are very likely to be inaccurate.

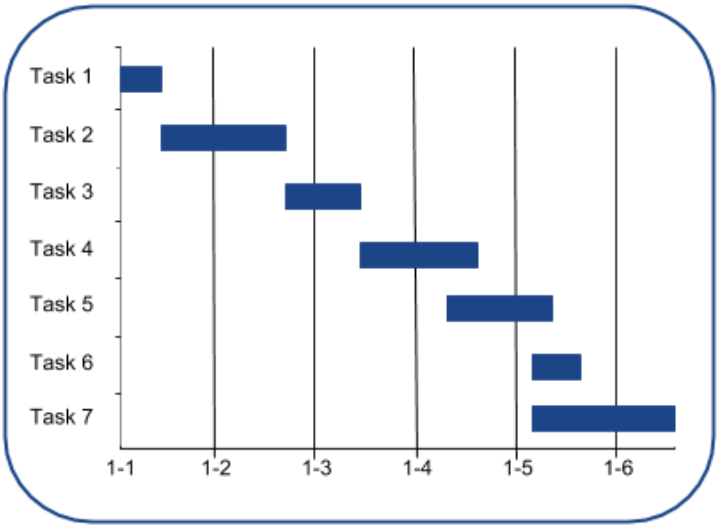

Fig. 4: An example of a Gantt Chart.

Critical Path Analyses are used to identify the shortest way through the project schedule and the dependencies between the activities. Any delay in one of the activities on the so called 'Critical Path' will cause a delay of the project completion date. When the critical path is known, it can help project managers to control the progress of the project.

Besides these widely used artifacts, project teams may use other methods to make estimates. A method that uses a point system to estimate the size of tasks is *Use Case Points*. Use cases are generally larger than user stories and are usually documented as a collection of scenarios. The implementation of Use cases are sometimes spread over multiple iterations. The use case point-estimation process takes multiple steps. The Use Case Points are a product of Unadjusted Use Case Points (UUCP), Technical Complexity Factor (TCF), and Environment Complexity Factor (ECF). These metrics are used to assign a value on different aspects of the use cases, namely: the complexity of the use cases, actors and system boundary in the use case diagram (UUCP), the technical complexity of the overall system (TCF), and the complexity of the environment (ECF). The classifications usually occur in an initial team meeting by the project team. Point systems also have a beneficial psychological impact on the team; whenever a use case turns out to be much more complicated during implementation the points increase, which is not as demoralizing for the team as seeing the extra amount of work hours needed.

The Expert Method is a widely used simple method of asking an experienced person in the organization for his/her estimate on the effort required to complete the tasks. This method is very quick and inexpensive, but has a few pitfalls. The estimation is heavily biased by the personal assessment and ability of the expert. He should adjust the advice based on the competence of the project team, but this has proved to be a very challenging thing to assess.

2.1.2 Agile Estimating and Planning

The most important management and planning artifact in an Agile project is the *Product Backlog (Krebs, 2009, Schwaber, 2004)*. This is basically a To-Do list, containing all the requirements for the software system. In Agile these requirements are often captured in the form of *User stories.* These are represented in the following standard form: *"*<as a> I want <the feature> so that I can <business value>". Being able to see the business value of each requirement helps the developers to implement the system as expected by the customer.

The Product Backlog replaces the Gantt charts originating from the traditional planning tools. A big advantage of the Product Backlog is its simplicity: it's basically a list of all the user stories ranked on priority (user stories that deliver the most business value have the highest priority). This enables stakeholders to update the Backlog easily and timely. Between iterations there is an opportunity to add, delete and re-prioritize user stories in the Product Backlog. Gantt charts are a lot harder to adjust, and when taking into account the high frequency of change in Agile projects, it takes up too much effort to keep it up to date. The fact that a Gantt chart depicts a well thought out plan, while it is not updated consistently, will be deceiving when shown to the customer.

Prior to each iteration, the development team makes a selection of the highest priority user stories from the Product Backlog, that they will implement in the associated iteration. These selected user stories are further specified in a new artifact called a *Sprint Backlog* (originated from Scrum), which contains a lot more technical detail. In determining how many user stories they can implement in the next iteration, the team must make an estimate on the effort required to complete these user stories. A big advantage in estimation in Agile projects compared to traditional projects, is the fact that the estimates are made by the development team itself instead of the project manager. Accurate estimations will lead to satisfying demonstrations for the stakeholders at the end of each iterations, and increased trust in the development team. This is a benefit in every contract type, because the key ingredient for a successful contract is *mutual trust*. There are a few methods specifically for Agile projects that are used to help make accurate estimations. The Agile spirit (lightweight, simple and quick) can be recognized in these methods.

One way to improve the accuracy of the estimates after each iteration is by weighing the planned effort for each requirement against the actual effort that was needed to implement the requirement. By reflecting on the progress, the team can learn from its mistakes and successes. Based on the experience gained after each iteration, the estimates can be made more accurately. This is a very effective and natural way to improve the estimates, and can be used on all the following estimation methods.

Story Points are used to give user stories a value that indicates the size of the user story. With this point system, the development team can avoid using person-days or monetary values to estimate the size of user stories. Story points focus on the size of requirements, not the duration. Different point systems can be made in each development team as long as they are consistent and simple. For example, a user story with two story points should be double the size of a user story with one story point. One of the big benefits the story point system brings along is the ability to make simple predictions for the remaining duration of the project, based on completed iterations. This technique is called the measurement of *Velocity*, and can be seen as the speed of the development team. For example, when a team delivers 40 points for the first iteration, and the entire project is estimated at 400 points, the manager can roughly estimate the remaining duration of the project at 9 or 10 weeks.

An effective and quick way to elicit the personal estimates on the effort or size of user stories from every single team member is called *Planning Poker* (Cohn, 2005). This is a game-like process in which every team member has a hand of cards showing the Fibonacci sequence up to 100 (to account for the uncertainty in estimating large user stories). First a user story is announced by the Product Owner, upon which a time-boxed discussion ensues where the team member can ask the Product Owner questions that will help them make the estimates. The Scrum Master then tells the team members to each show their card that represents their estimate on the user story. All the team members have to do this at the same time as to prevent the team member

to influence each other. If the estimates don't differ much, the Scrum Master assigns the most frequent estimate to the user story. When there is a wide disagreement, the round is repeated for the same user story, and the estimates can be adjusted according to the new discussion. A disadvantage of these discussions is that it leaves room for dominant and well-spoken team members to interfere.

2.2 Contracting in Software Development Projects

Software projects are very complex, and the requirements are as hard to elicit and understand correctly as in any other field of projects. In the following subsections, we will discuss two of the most frequently used contract types, which can be seen as opposites of each other: Fixed price contracts and Time & Materials contracts. Then we will briefly cover 'Incomplete Contracting', which is a theory stemming from microeconomics. Finally, we will give a detailed explanation of Agile Contracting, and we will cover the currently available literature on Agile Contracting.

2.2.1 Fixed Price Contracts

Fixed Price contracts in software development define the price, scope, features, planning and deadline of the project (Hoda et al. 2009). The supplier is paid when the software system is delivered to the customer. All the contract elements can only be properly established after a detailed specification of the entire system has been made. Hence, the Fixed Price contract is a good complement to traditional software projects.

In section 2.1.1 we explained why the traditional approach might be troublesome for software development. The way the price for Fixed Price contracts is established often cause additional implications on the software projects. Whenever organizations want to have a software system made (with a Fixed Price contract), they post a Request For Proposal (RFP) to elicit bids from potential software suppliers. Most responses to RFPs consist of a bid, qualifications, prior similar projects, development methods to be employed, and a plan. The big problem that often occurs in this early stage is the fact that the software suppliers are to make an estimate on the cost and duration of the project based on the specification in the RFP, even though these specifications often are vague (Book et al. 2012). The software suppliers have to make a guess on the price, while also aiming to beat the price of competitors. This usually results in a price that is so low, that the software supplier that has won the contract subsequently experiences declining profits due to his (too) low bid. This is why the risk is completely on the supplier's side in Fixed Price contracts.

The Fixed Price contract does not work well with Agile projects in theory and in practice. The fixed price is based on an estimation of the effort and cost of the project. Just like in traditional software projects, there is a risk of inaccurate estimations. When the actual required effort turns out to be much bigger, it is often the quality of the software system that has to suffer. When Fixed Price contracts are used in an Agile project the chance of the actual effort being higher than the estimated effort is significantly higher, because the Agile approach encourages the

software supplier to embrace change. The ability to keep adding bells and whistles at no extra cost is tempting for the customer, but they often fail to realize that this leads to increased work pressure for the software developers, which will come at cost of the software quality.

Compared to our research, similarity can be found in a research project on the challenges that software vendors in India and New Zealand encounter in contracting for Agile projects (Hoda et al. 2009). Grounded Theory has been used to analyze the data gathered from interviews at several software companies. This research has shown that currently, software suppliers using the Agile framework have trouble offering any other contract type than Fixed Price contracts to the customer, because the customers are used to them. Knowing the exact date when the software will be released and the exact cost of the project is often perceived as valuable by the customers. This research can be viewed as further evidence supporting the argument that fixed bid contracts and Agile principles are not directly aligned, and that subsequently contract negotiation is a real issue for Agile practitioners.

2.2.2 Time & Materials Contracts

Another contract type that is used in Agile projects is the Time & Materials contract. In this contract type the software supplier is paid per hour (Stevens 2009). At specified time intervals (e.g. monthly) the software supplier sends the customer a bill according to the amount of hours that the supplier has worked on the project. The scope of the project is not specified upfront, and is not mentioned in the contract at all. Time & Materials contracts can be ended whenever the customer wants (e.g. when enough the system that is delivered so far is of enough business value to the customer, or when the customer's budget limit has been reached).

In Time & Materials contracts all the risk is carried by the customer. The supplier has no incentive to work efficiently, because the supplier is paid for every hour that it claims to have spent on working on the software system. In fact, it is more profitable for the supplier to work inefficiently (Thorup et al. 2009), because this prolongs the contract and hence the stream of revenue for the software company. To prevent this, the customer can impose control mechanisms that monitor the supplier´s efficiency. However, this can hinder the collaboration between the parties. Another factor that adds risk for the customer is the uncertainty that can be experienced due to the absence of a price estimate of the project.

The flexibility that is supported by Time & Materials contracts makes this contract type more suitable for Agile projects in theory. Unlike Fixed Price contracts, Time & Materials contracts do not conflict with the Agile approach to software development. When there is mutual trust between customer and supplier, and the supplier tries to work efficiently, the Time and Materials contract can be a really easy contracting solution for Agile projects.

2.2.3 Incomplete Contracts

Since we are looking into the financial aspect of software engineering and flexible ways of contracting, we thought that literature research on Contract Theory and Incomplete Contracts could perhaps bring an interesting new perspective on Agile Contracting.

A significantly important subject within Contract Theory is the amount of 'contract completeness'. Based on contract completeness, contracts can be said to be 'complete' or 'incomplete'. A contract in which, upon inspection, there is a lot of room for *ex-post* (after the contract has been signed) negotiation on certain contingencies is called an 'incomplete contract'. This can cause unpleasant situations, if one of the parties decides to take advantage of impreciseness in the contract (i.e. opportunism). Whenever there is no room for *ex-post* negotiation about the content of the contract, it can be called a 'complete contract'.

In literature the idea of contractual incompleteness is described by the following informal story (Hart et al, 1999): "Imagine a buyer, B, who requires a good (or service) from a seller, S. Suppose that the exact nature of the good is uncertain; more precisely, it depends on a state of nature which is yet to be realized. In an ideal world, the parties would write a contingent contract specifying exactly which good is to be delivered in each state. However, if the number of states is very large, such a contract would be prohibitively expensive. So instead the parties will write an incomplete contract. Then, when the state of nature is realized, they will renegotiate the contract, since at this state they know what kind of good should be traded."

In his paper, Saussier (2000) mentions two possible reasons for why incomplete contracts are being signed at all. Reasons that have been proposed usually refer to bounded rationality of economic agents and the verifiability of variables that are relevant to contract fulfillment. But since it's hard to formalize bounded rationality, the answer is searched in the non-verifiability of relevant variables.

The parties involved in contracting face a tradeoff between the costs of engineering a more complete contract and the costs resulting from an incomplete contract. Results have shown that the relative magnitudes of these costs are reflected by the degree of contractual incompleteness chosen in practice (Crocker et al. 1993). The cost of identifying contingencies relevant to the contract will increase rapidly in complex or uncertain environments. High environmental complexity and uncertainty will increase the cost of drafting complete contracts, and thus the parties are tended towards more contractual incompleteness. Whenever there is a record of past opportunistic behavior or the potential for hold-up, there will be an increased likelihood of *ex post* redistributive efforts with the associated bargaining costs, which results in the parties tending towards more complete contracts.

Tirole (1999) provides a summary for the literature on incomplete contracting that had been available before 1999. He states that there is no clear definition of incomplete contracting in

literature, and that presented incomplete contract models base the term ‘transaction costs’ on one or several of the following elements:

- *Unforeseen contingencies:* “The parties can not define the possible future contingencies *ex ante*. This forces them to set up a contract that abstracts those contingencies, or not to set up the contract at all.”
- *Cost of writing contracts:* “Even if all the contingencies can be foreseen, it might be too costly to document them all in the contract, because of the high quantity.”
- *Cost of enforcing contracts:* “Courts must be able to fully understand the terms of the contract and the stated contingencies and actions in order to enforce the contract.

2.2.4 Agile Contracts

Agile software development projects deliver the software in cycles. One of the reasons for why it’s called ‘Agile’ is because after each cycle, there is an opportunity for the stakeholders (especially the customer) to make changes; changes in the design, changes of the order in which the software increments are released, changes in the software requirements etc. The way Agile is designed should allow changes in almost every aspect of the project. BUT in practice there is a major constraint that limits the total amount of project elements that can be changed in between cycles: the contract. This hinders the ability to implement the concept of the Agile approach to its fullest extent. That’s why the world of software development calls for a contract type that enables stakeholders in Agile projects to reap all the benefits that the Agile approach can bring: a real Agile Contract.

This Agile contract type should ensure a fair risk distribution between the customer and the supplier, optimally in a way that both parties share the same risk and the same goal (Book et al. 2012). To be more specific, it should allow the customer to make desired changes in the project scope, without negatively affecting the profitability of the software suppliers. Ideally, the software supplier should be paid after each cycle. The customers should be given the option to end the project early, when they believe that sufficient business value has been delivered and that the remaining features won’t be worth the cost. In that case, the customer should pay a fine to the suppliers, to compensate a bit for the lost profit. This option is not completely unused in the Agile software development world right now. The customers should also be given the option to prolong the project duration by buying extra iterations (sprints), when they want additional features that can’t be implemented within the original estimated duration.
Before our research, our perception of Agile Contracting is:
Contracts in Agile projects, in which the supplier is paid per cycle, while allowing changes to be made in the project scope and ensuring a fair risk distribution between the customer and the supplier.

In the following subsections I will review two papers that present a form of Agile Contracting available in current literature.

2.2.4.1 The 'adVANTAGE' contracting model

The best contract model that we found was by Matthias Book et al. (2012), which is called "adVANTAGE: A Fair Pricing Model for Agile Software Development Contracting". In this contracting model, the software supplier is paid for their effort after each cycle. They claim that the key commercial principles of their model are fair risk distribution and mutual efficiency incentives. In their paper, it is explained that their contracting model combines elements of fixed-price and T&M contracting models: the model strives to provide some idea of the overall projects scope (in terms of requirements, time and budget) to the users and developers, as they would have in a fixed-price project, however without exposing the software supplier to the risk of being committed to that exact effort. Instead this contracting model ensures that the customer pays exactly for what is delivered (like in T&M contracts), however without exposing him to the main risk of T&M contracts: the absence of the suppliers' incentive to be efficient (i.e. the customer bears all the risk).

Because the contracting model that Matthias Book et al. have presented is such a good practical representation of our idea of Agile Contracting, we will go over the model in more detail. The first step (called 'Step 0') is done once in the beginning. In this step, the initial requirements are collected from the customer (in the form of user-stories) and the budget is estimated. The total effort estimates for all the user stories are not collected to calculate a fixed price tag for the project, like in traditional contracting models, but they are used in all iterations as an orientation point for the billing.

The next three steps, that are explained, are iterated steps: In *Step 1* the customer can prioritize, eliminate or add user stories. While doing this, the customer has to take into account the supplier's price estimates and his own budget ceiling. This aspect supports the principle of mutual efficiency incentives, because before each sprint the customer has to balance the importance of the user stories against his available budget and his desired timeframe. This contrasts with fixed-price contracts, in which the customer is often tempted to keep adding bells and whistles at no extra cost. After the list of user stories (Product Backlog) has been updated, the customer and the supplier can discuss which user stories should be implemented in the next sprint. *Step 2* of the model concerns the sprint implementation. At the beginning of each sprint, the user stories that are to be implemented in the sprint are refined into more detailed technical specifications (in close collaboration with the client). In using this contracting model, changing Sprint Backlog items not only conflicts with the rules of Agile, but it will also sabotage the subsequent inspection and billing step.

In *Step 3* the sprint is inspected and billed. This step is what truly makes this contracting model Agile. First each user story is inspected for completion and acceptance by the client, and the estimated and actual effort of all accepted user stories are tallied. The billing depends on whether the user stories were completed and accepted, and the difference between the actual and the estimated effort. The price for the efforts is calculated by multiplying the effort (in person-hours

or person-days) by the suppliers usual hourly/daily rate. The bill consists of the price of cross-cutting tasks (lumps sum for requirements elaboration, the scrum master's work and other efforts that can't be attributed to individual user stories), the estimated effort and the actual effort. Figure 5 (Book et al. 2012) shows the exemplary bill in case of underspending.

Prio.	User Story	Completed & Accepted	Estimated Effort (PD)	Actual Effort
1	User Story A	yes	18	18
2	User Story B	yes	7	7
3	User Story C	yes	42	38
4	User Story D	yes	3	5
5	User Story E	yes	10	8
	Total Effort		**80**	**76**

Estimated price (80 x 1,000 EUR)	80,000 EUR
Underspend (4 x 1,000 EUR)	–4,000 EUR
Cross-cutting tasks (lump sum)	22,000 EUR
Sprint Bill	**98,000 EUR**

Fig. 5: Example of bill in case of underspending (Book et al. 2012)

Whenever the software supplier's actual total effort for the sprint has been lower than the estimated total effort (i.e. underspending), then the customer does not have to pay those wrongly estimated hours/days. So for example: if the daily rate is 1.000 EUR and the actual needed person-days was 4 less than estimated, then the customer gets a 4.000 EUR reduction of charges. On the other hand, whenever the actual total effort has been higher than estimated (i.e. overspending), the supplier pays for the extra effort at a reduced rate (in the example in the paper, a 40% reduction is given). This way of penalizing the supplier for bad estimates, also contributes to the principle of mutual efficiency incentives. Contrary to the T&M contract model, the supplier has to stay efficient to prevent doing work at a considerably reduced rate. This also emphasizes the importance of making accurate estimations.

Prio.	User Story	Completed & Accepted	Estimated Effort (PD)	Actual Effort
1	User Story A	yes	18	18
2	User Story B	yes	7	14
3	User Story C	yes	42	40
4	User Story D	yes	3	5
5	User Story E	yes	10	8
	Total Effort		**80**	**85**

	Estimated price (80 x 1,000 EUR)	80,000 EUR
	Overspend (5 x 600 EUR)	3,000 EUR
	Cross-cutting tasks (lump sum)	22,000 EUR
	Sprint Bill	**105,000 EUR**

Fig. 6: Example of bill in case of overspending (Book et al. 2012)

User stories that have not been completed successfully, can either be transferred to the next sprint, or completely cancelled if the customer chooses so. The estimated and actual effort of the transferred user story will be taken away from the bill and placed in the next sprint's bill, where it will be accumulated with the additional effort made in that sprint. The efforts made on cancelled user stories will still be paid for (following the above rules). After each sprint the customer can terminate the project, when he feels that enough functionality has been acquired or the budget ceiling has been reached. The paper stated that this exit-strategy is risk-free for the customer, which implies that this model does not prescribe a penalty fee to the customer for terminating the project early.

2.2.4.2 Collaborative Agile Contracts

Thorup and Jensen (2009) report on their experiences with their contracting model called 'Collaborative Agile Contracts', which has been tested in two commercial projects. The aim of their contracting model is to distribute risk and benefit fairly between the customer and the supplier. The biggest distinctive feature of this contracting model is that the payment is delayed until a certain criterion is fulfilled. In most contract types this criterion is a calendar data, as is also the case in 'adVANTAGE' (Book et al. 2012). However in Collaborative Agile Contracts, the criterion for payment is to reach a predefined milestone where the customer is getting value from the increment of the system. Such milestones are each associated with the delivery of one area of functionality. Generally both parties would like to reach these milestones as quickly as possible. The supplier's efficiency will be rewarded with a quicker payout. The customer will think more carefully when deciding what features he/she wants to have implemented, because all the increments will be paid for separately. Just like in adVANTAGE (Book et al. 2012), this freedom enables the customer to discard some initial requirements. These factors contribute to the *mutual efficiency incentives.*

In a Collaborative Agile Contract the following elements are defined:

- A short and loose description of the scope, that can be seen as a kind of vision statement
- An hourly price that is 10-50% below the hourly price that would be used in a pure Time & Materials contract.
- A list of milestones, each of which will lead to payment of a fixed amount (although from the paper it is uncertain whether the authors mean that the same fixed price is used for all the milestones, or that each milestone will have its own pre-defined (fixed) price). A milestone is considered 'completed', when the associated increment of the system is deployed at the customer's site.
- A development process following the Agile approach.
- A suggested timeframe for the entire project and the individual milestone, that will serve as a guideline. However, there is no fixed deadline in this contract.

Note that this does not include a detailed requirements specification, and there are no fines at all.

The payment structure in Collaborative Agile Contracts is what truly defines this contracting model. The *total price* is divided in two elements, namely: the *completion price* (which is divided among the milestones relative to their size) and the *price per hour*. In a fixed-price contract the total price would be the only price, and in a T&M contract the price per hour would be the only price. In Collaborative Agile Contracts, the combination of these two price elements comprise two parameters that can be used to optimize the contract model according to the specific project.

An important factor in finding the right balance between the two elements is the emphasis on fast completion. In a project where the software release time (time to market) is very important to the customer, it is smart to opt for a high completion price. This will encourage the supplier to work as fast and efficiently as possible, because this will contribute to a higher profit. The Agile approach and the hourly pay should prevent the supplier from working inefficiently. In projects where the customer wants to continue development until all desired features are fully completed or in projects where there is a lot of uncertainty in the initial estimation on the project, it's more profitable to opt for a high hourly price. This gives the customer more flexibility, because normally (in fixed price contracts) additional feature requests from the customer are not very welcomed by the supplier since they require additional effort without extra pay. A high hourly price makes the supplier more open towards change, which supports the correct execution of Agile processes in the project.

3. Research Methodology

In this research we aim to gain knowledge on the implications for customers and the financial structure of Agile software projects, caused by contract models. Therefore, we must know the current practices and challenges in Agile software contracting. Hence, we have to get in contact with people, that have experience in the field of contracting for Agile software development projects. With these research objectives, our research will be of exploratory nature (Yin, 2008). By doing exploratory research we aim to collect perceptions and opinions on current contracting challenges in Agile software projects, and how contracting can be improved in Agile projects.

3.1 Research Design

We have employed two methods within our case study research: we conducted a workshop and we conducted semi-structured interviews. During our literature research we constructed a list of questions (the interview guide) that we would use to guide our interviews. When we received the opportunity to conduct a workshop in an Agile-related event, we used this opportunity to ask some important questions from the interview guide before conducting our first interview.

3.1.1 Data Collection

The workshop was a good way to elicit feedback from multiple people in a very short time. Since it was conducted before any of the interviews, it proved to be a an effective way to get a preliminary pool of information about the current situation of contracting Agile software projects in practice. The workshop was held at the event "AgiLEREN, Certificeren en Inspireren" hosted by the Agile Consortium at April 25th, 2013. The underlying theme of the event was connecting researchers of Agile with practitioners of Agile (many of which are managers in Agile software development).

Since the author's knowledge about the collaboration between the customer and the supplier in Agile software projects in practice was very limited, the goal of the workshop was to get a general idea of the aspects related to contracting in practice. The information gained from this workshop could give us some practical insight into the current contracting situation, which could help in adding better questions to the interview guide. Since open discussions can be distorted by dominant personalities, we aimed to structure the workshop in such a way that the perceptions and opinions from as many people as possible are collected. Hence we planned to present some questions on posters, on which we collected the participants' thoughts by using post-it notes. These questions formed the guideline during the workshop. The posters were based on three of our wider scoped questions from the interview guide, that were suitable to ask a more diverse audience. The layout of the posters was as such:

1. An Ishikawa cause-and-effect (fishbone) diagram was depicted, where we collected the participants' current challenges in contracting for software development projects.
2. Asked for suggestions on possible ways to improve contracting in Agile projects.

3. Asked for suggestions on important aspects to consider when contracting (Agile) projects.

A total of nine people attended the workshop, of which six people actively participated in the discussion and contributed suggestions to the posters. Among these people were Product managers in Agile Software Development, an Agile Coach who had also given presentations about Agile Contracting, ScrumMasters and a teacher at a university.

We started the workshop by explaining the purpose of the research and by providing a short background to the subject. Then the questions were introduced and every participant was asked to write up their suggestions on post-its and put them on the posters. After the participants had finished contributing their suggestions, they were asked to elaborate on their contribution.

We conducted five semi-structured face-to-face interviews, all in different organizations. The interviews took place between April and June 2013. We interviewed participants from different from a variety of roles, as to ensure that we collected different perspectives on Agile contracting. The interviews had an approximate duration of 1 hour, and they took place at the participants' workplace. Before each interview, we asked the participant for permission to record the interview. We mentioned that all data from the interview will be anonymized in the thesis and in possible future publications. The interviews were voice recorded on a mobile phone, so we would not have to keep taking notes, and we could concentrate on the interview.

The structured part of the semi-structured interviews was prepared by listing open-ended questions in an interview guide. We had to formulate questions that would help us elicit the information that we were looking for, and that would provide us with a guideline during the interviews. This interview guide was initially formed during the literature research, and during the research additional questions were continuously added to the interview guide. Especially the first interviews gave us better insight into the interview guide. While it is important to remain disciplined and focused in data collection we must be reminded of the exploratory nature of our research. Therefore we should be exploratory during the interviews as well. Given the variety of backgrounds of the participants, we altered the questions a bit according to the situation. For example, with one of the participants who is a jurist in the Research & Development area, we skipped the more software development-oriented questions, after he answered that his knowledge about this area was limited. The exact interview guide can be found in Appendix A.

3.1.2 Data analysis

To analyze the data from the interviews, we applied some elements from Grounded Theory, which is a systematic qualitative research method. Grounded Theory, developed by Glaser and Strauss (Glaser et al. 1967), provides us with a method that allows us to rigorously analyze our data from our interviews and to systematically generate theory. It is especially suitable for areas

of research that have not been studied in great detail before, and it allows researchers to study social interactions and human behavior. This is very much applicable to our research, since research literature on challenges caused by payments per cycle in Agile software projects is scarce. Also, Agile methods focus on people and communication.

Originally, in Grounded Theory, the researcher gathers data and then systematically derives a substantive theory directly from that data, instead of first developing a theory and then systematically seeking evidence to verify it (Hoda et al. 2009, Adolph et al. 2008). However, instead of starting with a general area of interest (i.e. 'Agile Contracting'), our research started with a Research Question, that narrowed our research scope to "challenges related to payments per cycle in Agile contracts". By applying some elements from Grounded Theory, we aimed to build a substantive theory that will contribute to answering the Research Question.

All interview have been transcribed line by line. After the interviews had been transcribed, we analyzed the data and explored the meaning in the data. This was done using a technique called 'open coding', which is a technique used to conceptualize the data. In the coding process, the transcripts are analyzed line by line and inspected for key points from each interview transcript. Each key point is then assigned a 'code' (i.e. a descriptive label), which formed our first level of abstraction in analyzing the transcripts.

Interview quotation: *"On the one hand, you want to give certainty about that, [for example by saying] 'we will build everything for cheap'. Well then we get the assignment like that, which is my job; getting assignments. On the other hand, you do of course want to get assignments where you can reach the financial performance that you agreed to. Well yes, that's interesting."* — P3

Code: "Challenge: Comforting the customer"

The codes that have been identified from each interview were constantly compared against codes from the same interview, and codes from other interviews and observations. This is Grounded Theory's *constant comparison method* (Glaser et al. 1967). In this example other similar codes were "Customer wants upfront specification (P3)", "Customer is not-knowing and cautious (P2)", and "Comforting the customer with Fixed Price (P2)". By following the constant comparison method we grouped codes together to create a higher level of abstraction, called 'categories' in Grounded Theory.

Category: "Desire for certainty"

Next, we assembled all the categories and ranked them according to the amount of codes that they contained.

Other elements from Grounded Theory that we applied are memoing and sorting. *Memoing* is the ongoing process of writing up theoretical memos during the Grounded Theory process (Hoda et al. 2010). We applied this technique during the open coding process, and it helped us gain ideas about the codes and concepts, and the relationship between them. This resulted in many separated and unstructured memos. To get clarity and meaning from all the memos, we began to conceptually sort the memos. This process is called *Sorting*, and it was done as one of the last stages of our data analysis.

The data from the workshop will be analyzed using a similar approach. Although the discussions from the workshop were not voice recorded, the data from the post-it notes were written out and expanded with memos immediately after the workshop (this can be found in Appendix B). The data was grouped according to the questions on the posters, after which the data was categorized.

3.2 Research Process

The process of the entire research project is depicted by the model shown in Figure 7.

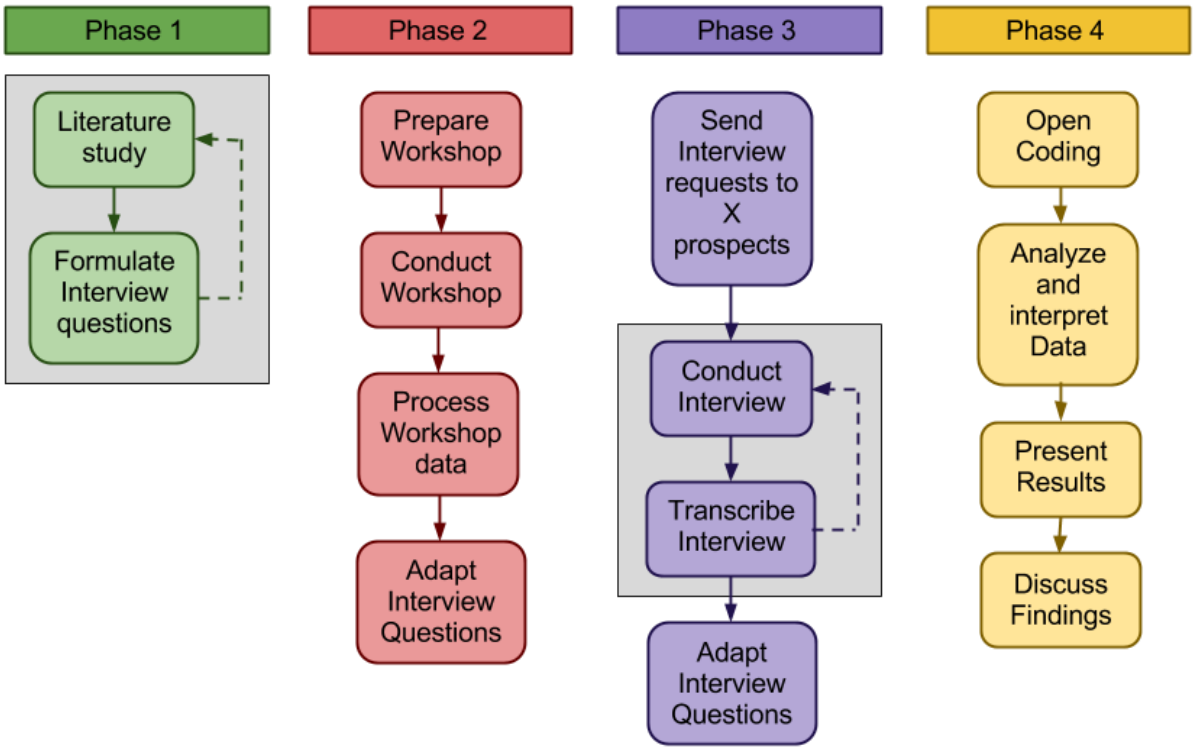

Fig.7: Process model of the research

4. Results

In this section the results from the Workshop and the Interviews are analyzed. First we give an overview of the main topics that came up during the workshop. Then, we give a detailed look at the results from the semi-structured interviews. Since all the interviews were conducted in Dutch, the results will be supported by translated quotes from the participants.

4.1 Workshop

After the data was grouped according to the question, we categorized the suggestions. These categories will shortly be covered in this section. The entire transcript from the workshop can be found in Appendix B.

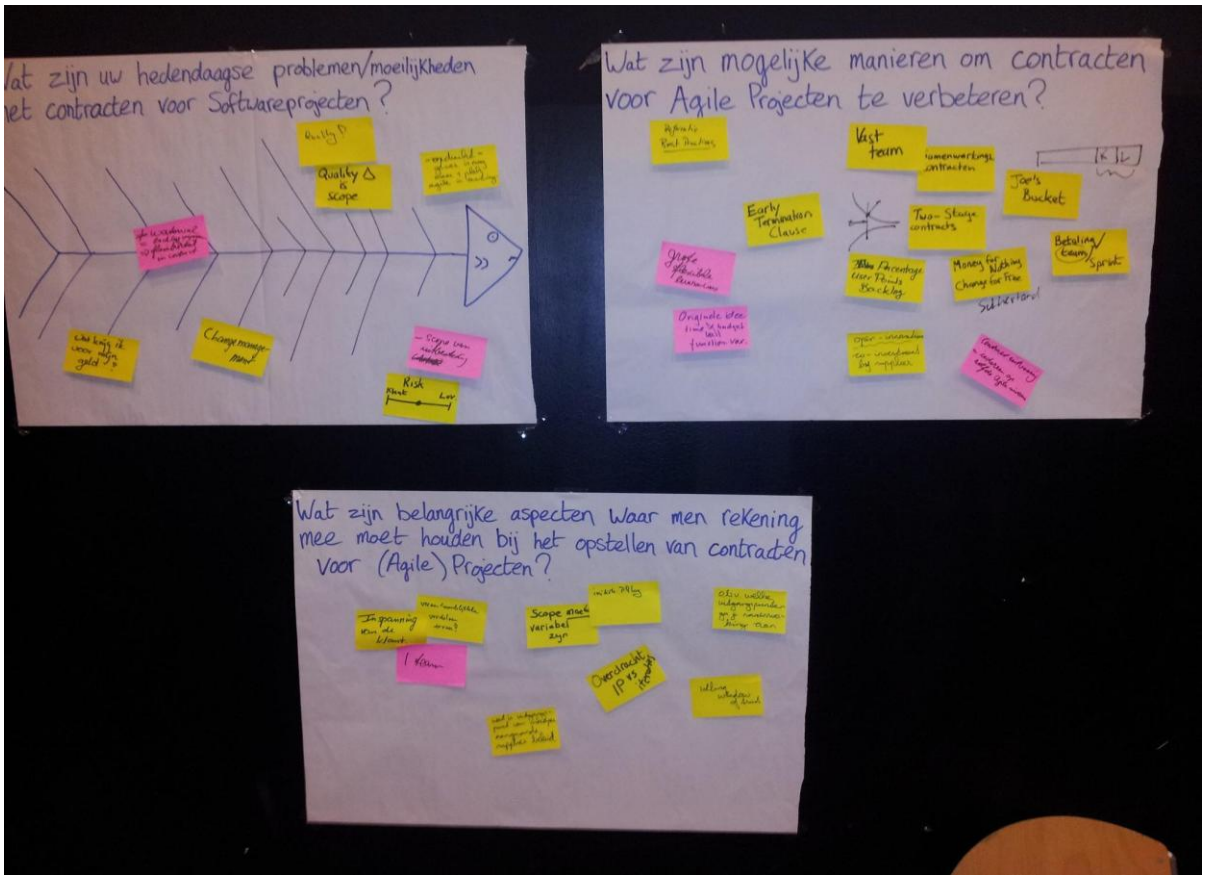

Fig. 8: The posters and the contributed suggestions

Figure 8 shows the posters shortly after they had been filled with the suggestions from the participants. The top left poster shows concerns the current *Challenges*, the one on the right revolves around possible *Solutions*, and the bottom one contains important *Aspects to consider* in contracting software projects. Table 2 shows an overview of the different categories created for each question. For each category, all the associated suggestions are shown (the exact writing on the participants' notes).

Challenges	
***Scope Change*:** The current way scope change is managed poses challenges.	"What do I get for my money?"
	"Change Management"
	"Quality is Scope"
	"Changing Backlog => Flexibility in Contract"
Quality: The customer's perception on the software quality poses challenges.	"Quality is Scope"
	"Quality?"
Outsourcing: Project with Outsourcing introduce challenges for contracting.	"Scope of Outsourcing"
Solutions	
Alternative Contract Types: suggested contract types that have not been seen in Agile projects very often.	"Collaboration Agreements"
	"Payment per Team per Sprint"
	"Two-stage Contracts"
	"Open – innovation: co-investment by supplier"
Scope Change: solutions aimed at overcoming the scope change challenges.	"Money for Nothing, Change for Free"
	"Original idea: Fixed Time and Budget, Variable Functionality"
	"Joe's Bucket"
Adjusted Project Practices: suggested changes in current project practices.	"Fixed Team"
	"Reference of Best Practices
	"Percentage User Points Backlog"
Early Termination: the option of quitting the project early.	"Early Termination Clause"
	"Money for Nothing, Change for Free"
Aspects to consider	
Collaboration: various important aspects in customer-supplier collaboration	"Trust"
	"Contribution/Effort from the Customer"
	"Based on what starting points do you start the collaboration?"
Scope of project.	"Scope must be variable"
	"Initial Product Backlog"
Intellectual Property: How is the intellectual property of the system transferred in an Agile project?	"IP-transfer vs. iterations?"
Development team	"1 Team"

Table 2: Overview of the Categories created from the participants' suggestions.

4.2 Semi-structured Interviews

In this section we will present the results that we have found from the semi-structured interviews. The size of the organizations varied heavily among the participants; from 10 to 145.000 employees. An overview of the participants is given in Table 3. In order to preserve the participants' confidentiality, we assign a code to each participant.

Participant code	Type of Organization	Position held in organization	Agile method	Contracting experience
P1	Research & Development	Jurist	none	Legal advisor in contract negotiations.
P2	Web Development	Co-Owner	none	Contract negotiation with customer
P3	Advisory	Partner IT	Scrum	Contract negotiation with customer
P4	IT Consulting	Head of ScrumMaster Course, and Agile Coach	Scrum	Experiences effects of different contracts in Agile projects.
P5	Software Development	Principal Agile Consultant	Scrum	Research in Agile Contracting

Table 3: An overview of the interview participants.

The rough structure of the interviews was as follows: We started with some general questions, by asking their position within the organization, how long they have been working there, and to what extent they are familiar with Agile and (Agile) software contracting. Then we asked a few questions about the organization, how projects are acquired and who is responsible for the financial aspects of the projects. We followed up with asking about their use of Agile methods, and what planning and estimating techniques they apply for software development projects. Not until we had a decent background on the organization, its projects, the project practices, and the participant's role in these projects, we started with the question about contracting. The exact interview guide can be seen in Appendix A.

In Figure 9 we present an overview of all the categories that we have distinguished from the different codes. The graph shows the amount of codes that each category contains. It can be seen that the most codes were related to the topic 'Payment Per Sprint'.

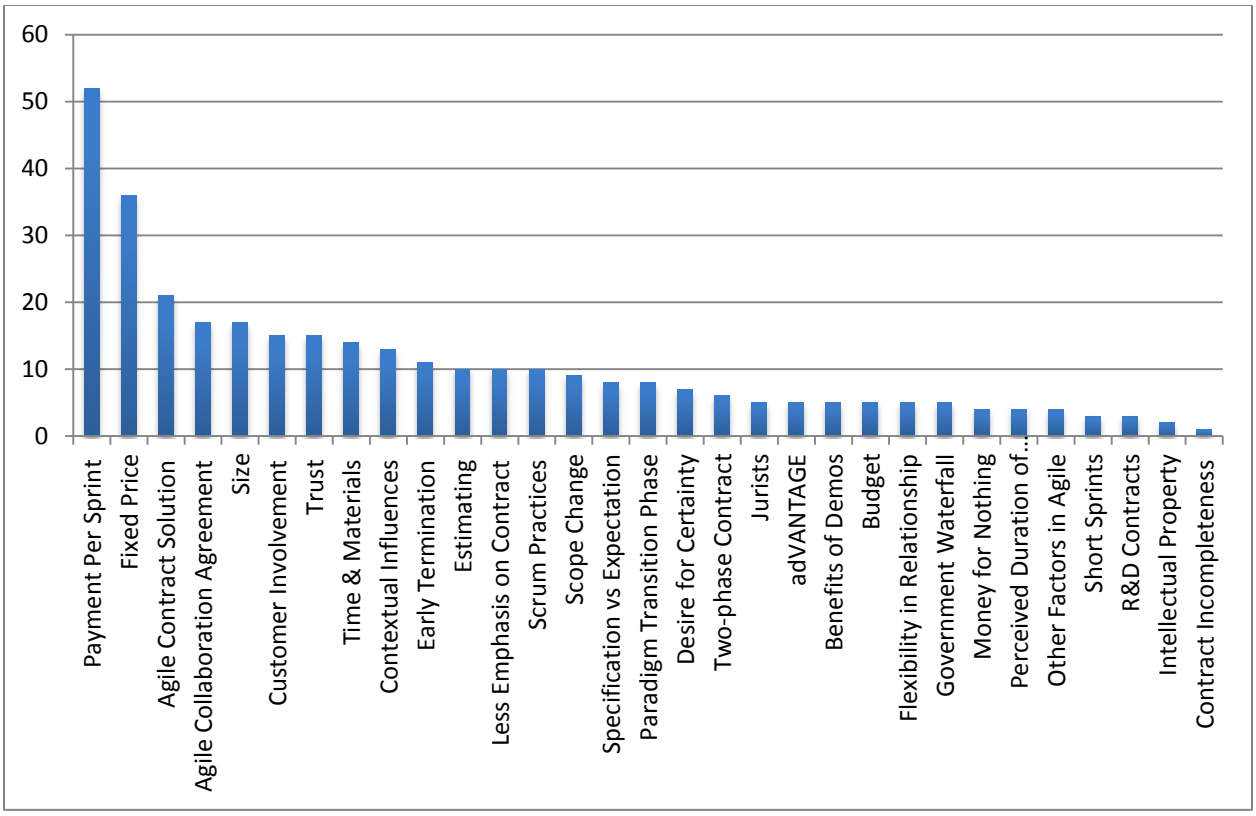

Fig. 9: Overview of the categories and the amount of codes they contain

Among the categories, some categories covered a wider topic. To clarify the data further, we have assigned the smaller-scoped categories as subcategories to the wider-scoped categories. The resulting structure of the categories can be seen in Figure 10. It can be seen that the (31) categories have been divided in 9 big categories and 22 subcategories. There are a total of 280 different codes. All the categories and their respective codes can be found in Appendix C.

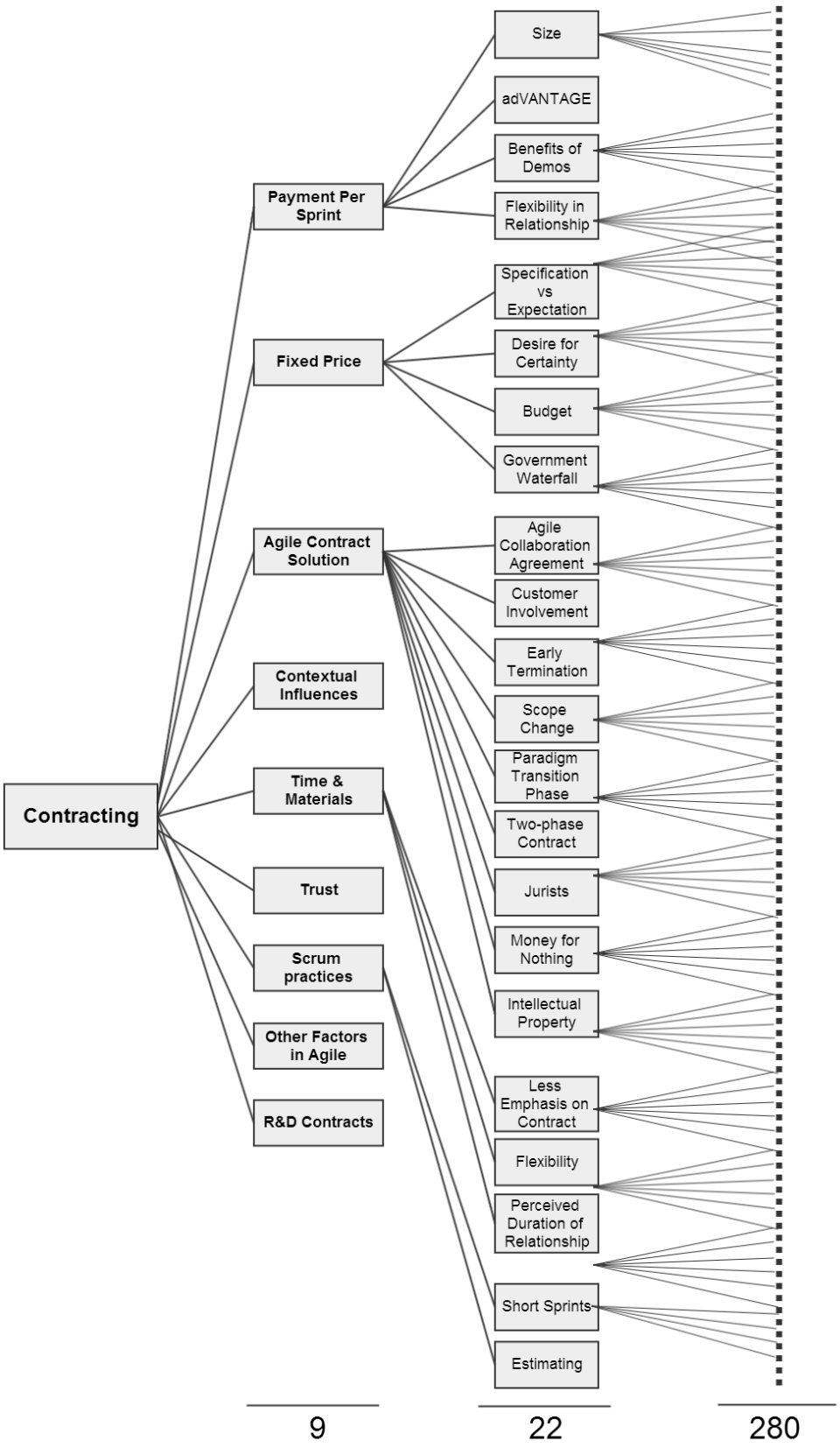

Fig. 10: Model of the relation between the Categories and the Sub-categories.

In the following subsections an in-depth explanation of the results is given. We have structured the subsections by using Figure 10 as guideline. Not that not all the subcategories from Figure 10 are given a separate subsection when explaining the results. Some subcategories are shortly mentioned in a few sentences, and some subcategories are not covered at all. Only the statements and findings that are most relevant and interesting (in our opinion) are covered. As can be seen in the interview guide in Appendix A, main topics in the interviews included: the participants' currently used contract type, challenges experienced in software contracting, and possible solutions for Agile Contracting.

4.2.1 Payment Per Sprint

We asked every participant what they think about the idea of doing the payments per sprint/iteration in an Agile project. The amount of experience and knowledge about Agile was relatively high among the participants that actively use Agile methods (two Scrum Trainers, and one IT partner). The opinions on Payment Per Sprint varied widely among them.

> *"No, I have never seen contracts done in that way."* — P4

P3 currently uses this method of payment with success, and is confident that this is a very effective way of handling the payments in Scrum projects. In contrary to the adVANTAGE pricing model that we saw in our literature research, this organization uses a fixed price.

> *"Billing is easy; I want to do that per Sprint. I want to do the billing as soon as possible. I often do that per Sprint, however it doesn't always work, because before you know it you're behind 2-3 weeks. But in general....there will be a bill per Sprint. It is a fixed amount per Sprint by the way, so whether or not we make the exact agreed hours or not, that doesn't matter. But we charge for every Sprint, and we try to do this as close to the end of the Sprint as possible. This also has to do with the current economic conditions. Every company is trying to invoice quickly these days, because payments are taking so long."* — P3.

Benefits P3 believes that one of the benefits for the *supplier* is the fast payment, and thereby some reduction of risk. The benefit of Payment Per Sprint for the *customer* is the flexibility that can be offered in multiple aspects of the relationship. In the contract an agreement is stated for a certain amount of sprints. Every time the agreed amount of sprints has been completed, the customer can buy more sprints. At the start of the project customers can agree to buy a small amount of sprints to get introduced the Agile approach. In these sprints the customer can witness the Product Backlog being created and managed, and he could receive the first part of the system.

“I always say ‘You only get on board for one sprint, and when you’re dissatisfied with a sprint, we can always just say goodbye’. Those are always enlightening insights for the customer.” — P3.

P3 elaborates that it is of course the idea to build the desired system together, but it is best to offer options like these to the customer just in case the collaboration works out differently.

"One of the problems is, that [the customers] think that the moment they agree to start the project, they have practically ordered an entire system. Although this is true to an extent, you start a relationship to build a system together. But one of the reasons why Scrum is used, is to take small steps to discover whether you're on the right track, and it's important to us that this idea of flexibility is also reflected into the relationships with the customer." — P3.

Another benefit that was mentioned by P3, is that the payment directly follows the acceptance of the sprint, which makes the payment more ‘natural’ for the customer.

“It’s very logical right? You have a sprint, you get a report with the sprint, you get a demo with the sprint, well then it’s pretty logical to add a bill to the sprint as well.
...
The payment schedule is not the main point. I think the main point behind this is that [the customer] just wants to put a checkmark that they’re satisfied with the result of the sprint [before making a payment].
” — P3.

Disadvantages A disadvantage that was mentioned was the fact that a contract like this would create a short term vision, which could lead to developers creating a *technical debt*. When there is a contract for a couple of sprints, the suppliers would then be focused on those sprints. They will try to please the customer by presenting as much functionality as possible at the sprint demos, and thereby putting the quality of the system at risk.

“...no I think it’s bad actually, paying per sprint.... Then [suppliers] will quickly start creating technical debt. Technical debt is a term that is used when you don’t completely finish things. For example when there are still some bugs [developers say] ‘well, those bugs are not so important right now so that can wait a while’. Or when the documentation [or tests] hasn’t been finished yet [developers say] ‘it will be good enough for the demo’ or ‘the customer won’t notice anyway’. But sooner or later you will start to notice the effects of that...because the bugs will keep piling up. So when you say ‘you will just be paid for the next few sprints’, then you take the risk of [developers saying] ‘well then I won’t do anything more than is necessary’. While when you say like ‘we have the intention of working together for a long period of time’, then you get

> *[suppliers saying] 'well then I want a satisfied customer for a longer period of time, so I will start fixing these bugs already...or I will put an hour extra into proper documentation, so I can benefit from that later'. So when you say 'payment per sprint' I think that won't work well."* — P4

Besides being fairly positive about Payment Per Sprint, P3 does mention that this payment structure requires discipline in the financial aspect. He stated that his environment — one of the largest professional services organizations in the world, that mostly does financial advisory — plays a role in the successful realization of this payment method.

> *"But from my perspective, this is the way to do billing in Agile. This is the way that it should be done, so I think it will grow, because it's just logical. However this requires discipline in the financial aspect, something that IT people are not very good at. I am being helped with that by my environment; in this house they always know everything financially." — P3.*

Size The size of the organization (supplier side or customer side) and the size of the project all play a role in determining whether this method of payment is feasible. P3 explained that smaller projects have an upfront price estimate, and a planning of the sprints.

> *"But smaller projects, for example with the government, there you first have a price estimate and a planning of the sprints. So formally they also buy sprints, but there is a bit more specification of what the system should do....so a bit more towards a waterfall-like description of the end result. But when I look at other Agile contracts, this is pretty Agile." — P3.*

P2 (Web development) did not see any added value in doing the payments per cycle for his organization. This is because he claims that does not have very complex cases.

> *"Seriously, we don't have complex cases. It does not make sense for us, it is not relevant for us." — P2.*

4.2.2 Fixed Price

Since we asked every participant about their opinions and perceptions on the current challenges in Agile Contracting, 'Fixed Price' was mentioned many times. As a confirmation of what we learned from the research papers that we studied and the workshop, we often heard that the customer's desire for certainty and upfront specification opposes a challenge in Agile projects.

Benefits The main benefit for Fixed Price contracts is the certainty that it provides to the customer. The main problematic aspect of Fixed Price contracts is that it also incorporates a detailed upfront specification (fixed scope) and a fixed deadline. However the web development

company gave us an example of Fixed Price contracts being used in small projects without making a complete upfront specification (i.e. no fixed scope). This leaves them with the benefit of giving the customers certainty.

> *"Our customers have no idea what the process looks like of developing something from something that does not yet exist, to something that exists. That's why they are very careful and restrained. We try to give them as many things possible to get them [feeling like] 'I feel comfortable with this, I can agree to this'. Offering assurance, that's what it's all about.'* — P2.

To manage possible unforeseen contingencies, they always plan some financial buffer during the price estimation.

> *"When we discuss the requirements, then that's always pretty do-able on one page. Then it's a matter of discussing, but we are not too strict with that. In estimating our price, we always make sure that there is a financial buffer for us, and we use that buffer for contingency planning....and if along the way the customer says like '...I'm not satisfied, I actually wanted it like [another way].' then we are not very difficult when it comes to that."* — P2.

Problems What was mentioned multiple times is that the detailed upfront specification is the most problematic factor in Fixed Price contracts.

> *"The reason why fixed price is so problematic is because the entire project you're stuck to the scope that was specified in the contract."* —P5.

An important distinction was often emphasized: the distinction between *specifications* and *expectations*.

> *"[described a previous case where] the system was delivered, it did not meet the expectations, but it did meet the specifications, and that's typically where Agile Contracting should provide a solution...it's way more important to meet the expectations than the specifications. Of course you do need specifications, but those should be more like goals, 'what is the intended goal of the system?'. That is, however, difficult to state [formally], so it's better to verify the system against the goals instead of the specifications."* — P4.

Government Waterfall Something that was mentioned by multiple participants was the regulations that are associated with software projects for the government. Governmental software projects should always have a complete upfront specification, price estimation and deadline.

"The contract model, that's where you really see the friction between waterfall and [Scrum], because you keep seeing that customer wants to specify as clearly as possible what they are going to get — in functionality, not in man-hours, which is basically what a sprint is really. And especially in the government, all the standards are aimed towards waterfall-like constructions, where everything is specified very clearly. So that's where the most friction is. That means that we're actually always waterscrumming, because...you rarely see customers that completely trust that it will be alright, which is justly of course." — P3.

"And we also do work for the government. I know they understand scope creep, because they've been doing projects for a long time, but they always say 'we need fixed price'. ...We work with modules and even if they know that the work is going to inflate over time, they prefer to say 'that's a separate project'." — P2.

4.2.3 Agile Contract Solution

In our interviews certain suggestions were shared for new ways of Agile Contracting. The Agile practitioners among the participants all shared elements, that they believe should definitely be implemented in order to improve Agile projects. The background of P5 was already known due to his participation in our workshop, and there we discovered that he had done personal research on Agile Contracting. Therefore, we could dedicate the entire interview on hearing out his perception on solutions on Agile Contracting. He presented a set of key elements that he believed are essential for overcoming the current challenges in contracting for Agile projects. In this subsection we will explain these key elements and show the similarities and differences with the perception of other participants. Before we present the suggested solutions, we first highlight the challenges that these solutions aim to overcome.

Customer Involvement An aspect that often poses a big challenge in Agile projects is the lack of customer involvement, as is also highlighted in previous research (Hoda et al. 2010). P3 explained his current experience with regard to customer involvement, and underlines why it is so important.

"We also invite the customer, and if he joins in more than once a week, then that can be seen as often. Something that also happens often is that the product owner goes and works at the customer's site. In that case he is present [with us] less than you'd ideally want to, but then that's also understandable.

...

Because we are together with the customer a lot, the customer sees that a lot of work is being put into [the solution]. I think that's a really big advantage of this approach, that the customer sees who is working, what he's doing and what he has done." — P3

In other research articles, the problem of lack of customer involvement has been mentioned (Hoda et al. 2010). However one of our participants mentioned a difficulty with regard to customer involvement originating from the suppliers side. P4 said that he can sometimes notice a certain level of uncomfortability from the suppliers, when the customer or a consultant joins in on the developers' meetings.

> *"From my own experience...when I'm a consultant and I say [to the team] 'I want to join your estimation sessions' or 'I want to join your retrospective session'..every meeting that you guys have, I want to be able to just sit in to see 'how do you guys do things around here?' and 'what could we improve here?'. And I can notice that sometimes there is a lot of resistance from the people like 'well, we'd prefer that you don't join in...'....Well in general it is accepted for demos, and for planning sessions they also think it's pleasant [if the customer is attending]. But for other sessions it's sometimes perceived as monitoring, like 'You don't trust us?'"* -- P4

Scope Change P5 stated that the substantial problem of scope change in Agile software projects should definitely be taken into account when looking for improvements in Agile Contracting. This traces back to the problem of complete upfront specification of the system features. When all the features have been formally specified upfront, then there is very little room for any scope change.

> *"Ideally in Agile Contracts, scope change should be freed: whenever a customer wants to change the scope during the project, this should be possible without being stuck to any upfront specified features. This is currently the case with Fixed Price contracts, because Fixed Price also implies Fixed Scope and Fixed Deadline. So whenever a customer is dissatisfied with the system that is delivered, and wants some changes, the supplier says 'nope, can't do it, look at the features that are stated formally in the contract.'"* — P5.

First, he mentioned Jeff Sutherland's attempt at freeing scope change, which is the 'Change for Free' part from his strategy 'Money for Nothing, Change for Free'. Change for Free allows the customer to change one feature from the Product Backlog that has not been implemented yet, and swap this for a new feature of the same size. However this can cause a bargaining-session with every change request.

> *"There is 'Change for Free', which is basically Jeff Sutherland's attempt at freeing scope change. Change for Free allows the customer to pick a work-item from the Product Backlog, that hasn't been implemented yet, and replace this with a work-item of the same size. That is a pretty good idea. However, the problem with this is, that with every change request they will start bargaining about whether the work-item is of the same size or not."* — P5.

He would then explain two other possible solutions to free scope change. The first possibility that he suggested was aimed at defining and specifying a 'common goal'. This could be done by specifying the customer's intended underlying goal of the system instead of a set of system features. However he then added that this turns out to be very difficult (if not impossible) for jurists to formally state in a contract.

> *"What happens now is that the software supplier promises the customer a particular 'box' of features....One way of doing this (freeing scope change) is to create a common goal....by very concretely specifying the customers' underlying goals of the system, for example 'we can now make 100 units per month, but [with the new system] we want to make 200 units per month'...when I pitched this idea to jurists, they all said it would be very difficult to measure the amount of influence that the system has on the results."* — P5

Agile Collaboration Agreement The second possible solution to free scope change that P5 explained, was aimed towards specifying the nature of collaboration between the two parties, for which he proposed two different approaches: forming a joint-venture, or setting up 'Collaboration Agreements' for Agile software projects.

> *"The other possibility of freeing scope change is by collaboration. A way this can be done is to form a joint-venture. Then the software supplier gets a share of the profit...I believe this is a possible way to do it. Another option is aimed towards Collaboration Agreements, and when I proposed that option, the jurists [reacted] 'Wow, yes, well that's something we CAN do [as opposed to specifying the customer's goals]'....that would contain things like the responsibilities and expectations of the parties...And that's something [software engineering jurists] are currently looking into with Agile Contracts."* — P5.

The importance of proper collaboration and *trust* in Agile projects was also mentioned by other participants. P3 and P4 both mentioned that they believe that the customer should monitor the progress of the project more. When asked their opinion on what would be the ideal way of doing contracts for Agile, many ideas were aimed towards agreements on collaboration. According to these participants (P3 and P4), the responsibilities and expectations of the customer should be clarified and stated in Agile Contracts for more effective collaboration, and a higher chance of project success.

> *"I think the best would be a pure Agile way, in which you, as a customer, can watch over the progress...very frequently. I would think like 'you as a team get all the freedom to do what you have to do in an Agile way, within certain boundaries, but that as a customer you — which (kind of) is a part of Agile — are present with the team very frequently, almost on a daily basis, so that you can see if there are progressions...And of course,*

when you see that the team performs well, then you give them a bit more freedom, but if you get the impression of 'this doesn't suffice', well then as a customer I would be on that a bit tighter....And it's very possible to state that in a contract I think." — P4.

"On the one hand customers should have more trust, on the other hand they should have less trust. They should have more trust in that you can get through it with a good supplier, that you don't have to completely specify how the end result should look like then. That's where they should have more trust in. They should have less trust that, once they have started the relationship, the supplier will make it all alright. They have to critically watch the supplier. Give feedback during the retrospectives, join the discussions, if necessary end the relation [in a formal discussion], instead of letting the bills pile up...So yes, go ahead and trust that it will be okay, but monitor that it will be okay as well." — P5.

In our interview with P1 (R&D jurist) we got a detailed explanation of the structure of 'Research Collaboration Agreements'. There we saw how these agreements cover important project variables very well, even though the end result in Research & Development is rather difficult to specify upfront. Research Collaboration Agreements state the way the involved parties (researchers or organizations) are to collaborate with each other, and to what (research related) goal they are working. In these agreements, it is most important to specify which results are going to belong to which party. This concrete example of a Collaboration Agreement gave us a sense of the possibilities of Collaboration Agreements in Agile software development.

Early Termination Our interview guide also contained a question about the option of early termination. Something that was often mentioned, was the 'Money for Nothing' part of Sutherland's 'Money for Nothing, Change for Free'.

"Something that I often explain in the [ScrumMaster] courses, is the 'Money for Nothing, Change for Free' principle. Where you...make upfront agreements, that the moment you terminate the project early, you say 'you still get paid 20% of the remaining budget, and you don't have to do anything for it anymore'....and then [the customer] saves 80% [of the remaining budget], and that 80-20 rule is a guideline. But it [(using the Money for Nothing principle)] is actually rarely applied in practice." — P4

The 20% that is given to the supplier can partly be seen as a bonus for delivering enough business value sooner than expected. The other part can be seen as a compensation for the fact that the supplier's resources have to be allocated to another task unexpectedly, so that 20% is basically also a risk premium (P5). Just like P4, P5 also shared from his own experience that he has not seen this being put to use very often, although he believes that this element is very much required in the ideal Agile Contract. P3 has mentioned the possibility to terminate the project early multiple times. However, he had negative connotations in mind with regard to the reason of

the early termination. When we explained the idea of possible early termination with the reason being that the customer is satisfied enough with the delivered features of the system, he explained why he thinks won't be used very often.

> *"I have never seen that in a contract. It's very funny, because it's actually very logical to do it like that. That's very good. Although it can be seen that very many customers have a lot of trouble with de-scoping, it's very hard to decide that you don't want something, even though it barely has additional business value. Customers are not used to that."* — P3.

Two-phase Contract To handle the 'time and money' aspect of Agile Contracting, P5 recommended a contract model called a 'Two-phase Contract'. This is a contract model that is not seen in research literature. Figure 11 (provided by P5) illustrates the structure of a Two-phase Contract. Here it can be seen that the Two-phase Contract is based on the 'Cone of Uncertainty' (McConnell, 1997). The vertical axis represents the 'uncertainty-factor' in a project, and the horizontal axis represents time. At the beginning of a project, there is a lot of uncertainty (about the scope, the price, the duration etc.). This gradually declines along the duration of the project.

The 'Two-phase contract' uses this idea, and divides the project into two phases. The first phase is a relatively short phase that is aimed at getting through the initial uncertainty, and creating a base of trust between the customer and supplier. It was recommended to use Fixed Price for this phase, since it is relatively short, and the required effort can be estimated relatively easy. The example in Figure 9 shows a first phase of three sprints. By then: the productivity of the team has been witnessed by the customer, the scope of the project has been forecasted (at least way more accurate than before the start of the project), the main impediments have been identified, an increment of the system has been delivered, and above all more trust has been established. When those elements are already in place, the parties could basically make any contract type work well for the second phase (e.g. Fixed Price, T&M etc.).

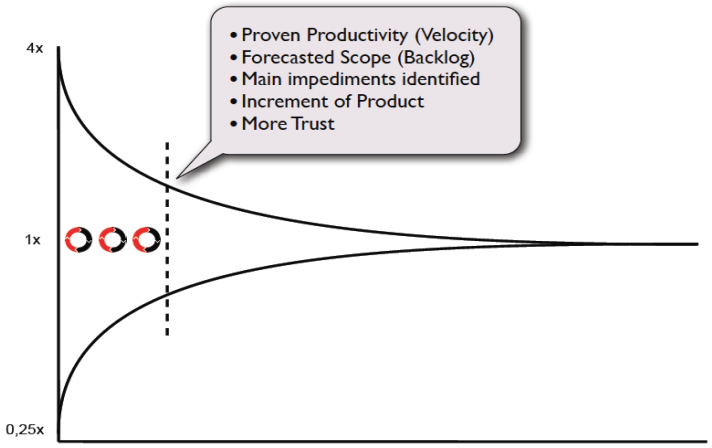

Fig. 11: The 'Two-phase Contract'

Paradigm Transition Phase We were also curious to know the participants' opinion as to why they believe that such solutions haven't been implemented yet. The common answer revolved around the fact that we are currently in a Paradigm Transition Phase: the world of software engineering is gradually switching from the Traditional approach to the Agile approach.

P4 explained that people in software development currently still have work attitudes that stem from the Traditional methodology. He elaborated on this by giving an example from his past work experience.

> *"In my courses I often use a comparison: before this I've worked at [phone company] for a while, just during the time when mobile phones did not need to have a long antenna attached to it anymore...So we started selling mobile phones without antenna... And what happened? Nobody bought them. We started investigating [why they didn't buy them]... Then we figured out that in their perception, mobile phones simply are supposed to have an antenna. So when we figured that out, we attached some plastic thing to it that resembled an antenna, and then we started selling them again! Finally, when that generation had become used to the fact that the antenna doesn't have any added value, they could start selling antenna-less phones... I notice that in the world of Agile Contracting we are also in that transition phase."* — P4.

P5 explained the paradigm shift with a slide from his presentation on 'Agile Contracts' (Figure 12).

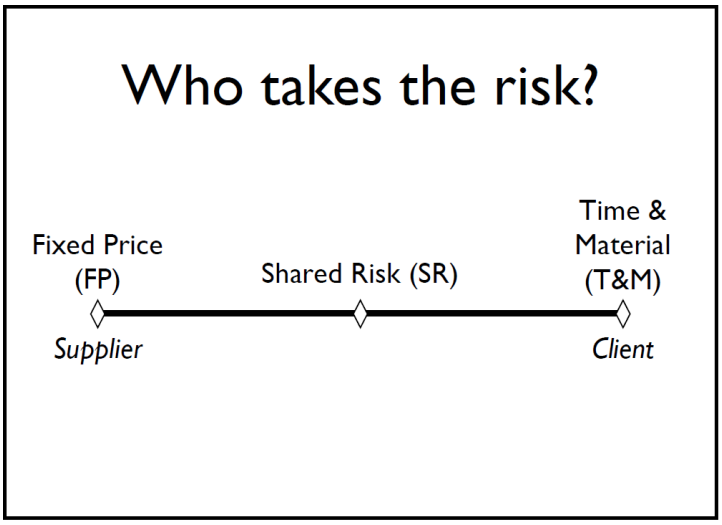

Fig. 12: Shift of project risk illustrated.

> *"We are currently in a paradigm shift. First there were Time & Materials contracts where all the risk was with the client... Then they decided to throw it all around, and put all the risk on supplier (Fixed Price). And with Agile Contracting we are trying to put the*

risk slider in the middle, where the risk is evenly shared between the customer and the supplier." — P5

Jurists When asked what current challenges are in contracting from the jurist-perspective, P1 stated that when jurists are making a contract for (two) parties, they often don't really know what the underlying intentions are from these parties. This somewhat limits them in their ability of setting up correct and accurate contracts.

"What I often encounter is that many times jurists don't know, what the intentions are of the parties." — P1.

This was confirmed by P5, who explained that he had conducted a workshop at a Dutch organization of jurists. In this workshop he explained Agile to the jurists and he presented a case, that was aimed to get the jurists thinking about possible solutions. So far they believe that a solution based on Collaboration Agreements is the most feasible, and P5 said that it is largely a matter of time before the effect of 'Agile Collaboration Agreements' is discovered.

4.2.4 Time & Materials

One particular participant is currently using a Time & Materials contract. This contract contains a certain amount of feature points that has to be realized, instead of a specification of the system that is supposed to be delivered. The customer does not precisely monitor the actual amount of man-hours that has been burnt every month. The suppliers maintain a general overview of the amount of hours that has been put into certain activities, and that amount will be paid by the customer. Our literature study has taught us that the main problem with T&M contracts is the fact that the supplier has no efficiency incentive. Therefore we were curious for this participant's opinion as to what the positive and negative aspects are with T&M contracts. He told us that the big advantage is the ability to really work Agile. The negative aspect is that it creates a very short term view, which could come at cost of long term aspects.

"The big advantage is that you are really able to work in an Agile way. ...The disadvantage is..the short term view is very strong in this. [Developer's thought] 'I have to deliver something now, because I have to keep the customer satisfied on the short term', and..this could come at cost of certain longer term affairs. ...And in the end you actually build a relationship where the customer realizes that 'I can't have my system developed by someone else tomorrow, so we are in fact dependent on each other...So the negative aspect [of T&M] for the customer is, that you think that you have freedom, but in reality it's limited." — P4.

5. Discussion

Among suggested solutions we have seen new contract ideas for Agile projects, and modifications to contracts currently used in Agile projects. The new ideas were:

- Collaboration Agreement for Agile
- Two-phase contract
- Goal specification instead of feature specification
- Forming a 'Joint-venture' for an Agile project.

Collaboration Agreements for Agile are still subject to investigation by jurists, since they have to make sure that the contracts are legally feasible, correct and complete. *Two-phase contracts* seem like a good solution that is not difficult to implement. However, formally *specifying the intended goals* of the customer (with regard to the system) did not seem like a very feasible solution, since jurists claimed that it would be difficult to formally measure the supplier's influence when the customer has reached the goals. Also, the challenges and the amount of effort involved in forming a Joint-venture for Agile projects are yet to be clarified.

Many suggested solutions could be seen as a modification to contracts currently used in Agile projects. Among these modifications were: 'Early Termination Clause', 'Money for Nothing, Change for Free', 'planning financial and time buffers for the project' and 'Fixed Price Per Sprint'. These suggested solution have all been used by some of the participants.

The wide variety of the participants' backgrounds gave us multiple different perspectives. This could help us get a more complete image of the possibilities in Agile Contracting. From our results, we believe that there is not one specific contract type that is best for all Agile projects. There are many variables that can make software projects very different from each other (e.g. size, amount of uncertainty, experience of the parties etc.). The context of a software project affects which contract type is the most suitable.

We believe that it is possible to make a guideline that can help choose the most suitable contract for a specific situation. And the 'new', rather unknown contract types that we have seen in our research could help in this matter, since they could be good additions to the current range of available contract types for Agile contracts. A guideline for choosing the most suitable contract in a particular situation should (roughly) distinguish the context of the project according to a set of variables. One of the variables that should be used in such a guideline is *size* (or duration) of the project, because this in turn impacts the amount of *uncertainty* and *risk* in the project. In a very big project, there is relatively much initial uncertainty (and therefore risk). In that case it may be helpful to use a Two-phase contract, wherein the different project dynamics during the uncertain starting phase of the project are taken into account.

Another important variable is the *customer's experience with Agile* (whether the customer knows how proper collaboration should be), because if there is little experience with good Agile customer involvement, it may be a good idea to opt for a Collaboration Agreement. This will enforce more input and feedback from the customer into the project, which in turn leads to a higher chance of the project being successful.
Trust is also a very important factor in a project, however this is hard to measure. When a customer is about to start a project with a supplier that he has no (direct or indirect) experience with, the expectations of the project are likely to be very different than for projects with known associates. And when a project is started on an existing bond of trust between the customer and the supplier, then the challenges related to contracting are already significantly diminished.

Our research results have shown us that Fixed Price and Time & Materials contracts can still be used successfully in Agile projects, as long as it is used in the right context. Fixed Price contracts can still be a good option in Agile projects if there is very little sense of ambiguity in the scope of the project (i.e. when the features are not very complex), because scope change will still be problematic. This way the suppliers can still somewhat take an Agile approach in the project, while the customers feel secure about the project because of the fixed price, scope and deadline. T&M can be a very simple way of contracting for Agile projects, if the customer trusts the supplier. However this should be combined with a sufficient amount of customer involvement, and the suppliers should fully allow the presence of the customer coming in to check the progressions of the project. Something of which P4 stated, is rarely happening in practice.

With regard to the Payments Per Sprint contract that we saw in our interview with P3, we do believe that a fixed price per sprint is a better solution than what we saw in the adVANTAGE pricing model. The fixed price should be calculated according to the team's burn-rate (Velocity), and whether the sprint goals are reached in a shorter amount of man-hours or longer, that will not be an issue. And the parties could always agree on some clauses that involves significant deviation of burned hours. The successful implementation of Payment Per Sprint-contracts in P3's organization could perhaps have been influenced by the fact that this organization has a powerful position, since it is one of the largest professional services organizations in the world. They have as much financial expertise as any organization. However it is not very unordinary, because it can be seen as a sort of T&M contract with a fixed price. Since this payment structure has been implemented by an organization full of financial experts, we could imagine that a detailed profitability analysis has been done before it was decided to use this payment structure.

Another thing that became clear during the research is the fact that there is a very different perception of the meaning of the word 'contract'. Jurists (and other experts in the field of law) look at every element of a contract, while people from the software engineering world have slowly shaped 'their' perception of a contract with an emphasis on the agreements on the price, financial structure, scope and deadline of the project.

6. Conclusion

We started this research with a general interest in making contracts more Agile. In particular, the idea of doing the payments per sprint/cycle seemed appealing to us.

> ***Research Question:*** *Agile projects are developed in cycles which means that developers also can be paid per cycle. What are possible solutions to make contracts more Agile?*

In our interviews we had one participant who currently does contracting with the (fixed) payments per sprint. Benefits that were mentioned included: *faster payment (i.e. reduced risk for supplier) and providing flexibility/options to customer*. So this could in fact be a possibility of making a contract more Agile. However, it is not per se a contract that will ensure successful Agile projects. Another participant mentioned that a possible disadvantage of this approach could be a *short-term vision* from the supplier-side, which increases the chance of creating *technical debt.*

In our research we have been introduced to rather new ideas regarding contract types that have not yet been seen in Agile software projects. These contracts are not specifically aimed at 'making contracts more Agile', but they are aimed at specific challenges that are often posed in Agile projects. We discovered what some of the biggest challenges in Agile projects are:

- *Lack of Customer Involvement:* if the customer is not involved in the project enough, it will be more difficult for the supplier to develop the right system.
- *Difficulties in Scope Change*: if the entire scope of the project has been completely specified in the contract, scope changes become very difficult.
- *Trust*:
  -*Lack of Trust*: Before the project has started, customers tend to lack trust in that, by using the Agile approach, the supplier can deliver the right system without having to specify the system completely upfront.
  -*Too much Trust:* When the project is started, customers tend to have too much trust in that the supplier will deliver a good system, which in turn leads to less effort being put in the project by the customer.

The new contract types that we have seen should overcome these challenges. The most interesting suggested solution was using a Collaboration Agreement that is designed for Agile software projects. A Collaboration Agreement specifies the responsibilities and expectations of the involved parties. In the Agile context, this can be used to ensure sufficient customer involvement. This contract also does not specify the features of the system, so scope changes can be done freely (as intended in the Agile framework). It is aimed at executing the project in the most Agile way possible. When customers are involved in the project more, they can oversee the progress of the project, and give feedback more frequently.

As the research progressed we gradually found out that there are many aspects that affect software projects. Even though they are all called ‘Agile software projects’, they can vary greatly, and therefore for each project the most suitable contract can also be very different.

The findings from our research, contributes much interesting information to current literature on Contracting for Agile projects. We introduced multiple new ideas for future contracts in Agile projects. We summed up important recurring problems in Agile projects, and we explained how the new contracts aim to solve these. We discussed their feasibility and the opportunities that they present. These findings can lead to more concrete solutions that will improve Agile projects in practice. Our research also sums up a set of modifications to currently known contract types that have already been applied successfully in practice, but they are not yet very known. Our research could introduce Agile practitioners to these solutions, that are very much applicable.

Future research

Something that could be a great contribution to the Agile software engineering world, is a more concrete overview (i.e. contract choice guideline) of which contract type would be suitable in certain situations. To achieve this, more research should be done into the variables that define software projects, and how these can be measured (so that people can compare their own project against the variables described in the guideline). Also, more research should be done into the possibilities of ‘Agile Collaboration Agreements’, and ‘Two-phase Contracts’. These contracts should be tested in practice, and the effectiveness should be analyzed to check the feasibility of these solutions.

To get a better understanding of the juridical possibilities and challenges with regard to Agile Contracting, interviews should be conducted with jurists that are specialized in software engineering.

References

-Adolph, S., Hall, W., Kruchten, P.: *A Methodological Leg to Stand On: Lessons Learned Using Grounded Theory to Study Software Development.*

-Awad, M. A.: *A Comparison between Agile and Traditional Software Development Methodologies.*

-K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries et al., "The agile manifesto," http://www.agilemanifesto. org/principles.html, vol. 7, no. 08, p. 2009, 2001.

-Book, M., Gruhn, V., Striemer, R.: *adVANTAGE A Fair Pricing Model for Agile Software development Contracting.* 2012

-Boehm, B., Turner, R.,: *Management Challenges to Implementing Agile Processes in Traditional Development Organizations*. IEEE Software, 2005.

-Van Cauwenberghe, P.: *Agile Fixed Price Projects part 2: "Do you want agility with that?"*

-Crocker, K.J., Reynolds, K.J.: *The Efficiency of Incomplete Contracts: An empirical Analysis of Air Force Engine Procurement.* The RAND Journal of Economics. 1993

-Doraijaj, S., Noble, J., Malik, P.: *Technical Report 12-08: Knowledge Management in Distributed Agile Software Development.* 2012.

-Dybå, T., Dingsøyr, T.: *Empirical studies of agile software development: A systematic review*. Information and Software Technology, 2008, pp.833-859

-Glaser, B., Strauss, A. L.: *The Discovery of Grounded Theory*. Aldine, Chicago, 1967.

-Gopal, A., Sivaramakrishnan, K. : *Research Note — On Vendor Preferences for Contract Types in Offshore Software Projects: The case of Fixed Price vs. Time and Materials Contracts*. 2008

-Hart, O.D., Moore, J.M. : *Foundations of incomplete contracts*. Review of Economic Studies 66, 1999. pp. 115–139.

-Hoda, R., Noble, J., Marshall, S.: *Negotiating Contracts for Agile Projects: A Practical Perspective*. XP, LNBIP 21, pp. 186-191, 2009.

-Hoda, R., Kruchten, P., Nobel, J., Marshall, S.: *Agility in Context.* ACM, 2010.

-Krebs, J.: *Agile Portfolio Management*. Microsoft Press. 2008

-McConnell, S.: *Software Project Survival Guide,* Microsoft Press. 1997.

-Nerur, S., Balijepally, V.: *Theoretical Reflections on AGILE DEVELOPMENT METHODOLOGIES.* ACM, Vol. 50, No. 3. 2007

-Royce, W. W.: *Managing the Development of Large Software Systems.* Technical Papers of Western Electronic Show and Convention (WesCon) , 1970, 25-28.

-Saussier, S.: *Transaction Costs and Contractual incompleteness: the case of Électricité de France.* Elsevier. 2000

-Schwaber, K.: *Agile Project Management with Scrum.* Microsoft Press. 2004

-Stevens, P.: *Contracting for Agile Software Projects.* 2009

-Thorup, L., Jensen, B.: *Collaborative Agile Contracts*. Agile Conference. AGILE 2009, pp. 195-200.

-Williams, L.: *What Agile Teams Think of Agile Principles.* ACM, Vol. 55, No. 4. 2012

-Yin, R. K.: *Case study research: Design and methods*. (Vol. 5). SAGE Publications, Incorporated. 2008

Appendix A: Interview Guide

Organization:

- What is the structure of your organization?
- Does your organization have a central administration of projects (Project Portfolio Management office)?
- Who is responsible for the financial aspects of your projects?
- How does the process of acquiring projects look like in your organization (from initial project request to start of development)?

Agile Methods:

- How long has your organization applied Agile methods?
- What kind of Agile methods do you apply (Scrum, Lean, XP)?
- What kind of agile project practices do you apply (e.g. Frequent demonstrations to customers, standup meetings, development in iterations)?
- What is the duration of each sprint in your projects?

Estimating and Planning:

- What kind of planning techniques do you apply for software development projects?
- What kind of estimating techniques do you apply for software development projects? What are your experiences with these techniques?

Agile Contracting:

- I am going to ask you questions about software contracts now, and when I say contracts I am interested about the agreements on billing, the scope/size of the project, the deadline, and the budget.
- "What kind of contract models have you applied on your Agile projects? How does the current contracting system work?
- Have you ever done payments per sprint in a project?
- Which contract type is the most popular among your customers?
- What are the positive aspects of this contract type for you? And for your customer? What are the negative aspects?

- What are important aspects to consider when contracting Agile software projects?
- What do you think could be improved in your current way of contracting software projects?
- What is currently your favorite contract type to use in an Agile project?
- How are subcontracting cases handled in your contracts? Do they often provoke difficulties?
- What are the challenges you're facing with current contracts in Agile projects?
- Given the difficulties in estimating **big** projects upfront what would you think of a different contracting model, for example a contract that incorporates billing with payment per cycle?
- Have you ever tried explaining the negative effects of fixed price-contracts when your customer preferred such a fixed-price contract?
- In some Agile contracts it's possible for the customer to end the project earlier than originally planned, when the customer doesn't think it will be worth the money to continue the project. In some cases the software supplier is given a compensation/bonus (20% of the remaining project price). What do you think about the option for the customer to end the project **early**?
- What do you think about the option to buy extra sprints after the project has reached its original end-date, instead of ending the project and setting up a new project. That extra agility, do you think you could just offer that to a customer and make that work in practice now?
- [show him/her the "Fair pricing model for Agile Projects" and explain a bit how it works] What do you think of this model? Do you think you could use this right away? Why or why not?
- What are the benefits of new Agile Contracting in your opinion?
- Is it difficult to try a new contract type like this in practice (i.e. to convince the customer to sign this)?
- Why do you think Agile Contracting hasn't been adopted in practice yet?

Appendix B: Workshop transcript and memos

Due to the limited time for the workshop, not all user suggestions could be discussed. Therefore some suggestions could not be explained by the participant.

Current challenges in contracting Software Projects:

-Change management: Confusion/miscommunication about the scope that is agreed upon.

-What do I get for my money: customer is

-Scope of Outsourcing: There was a man who was currently doing/outsourcing a project in India for [big Dutch bank]. He had some trouble with that concerning the Service Level Agreement.

-Quality: The customer is being annoying regarding the definition of done and the level of quality that is demanded before the software is accepted by the customer.

-Quality is Scope:

-The customer is only Agile in just 1 place in act: ?

Possible solutions to improve Agile contracting:

-Collaborative/Collaboration Contracts: A contract that does not specify any scope. It only states that the two parties are obliged to collaborate with eachother in a specific way, and that the product (software system) should deliver specific business value. For example, it states that a representative of the customer should be present on site with the supplier for a specific time.

-One team: No-one can leave before the contract has finished.

-Payment *per team* per cycle. He said that he has witnessed a payment per cycle Agile contract once, and that there was a big mess with the payments

-Early Termination Clause: they were very positive about that. The supplier gets a bonus for satisfying the customer before originally planned. The customer is very happy and very likely to follow up on the contract (?) and quickly signing another contract with the same supplier. Other organizations hear about the good project, and want to get in line for that supplier.

-**A Reference of Best Practices**: Keep a reference list of best practices of different previously closed contracts (i.e. learning from the past successes).

-Fixing Deadline and Price, but keep Scope Variable: He called it the 'original way of Agile Contracting'. I also saw this approach encouraged in a YouTube webinar from an Indian Software company.

-Two-Stage Contracts: A contract type where the first stage is a contract in which the customer and supplier are obliged to come up with a Product Backlog. The Once the Product Backlog has been established and the implementation can begin, the second stage of the contract can be negotiated. At this phase both parties have a better feel for how the project will look.

-**Money for Nothing Change for Free:** A Contract intended for SCRUM. By Jeff Sutherland.

The customer should participate in the Scrum Team, and contribute to a couple of things (Backlog, estimation of work items, definition of done, Sprint planning meeting, Sprint review meeting).

Money for Nothing => Early Termination Clause with a penalty of 20% of the remaining contract value to be paid by the customer.

Change for Free => Changes in the requirements requested by the customer are totally free

-Joe's Bucket: Plan a two extra time slots after the end of the project (e.g. 2 months) that can be used as 'overtime'. The first slot is for the Customer and the second slot is for the supplier.

-Contract execution = everybody on same Agile level: ?

-Big Flexible Supplier: ?

-Percentage UserPoints Backlog:?

Things to consider in Contracting IT projects:

-One Team:

-Initial Product Backlog:

-Contribution/Effort of the customer to the project:

-Distribute:

-Scope must be variable:

-Rolling window of trust:

-Based on what do you start the collaboration:

-transfer IP vs. iterations:

Appendix C: Every single Code

adVANTAGE	'adVANTAGE underspending' not impressive
5	Overspending more frequent
	adVANTAGE: good when supp lacks trust
	adVANTAGE actually just T&M per sprint
	adVANTAGE: hour-focus can break teams

Agile Collaboration Agreement	Monitor to improve collaboration
17	Ideal: if low trust - steer the developers
	Less specification but more monitoring
	How to state customer involvemt rules in contract
	More freedom allowed when proven their ability
	Customer's freedom possible to state formally
	Solution: Collaborative Contract
	Ideal: devs freedom in Agile but Cust may Frequent
	Ideal: allow more freedom if perform good enough
	Ideal C: customer very frequent checks on progress
	Customer should take action if project rusty
	Free scope change: Collaboration Agreement
	Cust should have freedom to join sessions
	Collaborative contract: goal specification
	Ideal: Collaboration Agreement instead of features
	Ideal: Agree that customer can join all meetings

Agile Contract Solution	Warranty in Agile?
21	Smaller work items are more easily explained
	AC is shared risk
	Ideal: Given Sprintlength + Budget + Burnrate
	Ideal: No upfront specify price or duration
	Ideal: Room for exception clausules
	Ideally no price negotiations - only results
	IT ppl not very good at financial
	Joint-venture: guarantee mutual success incentives
	Largely a matter of time before AC
	What to do with the iterative aspect?

		Two-phase ctr second phase arbitrary pricing model
		Two-phase contracts to manage trust issues
		People simply don't know another solution
		Monitor to improve collaboration
		Long intentions with clear early term agreements
		No upfront agreement on Early Termination
		jurist challenges: goal in contract very difficult
		Liability transfer in Agile?
		T&M Money 4 nothing 90-10
		Payment best be closely after the demo
Benefits of Demos	Agile good for projects with visible results	
5	Visibility of results important to customer	
	2-weekly Demos create trust	
	2-weekly Demos create team-feeling	
	Demos improve trust	
Budget	PPS: Budget depleted -> look for sponsors	
5	PPS: Budget depleted -> maintenance mode	
	PPS: Limited Budget -> prioritize reqs	
	PPS: Limited Budget -> Sponsors	
	Budget constrained projects	
Contextual Influences	Subcontracting Fixed Price	
13	Agile in-house big org for self-improvement	
	Agile in-house easier than b2b	
	Agile in-house rising	
	Fixed Price Distributed	
	[P3's organization's name] high hourly costs	
	Intercompany Agile is challenging	
	More outsourcing -> lower hourly costs	
	Subcontracting Per Hour	
	Subcontracting Per Hour Positivity	
	Subcontracting: managing freelancers	
	Scrum with 40+ developers is difficult	
	Real distributed teams in Offshore projects	

Contract Incompleteness	contract completeness	
1		
Customer Involvement	Customer should attend at least 1x/week	
15		How to state customer involvemt rules in contract
		Product owner on customer's site
		Product owner often on customer's site
		Product owner less present on-site
		Some devs resistant against outsiders joining sesh
		Preferably more than once a week
		Other than demos and planning -feeling of distrust
		Infrequent visits from customer
		Contact with customer: telephone and e-mail
		Co-located developers -> cust sees effort
		At start of project should attend more often
		Feedback by shortly testing
		Monitor to improve collaboration
		Customer involvement important
Desire for Certainty	Customer is not-knowing and cautious	
7		Comforting the Customer w/ FP
		Challenge: comforting the customer
		Ag vs Trad: Customer wants upfront specification
		Problem: being stuck to specs in the contract
		customer wants security
		Negativity Upfront complete specification
Early Termination	Early termination crucial	
11		Early termination agreements crucial
		Early termination only negotiated when necessary
		Early termination Funny
		Early termination not seen being used yet
		Early termination: suck cost
		De-Scoping rare/difficult for customer
		Money for nothing unknown but appreciated
		Money for.. often explained rarely applied

		Early termination
		What if supplier terminates early
Estimating	Experience based estimating and Delphi	
10	Planning poker	
	planning poker: dominant people	
	planning poker good to incorporate everyone	
	Experience based estimating	
	Estimating with Points	
	Estimating easier in known situations	
	planning poker in grooming sessions	
	planning poker: crowd sourcing	
	High executives not very involved in planning	
Fixed Price	Fixed Price estimation	
36	FP counteracts Agile working	
	Fixed price uncertainty of delivering right system	
	Fixed Price Risk Neutralising	
	Fixed Price Project Process	
	Fixed Price Government	
	Fixed Price Distributed	
	Fixed Price Deliverables	
	Fixed Price Contingency Planning	
	Difficulty Fixed Price	
	High pressure big projects: apply daily standups	
	Danger of 'Just in Time'-nature of Trad	
	Smaller work items are more easily explained	
	Fixed price clear to client	
	Addendum	
	Smaller work items	
	Request for Proposal Government	
	RE solution: use examples	
	RE solution: Translators	
	RE solution: prototypes	
	Problem: being stuck to specs in the contract	
	IT ppl not very good at financial	
	Challenge: specification	

FP projects often failed to satisfy the goals
Long projects divided in multiple contracts
FP over deadline: seperate contract
Initially little progress visible in Trad proj
Comparison with Traditional 'Milestone' Bills
Waterscrumming
Traditional reqs rarely SMART
Trad RFP huge estimateproblem if reqs amiguous
Negativity Upfront complete specification
Negativity Fixed Price
Mostly Fixed Price demands -> SMART?
realize that waterfall in-house is problem
Combo of waterfall and scrum

Flexibility in Relationship Apply Scrum ideology to Customer Relation
5 Flexibility to prevent legal problems
Flexibilty PPS helps bridge initial trust barriere
PPS: Flexibility Positive
Flexible contract

Government Waterfall Government regulations Waterfall-like
5 Government no-bonus regulation
Government Budget contrained
Fixed Price Government
Request for Proposal Government

Intellectual Property Intellectual property transfer in Agile?
2 Challenge: Intellectual Property

Jurists jurist challenges: goal in contract very difficult
5 Different mindset jurists
Challenge: jurist challenges
Companies can hire SE jurist to make contract
Jurist challenges

Less Emphasis on Contract	Legacy syst replacement: easy project goal	
10	PPS: No agreement on end-result+price	
	Decrease focus on Contract	
	Don't pressure on amount of points	
	Ideal: Focus on succesfully reaching sprintgoals	
	Ideal: Focus on feature/stories not price/duration	
	More Trust needed in Less Specification	
	Payment structure is not key	
	Succeeded in keeping specs very abstract	
	Instead focus on succesful project and collaborat	
Money for Nothing	Bad connotation word 'Bonus'	
4	Money for nothing unknown but appreciated	
	Money for.. often explained rarely applied	
	Value of Bonus-system hard to measure	
Other Factors in Agile	Customer always wants more for less	
4	Traditional supps qualityproblem employees	
	Agile has high demands for developers	
	Agile good for projects with visible results	
Paradigm Transition Phase	Waterscrumming	
8	devs do scrum - the rest still used to waterfall	
	We are now in Transition phase	
	Transition phase from fixed price to AC	
	Transition phase of waterfall to agile thoughts	
	Paradigm shift FP -> T&M -> now AC	
	Paradigm shift: FP to AC	
	percieved pressure on productivity	
Perceived Duration of Relationship	Misperception: Immediately order entire system	
4	Advocate of long term intentions	
	Creating intention of Longterm relation	
	T&M: short term vision -> cust uncertainty	

PPS PPS resembles T&M

52 PPS Per Team: Clarity with Shared Stakeholders

PPS: More in smaller projects

PPS on small projects up to e.g. 1million

PPS: customer pays bills with less resistment

Payment related to Verification Popular

PPS the way to go

PPS: Budget depleted -> look for sponsors

PPS: Budget depleted -> maintenance mode

PPS: Contract states process

PPS: Flexibility Positive

PPS: pretty Agile compared to others

PPS: Limited Budget -> Sponsors

Small company: No need for AC

PPS: No agreement on end-result+price

PPS: Risk increases if customer stalls payment

Starting phase lower performance

PPS: system-maintance service not PPS

PPS: Warning system

PPS: Limited Budget -> prioritize reqs

Fast billing preferred due to Global Economics

Clear fixed price per sprint

Buy extra Sprint Insightful

Bill ASAP

1 Sprint contract

PPS behind schedule

PPS makes sense for customers

Contract variables

Contract states certain amount of sprints

Doesn't understand Y PPS not used evrywhere

PPS: Small projects more predefined

Fixed PPS: don't mind under-or overspending

Flexibilty PPS helps bridge initial trust barriere

Halting project when bills are piled up

He is helped by KPMG in financial aspects

Never seen PPS

Payment best be closely after the demo

Payment Per Sprint (PPS)
Positivity PPC
PPS Contract: preliminary funct spec of 1st items
PPS creates technical debt
PPS Demands financial discipline
PPS is bad
PPS is do-able but real AC should have [list]
PPS is not very rare
PPS Fixed Price Per Sprint
Negativity PPS: moment of verification
PPS makes sense and will grow
Ideal: Charge monthly at Fixed burnrate
Ideal: Fixed PPS
Agile difficulty with Traditional

R&D Contracts	Research Collaboration Agreement
3	Services agreement
	confidentiality agreement

Scope Change	Ideal: Define Common goal instead of features
9	Ideal: Free Scope Change
	Ideal: Collaboration Agreement instead of features
	GOALS are hard to state formally
	Free scope change: Collaboration Agreement
	Define common goal: state goal very concretely
	Change for free creates bargaining about size
	Collaborative contract: goal specification
	Common goal: joint-venture

Scrum Practices	Stand-ups over Skype
10	sprints retrospectives and demos
	Scrum most frequently used
	Offshoring: daily video-communication crucial
	Velocity
	Internal communicator system
	Law of 30 feet

Always Daily Standups even with distributed teams
1 hour planning session instead of 1 day
Grooming

Short Sprints Ideal: 2 weekly sprints
3 4 week sprint is used
2 week sprints

Size Expected Negativity Agile Contracting
17 Agile in-house big org for self-improvement
Big project - Integrated Agile
Big projects less trust
Standard implementation processes
PPS: More in smaller projects
Agile frequent in larger organizations
Contract signer not involved in project
Big projects harder to do Agile
Instant deploymt of increment -> Small customers
Very long projects not suitable for Fixed
PaymentStructure not very important Small
Scrum with 40+ developers is difficult
sprint teams are relatively small
Small company: No need for AC
PPS: Small projects more predefined
PPS on small projects up to e.g. 1million

Specification vs Expectation Better to verify system against the goals
8 Cust's goals often different than supp's features
Importance of Customers' Expectations
Specifications should be cust's underlying GOALS
System specification VERSUS Expectations!
Traditional satisfy the reqs but not expectations
Agile can prevent Spec vs Expectat conflicts
Agile Contr prevent Spec vs Expectat conflicts

Time & Materials The Contract follows the Agreements

14	T&M benefit: allows to really work Agile
	T&M Contract content: amount of feature points
	T&M cust can stop when he wants
	T&M Disadv: Cust falsely thinks he has freedom
	T&M Freedom but actually still dependent on each
	T&M Money 4 nothing 90-10
	T&M: dependent on eachother
	T&M basically good for all projects
	Continuous negotation while 30 are developing
	Maintenance focused contract T&M
	Overall burndown track-keeping for billing
	No strict hour control by cust
	T&M: short term vision -> cust uncertainty

Trust Expectations initiate Trust

15	Big projects less trust
	Flexibilty PPS helps bridge initial trust barriere
	Demos improve trust
	Less specification but more monitoring
	Less trust Needed in day-to-day
	Trust important
	Trust issues with customer
	Trust should come from both sides
	More Trust needed in Less Specification
	Misperception of customers
	2-weekly Demos create trust
	Misperception of software suppliers
	Not enough mutual Trust for big Scrum
	Customer lacks Trust in Valuable Endresult

Two-phase contract Two-phase contracts to manage trust issues

6	Two-phase ctr second phase arbitrary pricing model