Finding a customer specific model driven development methodology
The CAPE Groep Case

Pieter van de Braak
26-7-2010

COLOPHON

Title: Finding a customer specific model driven development methodology

Date: 26-7-2010

Author: P.N. van de Braak

Under supervision of:

CAPE Groep B.V.
Kosteeweg 13
7447 AJ Hellendoorn
Phone: Tel. +31(0)54-8656065
Fax: +31(0)548-656512

Universiteit Twente
Business Information Technology
P.O. box 217
7500 AE Enschede
Phone: +31(0)53-4894995
Fax: +31(0)53-4892159

Graduation committee:

M. Iacob (University of Twente)
M. van Sinderen (University of Twente)
P. Verkroost (CAPE Groep)
R. ter Brugge (CAPE Groep

CONTENTS

1. INTRODUCTION

The goal of this first chapter is to create a clear picture of the context in which this research has been conducted and how it came to be. Paragraph 1.1 shortly describes the background of the problem which is then followed up by paragraph 1.2 that describes the context of the research then paragraph 1.3 will describe the reason for this research and 1.4 gives an overview of how this report is structured.

1.1 Background

Software development has been a field of discussion ever since the introduction of it. There have been many different methodologies and silver bullets that claimed to reduce risk, cost and time of the projects. For some this was true however, as technology is constantly growing, new methodologies are required to fully reap the benefits of these new technologies.

One of the new technologies that has been introduced in the past 10 years is model driven engineering. Model driven engineering has had a rough start, but the last few years it has seen large growth. Model driven engineering is the development of software based on the use of models. Spurred by the business side who require better fitting and new software, at a faster pace, programmers use models that are more easily understood by the business than programming code. Model driven engineering has proven that it can provide software much faster than a classical coding project. Practice shows that model driven engineering can not only provide software on a quicker pace but that it also allows for easy feedback to the clients making it more efficient. This however implies that it requires a different approach as the majority of the effort now is placed in the designing (modelling) of an application rather than coding it. How can this new technique be best supported by a development methodology and what influences this?

Software development methodologies can help to structure the development, making it more efficient. But customers can be a nasty barrier that has to be overtaken. Customers sometimes choose the opposite way of what is recommended and thus it requires adaptation from the developer to facilitate a successful project.

In this thesis we conduct research to finding and aligning a software development methodology with the customer that can utilize the advantages of model driven engineering.

1.2 Research context

1.2.1 CAPE Groep

This research takes place at CAPE Groep in Enschede, the Netherlands. CAPE Groep is an advisory company which operates in the domains of construction, transportation and logistics and naval construction. CAPE Groep consists of two divisions: CAPE Consulting and CAPE System Integrations.

CAPE Consulting mainly focuses its effort on advising companies during the selection of new software but it also supports companies during the implementation of software, does process optimizations and deals with project management. They position themselves between software suppliers and the customer and support the implementation from start till end.

CAPE System Integrations provides integration services ranging from board computers to ERP systems. They provide integration of existing systems with new or with old software. This is done on basis of off-the-shelve before custom made. However if no fitting solution is available on the market they also play the role of application supplier and provide their own solution. Since 2007 CAPE Groep started to build their software solutions based on the Mendix platform. Mendix is a software platform which is provided by the equally named company.

1.2.2 Mendix

Mendix was founded in 2005 and started as a spin-off from the Technical University of Delft and the Erasmus University of Rotterdam. The name Mendix comes from the verb ‘to mend’: “To repair, to

restore, to improve and to correct". Their goal is to improve business applications by making it much easier to measure, extend, integrate, build and adapt business processes.

Mendix provides a model-driven enterprise application platform that enables business analysts to build service-oriented business applications that can be integrated and adapted in many existing IT & business environment. Key benefits are increased flexibility, accelerated application delivery and reduced complexity. Mendix positions itself as a provider of a software factory. They do undertake software projects of their own but they try to keep this to a minimum.[1]

1.2.3 Market

As said earlier CAPE Groep operates in three different markets: The construction, transportation and logistics, and naval construction market. However, for this research we focus on their main market which is transportation and logistics. About 80% of all projects of CAPE Groep have their roots in the transportation and logistics market. Figure 1 shows the positions of CAPE Groep and Mendix, in the transportation and logistics market.

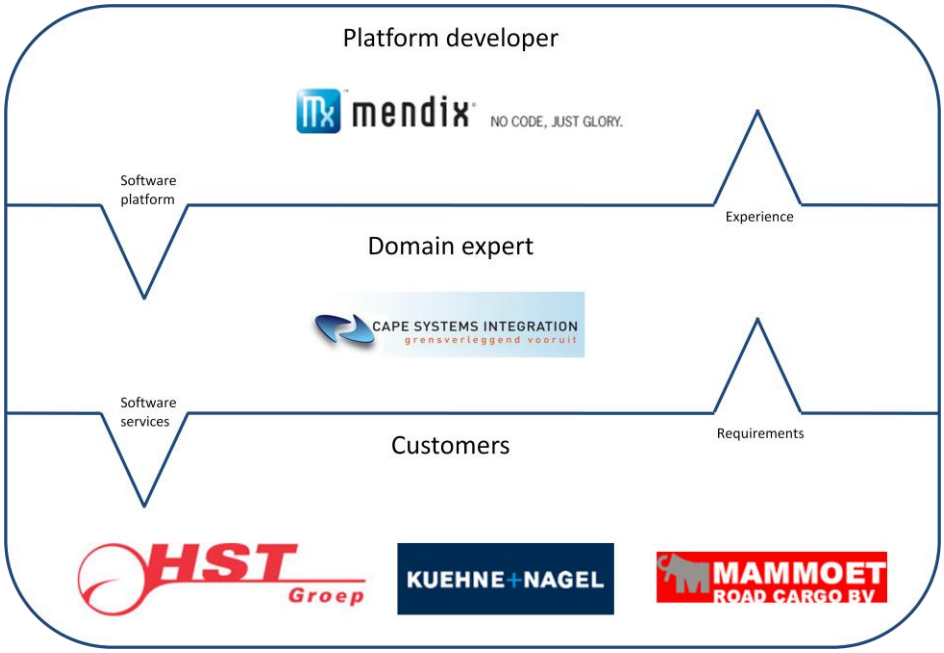


Figure 1: Market situation

1.3 Reason for research

A research thesis is always structured around a main problem. A problem is defined as a gap between experience and desire of a stakeholder. This means that a problem can be seen as a difference between what is desired and what is reality [2]

In 2007 CAPE Groep started with developing their own software solutions. Two example projects that will be used in this thesis are the order entry portal of HST and the Board computer portal at Mammoet Road cargo. Both projects were model driven development projects in which CAPE applied the Mendix platform and in both projects CAPE Groep experienced some lag and delay because of discordance with the customer on the form of contract. Furthermore, CAPE Groep noticed that applying the same development method with different customers lead to varying results. These varying results lead to the demand for a development methodology that could be adapted to the customer. CAPE Groep believes that if they align their development methodology better with the customers their projects will require less effort to produce the same outcome.

CAPE Groep is interested in how their development methodology affects the development time among different customers and how different types of contract affect development. So how does the

communication, the development method and the contract between the developer and customer influence development? And how can this be improved?

Among other things this research makes use of the practical experience derived from already finished projects at CAPE Groep. In these projects we search for occurrences where influence of the customer is exerted on CAPE Groep and the development process. However, not only the occurrence itself is important but also the causes behind it are of paramount importance, because knowing why and how a customer reacts to a certain situation allows for timely interaction and can reduce the risk of annoyance both at the side of the customer and the project developer. After all, an ounce of prevention is worth a pound of cure.

This research is conducted in the context of model driven development.

1.4 Structure of report

This report is structured around six chapters. This chapter provides a small summary of the problems that are present and tries to explain the reason for this research. Chapter two describes the structure behind this research by identifying the process that is used during the research and by clarifying the research questions. In the following chapter the theory and background information, that was required to perform this research, is fully discussed. Among others are the topics: Software development, Model driven development, Development methods, Customers in software development and Contracts. All of these topics are combined in chapter four to create the framework. This framework is validated in chapter five, which discusses the framework among several experts from CAPE Groep. Finally chapter six concludes this research with conclusions and recommendations for future research.

2. RESEARCH DESIGN

This chapter will further elaborate on the research question that is answered within this research. Paragraph 2.1 will introduce the project charter that is the charter in which this thesis will conduct its research. Then paragraph 2.2 will discuss the problem at hand, followed up by the project goal and scope. Then as last the research question will be summed up and a structure of this paper is made to show what subjects are covered in which chapter.

2.1 Project charter

According to the CHAOS research from Standish Group there are many reasons as to why software projects fail or exceed their budget, the most important being lack of executive support. However, in their top 10 they also include the formal methodology on the 8th place. They state that using a formal methodology can increase the chance of success by 16%. This shows that, although not the most important aspect, having a formal development methodology significantly increases the chance of bringing a project to a successful end[3].

As said in the introduction, a good methodology helps structuring the overall process of the project and ensures that fewer mistakes are made. Mistakes such as bad testing of software or miscommunication due to bad specification. To reduce these mistakes it is important for a software developer to make a good forecast of the situation to ensure that proper and timely intervention can be realized. This research will specifically handle the development of software using model driven technologies and look at the influence of the customer on this process.

Model driven technologies can be more flexible than classical development technologies or pure code generating technologies. The reason that they are more flexible is that they can produce results faster and enable easier and quicker communication between customer and developer.

During this research, knowledge of managing and developing with model driven technologies that is obtained by CAPE Groep in previous projects is used. Within these projects a distinction will be made between the different project characteristics customer type, contract type and development method.

Possible connections that are found between the development methodologies and the project characteristics can help with choosing a good fitting development methodology in future projects. Thus, the research provides insight in the backgrounds of common problems and properties of model driven software development and will prescribe a project methodology for a given customer. The research can be seen as a diagnostic research.

2.2 Problem statement

Based on the project charter the following problem statement has been made:

'How can model driven software projects become more efficient by aligning development with the customer and contract type'

2.3 Research goal

The goal of the project is to improve the efficiency of model driven development projects by aligning the customer with the contract, and the development method.

2.4 Research approach

The main research question will be validated and answering following a few steps. First a literature study will be conducted concerning model driven development. This will provide the research with its scope by defining the outer bounds.

Second the literature study will provide the definition of a software development methodology and we will identify what different development methods are applicable for model driven development.

From here on literature on customers in model driven software projects will provide us with the insights required to map the customers with the development method.
Finally a study in contracts for software projects is done so that these can be mapped between the development methods and customer types.

Combined, the above studies will provide the information needed to create a framework that allows mapping of the customers to the contract and the development method. This framework will be validated by experts from CAPE Groep.

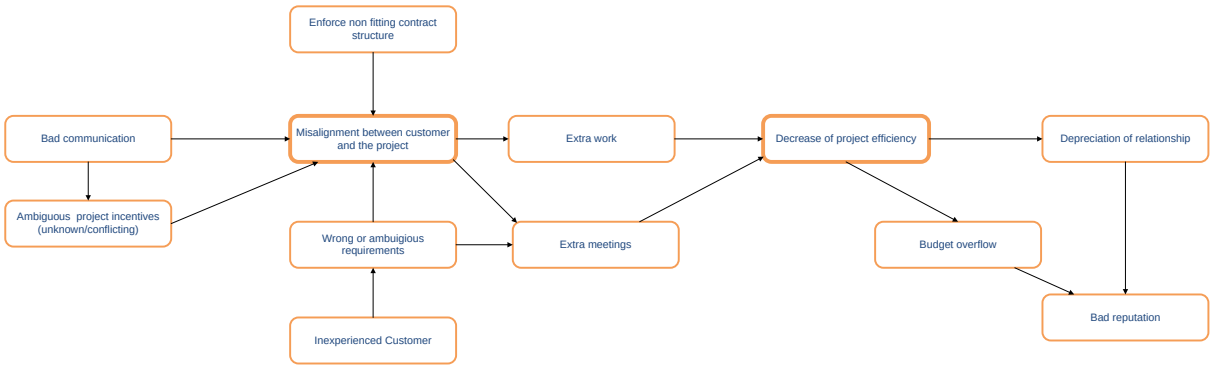


Figure 2: Initial problem identification

2.5 Research scope

In order to keep the project manageable within the predefined time limitations it is important to define what falls within the research but also what falls outside of the scope. We define the outer limits as follows.

This research only focuses on software development that makes use of model driven development. Although software development is discussed it is only touched to show how model driven development evolved.

The project will only focus on problems that occur with the developing of software. Although this might sometimes overlap with project management it will not focus on general project management.

2.6 Research questions

1. What types of contracts are used in the software engineering field?
2. How does the customer influence software development projects?
3. How can a development method of MDD be aligned with the customer?
4. How can the efficiency of software projects be measured?

2.7 Structure of research

Figure 3 shows an overall structure that is required for this research to produce its outcomes according to [4].

The subjects which are contained within the dotted line are the parts that require theoretical research. The other parts are the practical parts considering the CAPE Groep Case.

2.8 Research method

This paragraph describes which methods are used to obtain the information required to answer the research questions.

Table 1: Research methods

Research question	Approach
What types of contracts are used in the software engineering field?	Literature study
How does the customer influence software development projects?	Literature study, MDD conference, Mendix essentials, unstructured interviews
How can a development method of MDD be aligned with the customer?	Literature study
How can the efficiency of software projects be measured?	Literature study, unstructured interviews

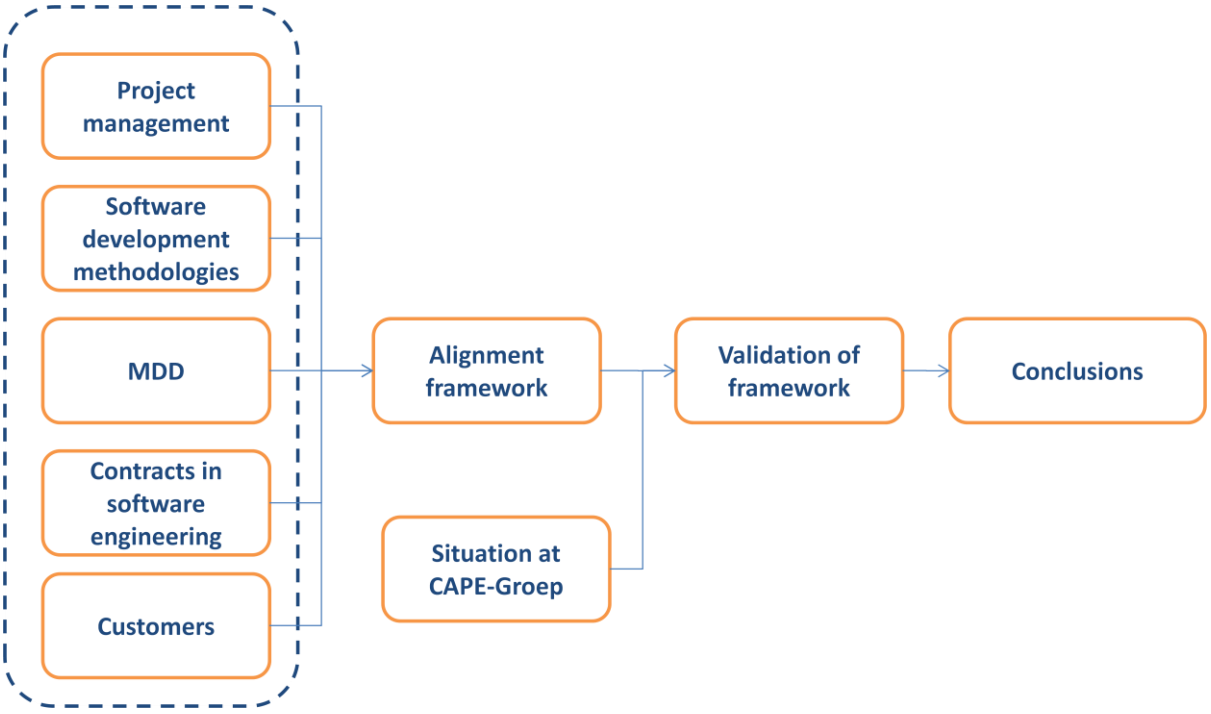


Figure 3: Structure of research

3. STATE OF THE ART

This chapter will try to give more insight in the terminology of software engineering and model driven development. It will then continue to explore the used contracts in software engineering followed up by the influence of customer on a software engineering. These insights form the foundation that is necessary to understand and support this research. The most important concepts that will be thoroughly investigated are model driven development, development methods, customer characteristics and contract types.

3.1 Software engineering

This research aims to increase the efficiency in model driven software development projects. Before we start, let us first take a look at the definition of software engineering.

According to [5] s*oftware development is the set of activities that results in software products. Software development may include research, new development, modification, reuse, re-engineering, maintenance, or any other activities that result in software products.*

Software development thus not only consists of coding, but also of the planning and managing of processes that result into software products. When compared against different definitions of software engineering we find that engineering and development is actually the same thing.

The first definition of software engineering dates back from 1968, given at the first NATO conference [6]:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

Some other definitions respectively given by [7-9] are:

Software engineering refers to the disciplined application of engineering, scientific, and mathematical principles and methods to the economical production of quality software.

(1)The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).

Software engineering is the establishment and use of sound engineering principles and good management practice, and the evolution of applicable tools and methods, and their use as appropriate, in order to obtain – within known but adequate resources limitations – software that is of high quality in an explicitly define sense.

These definitions use rather different words to describe the field. However, the essentials are the same. Software engineering is a practice that makes use of formal and disciplined methods to research, create, operate and maintain software. Software engineering can thus be seen as a very broad field. This is why it is so important that development methods are applied to structure its content.

3.1.1 Origin of software engineering

As stated earlier software engineering originates from 1968 when at a NATO conference the term was introduced to gain access to engineering funding. However, software was already being made long before this conference.

In the early fifties applications were created by hardware providers. The programs were relatively small and often written by just one person. The problems that had to be solved were mostly of technical nature. If a program contained any errors, the programmer studied a dump of memory and then tried to fix the error in the output. Sometimes, the execution of the program would be followed by binary reading machine registers at the console.[10]

In the 1960's people started to realize that programming, which was what software engineering was all about at that time, had become too complex. Programs were becoming very large and it was increasingly difficult to debug applications. This is why new programming languages were introduced that made the code more readable, by introducing a higher abstraction level, so that errors were more easily detected and software was created faster. At the same time, more and more programmers were involved in software projects.

Software engineering reached a point at which computers were rapidly evolving but software development wasn't. This lead to the software crisis as it was named by Dijkstra in 1972. To head this crisis again new programming languages were introduced such as FORTRAN, COBOL and ALGOL, which tried to reduce the complexity of software projects by adding extra structure to the programming languages. But also programming methods were introduced to structure the development process and the management of resources. In 1970 Royce talks about a structured and sequential approach that introduces the phases of requirements elicitation and application design. This model would later come to be known as the waterfall model. Software development methods will be further discussed in paragraph 3.4 [11]. With the introduction of the personal computer, the field of information technology became available to the large public making it grow faster than before.

With the growing availability not only large companies were able to attract IT but also smaller companies started to use IT. With a rapidly expanding market IT was once again growing abundantly and with this abundant growth came again more complexity. A great example is the crash of the long-distance network of AT&T in the United States, which paralyzed some of the world's key financial and business institutions. The costs for this crash were estimated in the hundreds of millions of dollars. Yet the root cause was tracked to a C program that was missing a "break" statement – a kind of coding error which is not uncommon at all and very difficult to detect once a program surpasses a million lines of code [12].
With the growth of IT, applications became larger and larger making it harder to check for these type of errors. In the book The mythical man-month, Fred Brooks identifies two types of complexity. Essential complexity: complexity which is inseparable from the problem. A famous example of this in logistics is the travelling salesman problem. The other type of complexity is accidental complexity, which is the complexity that is a direct consequence of the resources and methods that are used to approach a problem. For example, construct a house without the use of bricks and cement or to build a bridge while only using reed.

Accidental complexity is one of the reasons why models were introduced in the world of software engineering.

3.2 Models in software engineering

Models are a very broad concept and they have been used in many engineering fields before being applied to the software engineering field. Examples that are known to most are blueprints of buildings or charts of landscapes. Over the passing ages these models have been used and have been perfected to become a reliable tool in construction and engineering. With the introduction and adaptation to model driven engineering, models start to get a more prominent role in software engineering and so now their development has started to shift as well. To understand how these models have developed and what they mean we start by looking at them from a more general perspective. After all, when applying a model driven development approach, models are used in much the same way as in other engineering disciplines [13].

A general definition of a model is given by Starfield, a model is a representation of a concept. The representation is modelled with a purpose. The model purpose is used to abstract the irrelevant detailsfrom reality [14].

This implies that models are used to model or portray only part of reality to reduce the complexity. This is supported by Seidewitz who states that a model is an abstraction (also called representation or denotation) of an object system (also called system under study) expressed in some language. Where an interpretation of a model gives the meaning of the model relative to the object system [13].

Models thus not only provide the ability to abstract a certain piece of a system under study. But they can also be interpreted to determine or predict some values of the system under study. It should be noted that the system under study can also be another model.

Eventhough there are many different definitions of a model all sources agree that models are used to abstract and simplify reality. Models provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on the relevant ones.
Hughes created a graphical picture to represent this process shown in Figure 4.

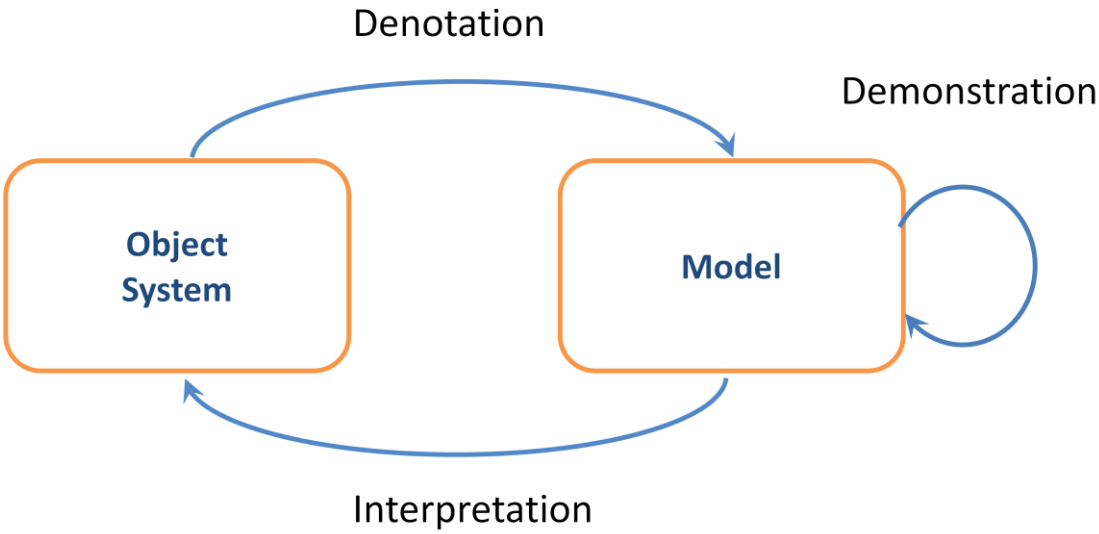


Figure 4: Model denotation demonstration and interpretation

In the figure the object system is the reality that is being denoted by a model. This model can be used to calculate or to demonstrate values so that it can be updated. At any given time the model can be interpreted to update the reality that is under observation. For a more precise description of how modelling works we refer to [13].

Following the above conclusions, models are mostly used as a description of a system under study. However, in software engineering models have so far been used as a specification rather than description. For example a weather forecast that provides the current wind speeds at a given location is a descriptive model. While a model, based on the weather forecast and used to calculate the optimal route for an airplane, is a specification model. This is also true in software engineering; a class diagram specifies what the software should look like. It is more often a design rather than description. Because it requires a lot of effort to keep both the models as the code up to date, the models are often used to create a general architecture but later discarded and not used anymore.

Within the class diagram example, the implementation of the class diagram depends on the interpretation of the programmer of the model. With interpretation we mean the mapping of the model's elements to the elements of the system under study such that we can determine the truth value of statements in the model from the system under study [13]. In other words, the interpretation of the model gives the model meaning in relation to the system under study.

If this relation is invertible then a representation of the system under study can be made, in which all statements about the system under study are true under the current mapping. These mappings form a model of the model-to-model relations and are called model transformations in model driven development.

3.2.1 Model transformations

The mapping of elements between two different models is called a model transformation. Model transformations are rules that transform models or transform data from one model to another. It is also possible that a model transformation transforms a model to executable code. [15] Schmidt discusses three types of model transformations that are used in model driven development; refactoring transformations, model-to-model transformations and model-to-code transformations.

Refactoring transformations reorganize a model based on predefined criteria. In this case the output is a revision of the original model, called the refactored model. An example could be as simple as adding prefixes to each element name in a model. These transformation are also called endogenous transformations or rephrasing transformations [16].

Model-to-model transformations convert information from one model or models to another model or set of models, typically where the flow of information is across abstraction boundaries. An example would be the conversion of a java model to an XML model. These transformation are also called exogenous transformations or translation transformations [16].

Model-to-code transformations are a form of model-to-model transformations, with the distinction that the code that is generated is not seen as model. This of course depends on the definition of a model. These transformations convert a model element into a code fragment. Model-to-code transformations can be developed for nearly any form of programming language or declarative specification. An example of model-to-code transformations would be the generation of Data Definition Language (DDL) code from a logical data model expressed as a UML class diagram [[17]].

Another distinction that can be made between the different transformations is horizontal versus vertical transformations. A horizontal transformation is a transformation where the source and target models reside at the same abstraction level. Typical examples are refactoring (an endogenous transformation) and migration (an exogenous transformation). A vertical transformation is a transformation where the source and target models reside at different abstraction levels. A typical example is the refinement of a platform independent model, e.g. a UML class diagram, to a platform specific model, e.g. java classes [18].

Summarizing, model transformations allow models to be supplemented with additional information or allow the translation of one type of model into another type of model. Model transformations can be executed manually but also automatically. These automated transformations are the strength behind model driven development or model driven engineering.

3.3 Model driven development

So far we have only described models and model transformations. Although these are the basis of model driven development it is still unclear what model driven development is and why it has such a potential.

Model driven development uses models as a basis rather than code. Even though code is also a model at some level, code is often not understandable for a lot of business people. Because of this, there exists a gap between the business analysts and the developers in traditional development. Model driven development can fill this gap so that communication between business analysts and developers runs smoother.

3.3.1 History of model driven development

Model driven development or model driven engineering is an approach to application development that uses the concepts of models and transformation to create an application based on an abstract representation.

In current software development projects tools are used to automate model transformations but this wasn't always the case as the concept of models is not new to the field of software engineering.

Ever since the beginning of software engineering models have been used to visualize the code and to help to communicate about the code. There are tools to help clarify functionality and to provide a better overview to the programmers. Examples of these type of models were use case scenario's and use case diagrams. Later on with the introduction of object oriented languages came models such as the class diagram and entity relationship diagrams. However because these models were not formally linked to the code they were often incomplete or inaccurate [12].
Due to lack of this official link it became too much of a burden to keep the models up to date during development. This led to incompatible models and is the main reason that software models are still mostly used as a design artefact during early development of the application [12, 19-20].

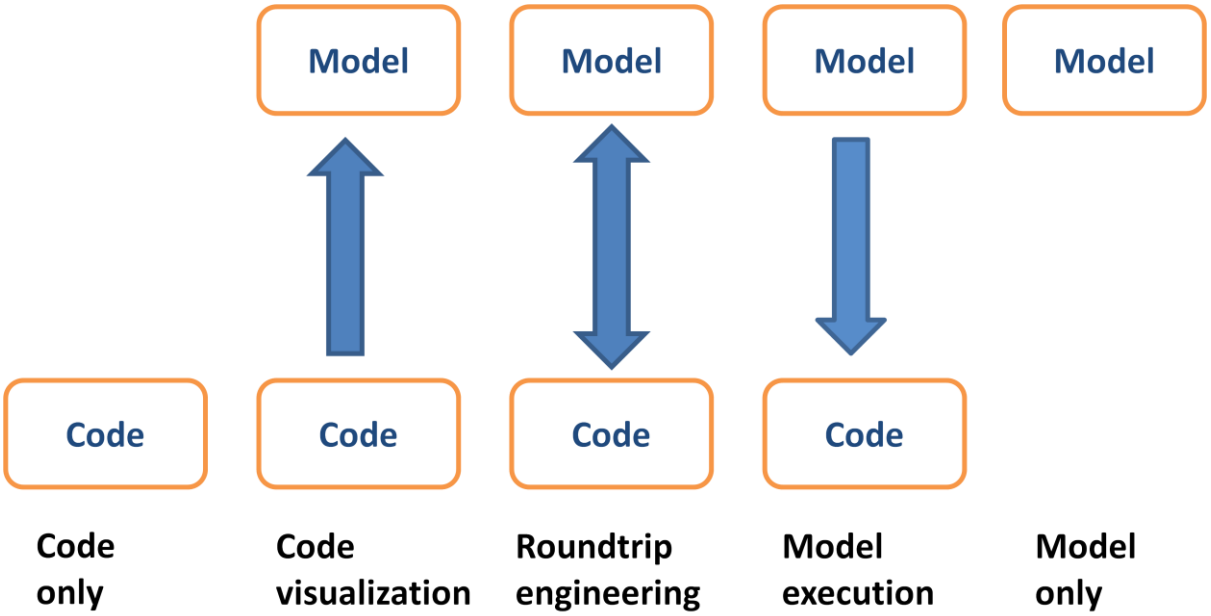


Figure 5: From model based to model driven [21] [22]

Figure 5 shows the progress of the usage of models over time, ranging from none at all, to model based development, towards full model driven development by use of model execution.

Currently most developers either use only code or sometimes hand sketched models that have no formal link to the code. They rely solely on their code to model their application in the form of packages and classes. When displaying their application to the business they tend to form a graphical model, often done in UML, which is often translated by hand. While this method has proven itself over the past decade it can be difficult to understand key characteristics of the system as each person has a different interpretation of non-formal models.[21]

A more formal approach is when developers use the approach of code visualization. As developers create or analyze an application they often want to visualize the code through some graphical notation that aids their understanding of the code's structure or behaviour. With code visualization a visual application generates the graphical model based on the code. It may even be possible to manipulate the graphical model as an alternative to editing the text-based code, so that the visual rendering becomes a direct representation of the code. Some tools that are able to create these visual representations or "diagrams" are for example IBM WebSphere Studio and Borland Together. The later functionality is also called roundtrip engineering. [22]

In this thesis we are especially interested in the newest developments in model driven engineering called model execution. Rather than in the other situations the model instead of the code is now the basis of the application. From the model, code is generated that can be executed directly, by run time environments. This has a great advantage over the more classical forms of software engineering because now it becomes possible for the business side to model their requirements in a formal way and store their knowledge in the model. In a way they are programming the application, in some cases it might even be possible for the business to program an application themselves. This increases the overall development speed as less interaction between the actual programmers or technical side and the functional side is required. [15]

A different view on the development of model driven development is presented by Hailpern & Tarr. They create an analogy between Julius Caesars observation of the Gaul and the types of model driven developers. They identify three types of model driven developers the sketchers, the blueprinters and the model programmers.[20]

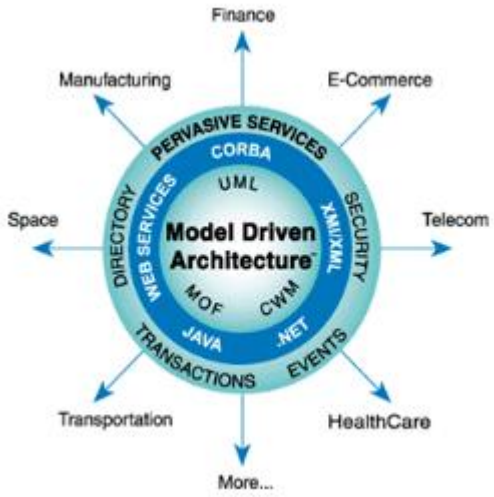
The sketchers focus on the use of modelling languages to increase and facilitate the understanding of the code. They do not talk about the code they are going to work on, they just explain the most important issues by use of models and code the rest themselves.

Blueprinters are experienced developers that draw out very detailed application designs and create an analogy between the architecture and application design. They then leave the implementation of the design up to less experienced designers.

Model programmers also support the use of modelling languages but with executable semantics. Model programmers can be best identified with roundtrip engineering and model execution. They make use of executable code either in form of a high-level programming language or by direct execution of the model. The model-programming camp is typified by the supporters of the object management group (OMG) vision of a standard called model driven architecture (MDA)[20].

3.3.2 Model driven architecture

Simplistically models are nothing more than an abstract view on something. But with this definition programming languages such as java and FORTRAN are also a model. While this is true, model driven development has a different view on models. The Model Driven Architecture guide from the Object Management Group provided a new definition for a model. The MDA guide states that a model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language [18].



MDA is a standard for model driven development that was introduced by the object management group (OMG) in 2000.

It is based around three goals; portability, interoperability and reusability. To achieve these goals they make a clear distinction between three types of viewpoints: The platform specific viewpoint, platform independent viewpoint and computation independent viewpoint. These viewpoints provide an approach that separates the system from the platform on which it runs. Each of these viewpoints can contain different models that can be linked to each other with help of transformations.

The computation independent viewpoint focuses on the environment of the system, and the requirements for the system; Models created in the computation independent viewpoint are called computation independent models or sometimes domain models. These models are made in a vocabulary that is understandable to the domain experts. This plays an important role in dealing with the knowledge gap between the domain experts and the design experts.

A computation independent model (CIM) is a view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification. It is assumed that the primary user of the CIM, the domain practitioner, is not knowledgeable about the models or artefacts used to realize the functionality for which the requirements are articulated in the CIM. The CIM plays an important role in bridging the gap between those that are experts about the domain and its requirements on the one hand, and those that are

experts of design and construction of the artefacts that together satisfy the domain requirements on the other hand.

The platform independent viewpoint focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent model shows that part of the complete specification that does not change from one platform to another. Models from this viewpoint form the bridge between the business side and the technical side. Where domain experts are able to create models for the computation independent models, technicians and implementation experts create the platform independent models. The latter are then transformed to the platform specific models by use of transformation rules. The platform independent models thus play the role of translator between the implementation and the requirements specification.

A platform independent view may use a general purpose modelling language, or a language specific to the area in which the system will be used. A common technique in creating a platform independent model (PIM) is to use a technology-neutral virtual machine e.g. the java virtual machine. A virtual machine is a set of parts and services (communications, scheduling, naming, etc.), which are defined independently of any specific platform and which are realized in platform-specific ways on different platforms.[18] In the java example, the virtual machine translates the java code to a byte code which can be read by almost any platform. A virtual machine can be seen as a platform that is running on top of a platform. Logically a java model is then specific to the java platform. However because the virtual machine itself is platform independent so is the java code i.e. A→B & B→C then A→C. [18]

The platform specific viewpoint combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system. For example the transformation of values stored in a model to a MYSQL database would require a different platform specific model then a transformation to a PostgreSQL database [20]. The platform specific model (PSM) thus store information how the application is stored on a targeted platform. The PSM is the most static model as it will only change when the platform changes.

In basis the separation of these viewpoints ensure that certain pieces of code are not required to be rewritten every time a new model is created. This is because the transformations are static and stay the same, unless the modelling language is unable to model something i.e. if the modeller is unable to model its needs in the model.

There are commercial tools that fully automate the transformation between the so called platform specific models and the platform independent models. They provide the user with the ability to create several CIM's and PIM's in a domain understandable environment which then translates these models to run time code via platform independent and platform specific models. Examples of these tools are Mendix and OLIVANOVA [1, 23]

3.4 Software development methods

Software development is a complex human activity which like many others has to be managed in order for it to go smoothly[24]. This is one of the reasons why software development methods were introduced in the early seventies.

Software development methods are methods that structure software projects in the sense that they provide planning, role distribution and often a format for work products. Because software projects are so diverse and are also applied in many different fields many different software development methods have been developed over the years. Examples are the waterfall method, scrum, crystal and rapid application development. Because software development methods have been developed in many different fields there is no one clear definition of what a software development method entails.

Software methods were first introduced in the seventies to provide more structure to the large software projects of that time. The waterfall method, introduced by Royce, is one of the most renowned[11]. It prescribes several stages that are common for most software projects. Because it was the first development method that was recognized most developing methodologies show the same stages. These stages are system requirements, software requirements, analysis, program design, coding, testing and operations. Even though Royce mentions the possibility of feedback from one step to another, the purpose of the waterfall model was to capture the concept of the application up front and to try to create the application based on this specification.

The waterfall method and evolutions of this method are often referred to as heavyweight, plan-driven or full life cycle methods. With this we mean, methods which try to specify the whole application up front [25]. Many researchers believe that these heavy weight methods are too inflexible to cope with current business environments and point out that this is the main reason why so many projects fail. They have adopted their view to newly introduced ideas about software engineering [26-27]. This group is focusing its attention to making software engineering more agile. With an increasing group of proponents and researchers, agile methods prove to be a promising technique.

3.4.1 Agile development methods

Agile development methods have coexisted next to classical development methods ever since the mid 80's. Qumer and Henderson offer the following definition for the agility of any entity:

"Agility is a persistent behaviour or ability of a sensitive entity that exhibits flexibility to accommodate expected or unexpected changes rapidly, follows the shortest time span, uses economical, simple and quality instruments in a dynamic environment and applies updated prior knowledge and experience to learn from the internal and external environment."

Agility thus provides the ability to adapt to unknown changes which is exactly what was missing in the heavyweight methods. However, agile software development is more. In 2001 a small group of software practitioners wrote a manifesto on agile software development [26]. This manifesto consists of twelve principles that form the basis of agile software development and can be found in Appendix A: Software manifesto principles.

Taken together with the notion of agility and of development method, an agile software development method can be defined as [25] :

"*A software development method is said to be an agile software development method when a method is people focused, communications-oriented, flexible, (ready to adapt to expected or unexpected change at any time), speedy (encourages rapid and iterative development of the product in small releases), lean (focuses on shortening timeframe and cost and on improved quality), responsive (reacts appropriately to expected and unexpected changes), and learning (focuses on improvement during and after product development)"*.

The main values of agile development are summarized by Warsta as individuals and close interactions, customer collaboration, iterative development and response to change [28].

3.4.2 Agile and MDD

Model driven development provides the user with an easy way to abstract and visualize an application. By introducing the three view separation of CIM, PIM and PSM by OMG, it becomes easier to cross the gap between the IT

3.5 Contracts

This research aims to increase efficiency of a software development projects by aligning the development method of the software with the preferences and properties of the customer. One of the most, important components that is used during the development of not just software but in any development project is the contract.

A contract can be seen as a specific job or work order, often temporary or of fixed duration and usually governed by a written agreement [29]. In software engineering contracts are usually set up in the acquisition phase of the project. They are used to setup a project organization [30]. They are required to reduce the risk of possible friction between the developer and the client(s).

Another reason why contracts are used is to spread the risk between the involved parties in a project. For example when two projects with exactly the same content are done at two different companies A and B. Then the project might be more expensive at A than at B because of a different project setting between the two companies.

While no contract is the same, Whang concludes that the core of a contract consists of; product definition, intellectual property protection and payment structure [30].

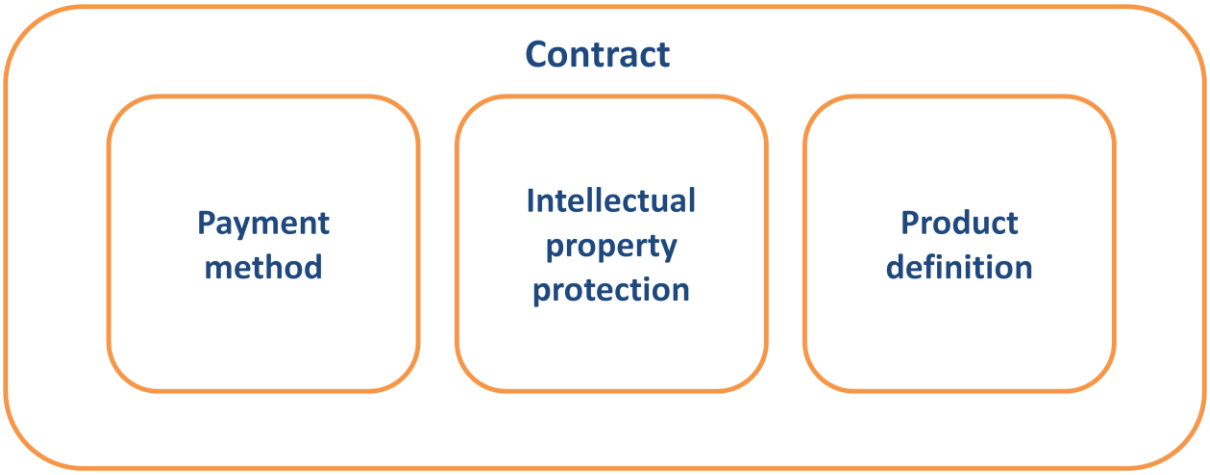


Figure 6: abstract contract

The product definition includes the description of what is supposed to be delivered at what point during the project. Software contracts specify the product in great detail. The product definition specifies the requirements for the software but often also hardware and documentation is mentioned. The software specification mainly consists of lengthy functional specifications that define the tasks and input/output of the application. Also it often specifies boundaries like the operating system, database environments or programming language. Very often a preliminary study has already been done so that a technical design document and a full requirements document are also attached. If this hasn't been done the contract will also have a list of services that the developer should provide during the development like, requirements analysis, interviews, design , programming, testing, implementation, support, training etc.

As software is not a physical entity that can be exchanged between two persons there always exists a potential danger that disputes will arise over its ownership. Since the same solution (with minor tweaks) could be used for a different or even competing company. Another example of a possible danger is a trade secret that might need to be revealed to the developer who also has contact with direct competitors of the client. To avoid these kinds of disputes contracts also often specify what content can be used by whom and who owns this knowledge. This part of the contract is called the intellectual property protection.

The payment structure describes how payments should be done and when they should be made. Most contracts try to specify an amount that the software is delivered in and couple this amount to a time to be delivered for a given cost. Some examples of this itemization are software licenses or service level agreements. Payment schedules are usually tied to the development phase i.e. after the completion of a certain phase a payment has to be made for that part of the project.

These three contract components together form the contract which can also be seen as a scale of risk that determines who carries what risk. Often uncertainty is covered by a fee of money, for example, if the developer does not deliver he will have to pay a fee. Or if the customer provides a very uncertain business case it could result into a wrong delivery which as Whang states could lead to legal actions from the customer. Hence that the contract should be structured in such a way that it tries to align the incentives of the customer with those of the developer.

Often contracts are categorized by their payment method. Turner identifies five different type of contracts by their payment structure [31].

- Cost plus (Time & materials)
- Re-measurement based on a schedule of rates
- Re-measurement based on a bill of quantities
- Re-measurement based on a bill of materials

- Fixed price

However, they state that the project management body of knowledge, an ANSI norm for project management, only recognizes one type of re-measurement contract which is the re-measurement based on a schedule of rates. Which is why they only validated their theory with 3 contract types. Both of these categorizations are found to be too specific, as the difference between these contracts are not likely to influence the customer behaviour enough to realign the development method. We thus require a different categorization. Before we further abstract the types of contracts we will first elaborate and support our hypothesis above by explaining what these contracts entail with an example of a car garage.

With a cost plus contract, the garage is paid for all incurred costs plus an extra fee to ensure a profit. The agreed profit margin can be a percentage of the total costs or a fixed margin. In this example it would mean the garage owner is paid for all hours his repair crew made plus all incurred costs for materials like paint, glass or bolts and screws.

The re-measurement contract based on a schedule of rates is a contract that, as with the cost plus contract, calculates the cost afterwards but does this so on an average cost per rate. This means that hours worked by the repair crew are measured and paid for by a predefined agreed price per, hour per x amount of screws or bolts.

Re-measurement based on a bill of quantities does not differ much from the re-measurement based on a schedule of rates. However instead of calculating the price for all hours and materials, an average price is taken for the size of an activity. For example if a car needs a painting job, the price could also be calculated by an average price per square meter, no matter what paint is used. So the price is paid in square meters rather than in litres of paint.

The re-measurement based on a bill of materials is the most abstract of all three re-measurement contracts. It calculates an average price for a whole activity. For example if you take away your car to a garage, you can choose to let it have a standard check-up and pay an average price for this check-up based on the type of car and the size of the car.

The fixed price contract is probably the oldest contract form that is used in software engineering. It calculates a price upfront for which a definite set of requirements has to be delivered. Thus if you would like to have a dragon painted on your car it calculates upfront the price of the estimated amount of paint, the estimated amount of working hours and the estimated amount of time the painting area is in use. From this a standard price is calculated, all hours, materials and time that is spent over budget are costs and risk for the garage owner.

3.5.1 Applying contracts

From the perspective of the developer it seems that a time and materials contract would always be the best choice as it will cover any cost of efficiency decrease. If the garage owner would take twice as long to paint the car of his client it would still be paid for.

However Turner and Simister provide us with a different perspective on this as they explain using the transaction cost concept. They define the transaction cost as the cost of planning, adapting and monitoring task completion. In short the costs of managing the contractual relationship [31].

In a perfect world each contract would provide the same out-turn costs however, as Turner explains due to opportunism by the developer and human errors from the client this is not the case.

Thus Turner states that the total cost for the customer equals the out-turn cost plus the transaction costs and that the transaction costs will differ between the different contract types for a certain product.

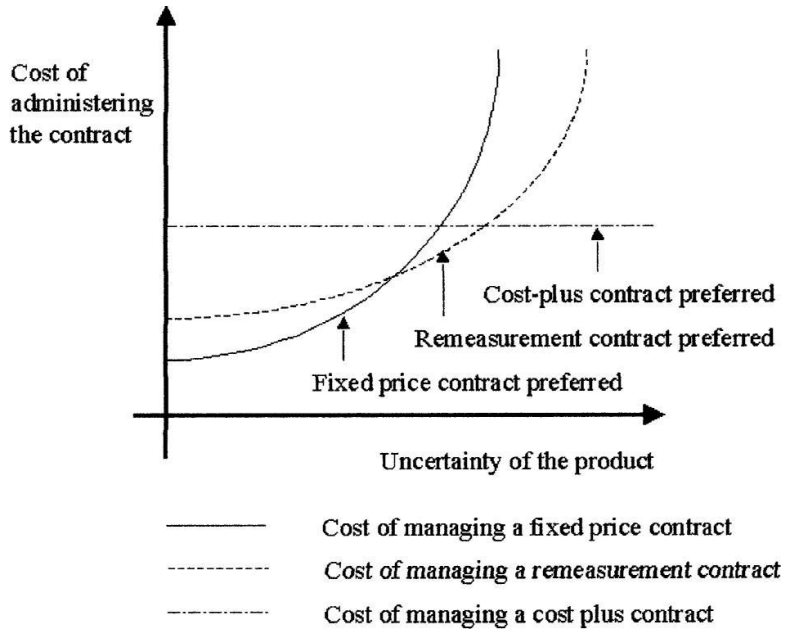


Figure 7: Transaction cost versus uncertainty of product[31]

This idea provides a way to choose a type of contract, namely the cheapest type of contract for the uncertainty of the product. However there is a flaw in this theory, it assumes that transaction costs and out-turn costs are not associated and we believe that this wrong. Different types of contract may incentivize developers differently. For example, if the garage owner had all the time in the world to paint the dragon he will most likely put more effort into beautifying the picture. Thus providing a better result.

This is why Turner extends his theory with the concept of goal alignment, stating that: “The most significant issue to consider when choosing a governance structure for the contract is the need to achieve goal alignment between the client and contractor. And to reduce the chance and benefit for opportunism by the client or contractor.” [31] To support this claim he proposes a framework that is shown in Figure 8.

Uncertainty in the process stands for uncertainty about how to solve the problem, e.g., should paint be used or can it be done with stickers. Uncertainty of the product is often seen as the responsibility of the developer.

Uncertainty in the process stands for uncertainty about what the problem is and how it can be solved, e.g., what will the dragon look like, how large should the dragon be. Uncertainty of the process is often seen as the responsibility of the customer.

There are 3 points to note:

- Figure 8 is based on the goals and methods matrix from Turner. [32]
- Turner states that it’s not fair for the contractor to bear the risk of an uncertain process when applying a fixed price contract in an uncertain process. While indeed this could lead to loss of profit, the developer could also gain extra profits from this as he could find a more innovative solution for the process. As turner formulates it “The contract does not need to be fair, it just needs to be clear”. In other words as long as the developer is aware that the situation is unclear he can choose for a fixed price contract if he thinks the solution is easy.
- The last note is that the fourth quadrant of high product uncertainty and low process uncertainty is not researched. This quadrant will be further explored as software contracts often fall in this quadrant.

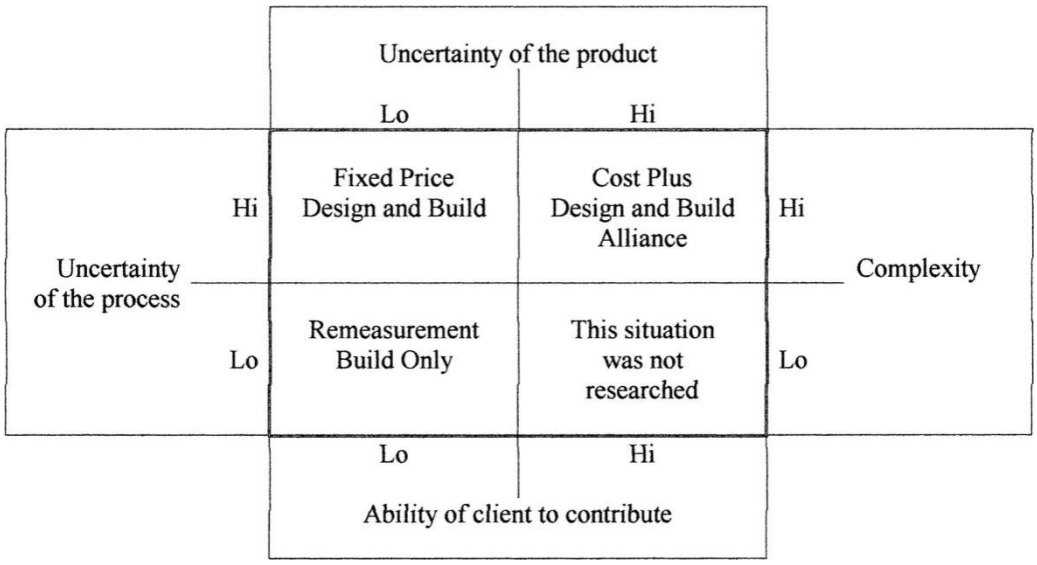


Figure 8: Selection of contract types [31]

3.5.2 Project triangle

The Contract can also be seen as a way to govern the project storing important decisions about the 4 quadrants present in the project triangle presented in Figure 9.

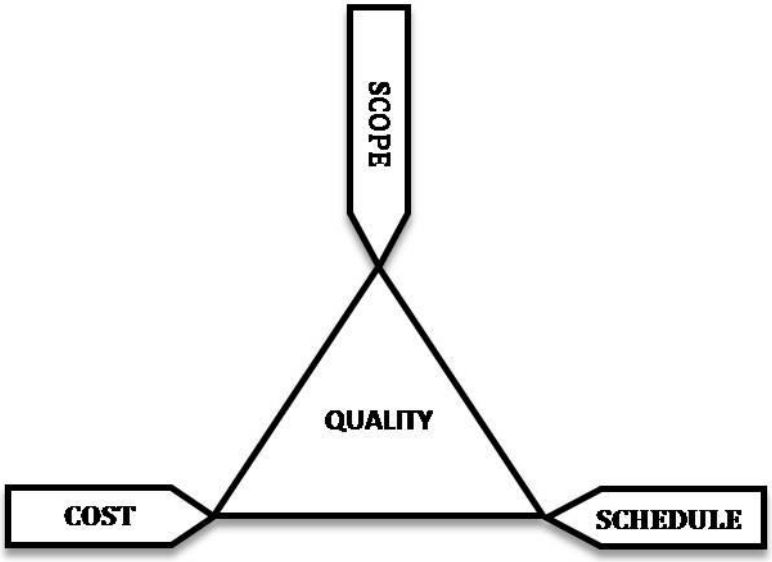


Figure 9: Project triangle

The project triangle was originally introduced as a project management tool covering the three dimensions of project management; Scope, Cost and Schedule. These three dimensions together ensure a certain quality to be delivered. The quality is depicted by the surface of the triangle. Thus if a contract specifies a certain quality of a project than if one of the dimensions changes during the project for example scope, one of the others will have to change as well. In practice a rule of thumb prescribes that the customer is allowed to fix two dimensions so that the developer can determine the third.

3.6 Risks and opportunities

In this thesis research is conducted towards formulating a development approach and a form of contract based on a type of customer or previous experiences with a customer. As explained in the previous paragraph a contract is an agreement that is used to establish mechanisms that help to

avoid and cope with future conflicts. Conflicts are often the result of a risk that is present in a project. This is why in this paragraph the concept of risk is further explained.

3.6.1 What is risk?

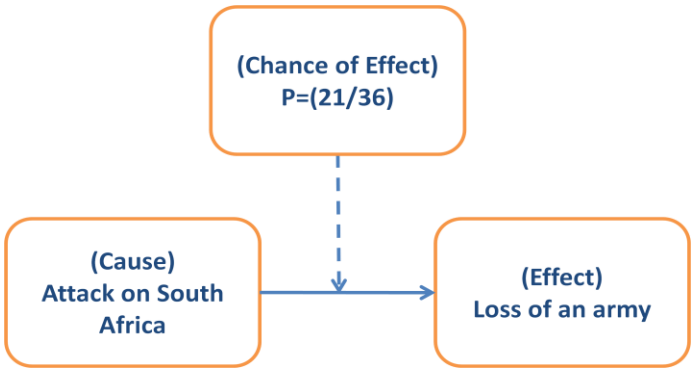
Everyone is familiar with the famous board game called risk, but what does it mean? What is risk and how is it involved in software development.

The term risk is commonly known as an undertaking that is dangerous and could result into harm. The formal definition provided by the Webster dictionary is “expose to a chance of damage or injury”. Boehm translates this definition into a concept of risk management, namely risk exposure, also called risk impact or risk factor [33]. He continues by extracting the following risk exposure formula:

Risk exposure = chance on impact * impact on stakeholder

To translate this to our game board, what is the risk of losing an attack from South Africa to Madagascar? First the chance on loss has to be calculated, e.g. Madagascar has two armies and is allowed to attack with one and South Africa has one defending army. This leads to a chance of loss of p= (21/36) =59%. Considering that if Madagascar loses he will lose 50% of his armies the risk exposure is quite large, and could be even bigger if Central Africa is also threatening Madagascar.

While real time issues are often not as easily specified as our example above, it is often possible to provide some estimation on the chance that a risk occurs and on the cost that the impact of the risk has. From the given example we can also conclude that a risk has a certain cause, an effect and a chance to occur. If the risk example was played by a computer who randomly decides on his moves the example would look like the following.



To further clarify how this translates to the world of software engineering some examples are provided.

What is the risk that the business owner of the customer will (effect) leave the project during its execution because of (cause) illness?

What is the risk that the project will be (effect) delayed when the project owner (cause) leaves the project team?

3.6.2 Uncertainties and risks

The source of any risk lies in the uncertainties that are present during a software project. Unlike many people think, an uncertainty is not actually the same thing as a risk. A risk is a possible outcome of an uncertain situation. For example, the economic stability of a company influences the continuation of large projects. Whenever a crisis occurs projects can get postponed or cancelled. But, when the company flourishes it might increase its budget for the project and expand the scope leading to an opportunity. Uncertainties thus provide two outcomes an opportunity and a risk. To further clarify the difference between risk and opportunities we provide you the image below.

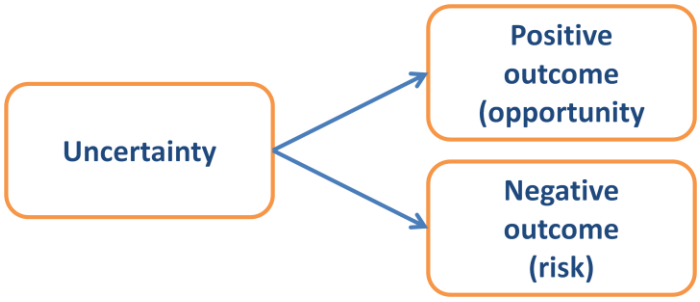


Figure 10: uncertainty causes opportunities and risks

In construction projects we also find a clear distinction between risk and opportunities. An example of an opportunity is: A project experiences delay due to an archaeological find. If during this period of delay the house prices are rising more than the rate of interest they will increase in value which increases the value of the project.

However these situations are not very predictable and cannot or should not be taken into account. Rather than counting on opportunities it is better to try to minimize the risks. In classical software engineering projects this is done in a similar way. To reduce the risks of software engineering large plans and descriptions are made and hence the name of plan driven software development. Reducing risks can also lead to an improved chance of the risk turning into an opportunity.

3.6.3 Risk properties

Risks can be categorized based on their own specific properties. These properties are important when trying to manage these risks.

Occurrence

This indicates in what period the risk could possibly occur and thus when special attention or counter measures should be taken. Sometimes these measurements have to be taken upfront in order to counter the risk. Furthermore, there are different ways that a risk can occur. For example the risk can be seasonable, meaning that it only occurs during a certain season and can thus be taken into account. The other extreme is that the risk is instant, which shows no warning upfront and so no counter measures can be taken.

Dependency
Events that carry certain risks with them are not always independent, they can be mutually dependant. In total there are three types of dependencies:

- Independent risks
- Mutually exclusive risks (either one of the risks can happen but never together at the same time)
- Dependant risks
    - Additional risks can occur as a result of implemented risk management measures.
    - Residual risks can occur after the implementation of risk management measures when the risk cannot be fully countered. To get a clear picture of what risks are dependant a cause & effect diagram can be applied.

Impressionableness
This is the measure of influence that the project manager can bring to bear to the source of the risk based on the four risk management strategies: avoid, transfer, reduce and accept. These strategies will be further discussed in paragraph 3.6.8. The size of the risk determines the necessity to risk management while impressionableness determines the possibility to management.

Chance
The likeliness of the risk occurring is quantified by its chance. As shown in paragraph 3.6.1the term chance can point to both the likeliness of the cause of the risk to take place as to the likeliness of the

effects of the risk. To quantify a chance Boehm suggests a three-way distribution. The table below shows this categorization of Boehm.[33]

Qualitative	Quantitative	Chance
Frequent	P ≥ 0,7	85%
Probable	0,3 ≤ P < 0,7	50%
Improbable	P < 0,3	15%

Table 2: Example table of qualitative risks versus quantitative

Impact
Where the property chance describes the likeliness of the risk, the impact pictures the extent of the consequences of the risk in the project control concepts: time, money, scope and quality. The terms impact and chance form the basics of risk management as they determine the scope of what to manage. Risks can be influenced by chancing their chance or their impact.

3.6.4 Risk management

Risk management is part of project management methods and aims at controlling risks that could possibly occur during the project. There are many different risk management strategies that can occur at different strategic levels. In this research risk management is used to determine how the developed method should be adapted. As said in paragraph 3.4 a development method can be seen as a form of risk management as it is often used out of fear to prevent certain risks.
Risk management is not a new concept, many software developers like Boehm and Talbot have dedicated their research to software risks and their management. However, still software projects fail as new techniques such as model driven development are introduced. Furthermore, risks are often not addressed appropriately.

3.6.5 Risk management process

The process of risk management is a process that never ends during a project. Because a project is continuously exposed to new risks, a project manager should always be alert.[34] The goal of risk management is first that it needs to identify and analyze risks up front. This is also the most important part in our thesis. A second part which plays a lesser role is that a good risk management strategy also continues to monitor and control the risks. While risks in general will reduce over time, new risks might show up which if left uncontrolled could destroy a project. According to Boehm Risk management is build out of 2 phases; risk assessment and risk control. Each of these phases has 3 sub phases. The whole process can be seen in Figure 11.

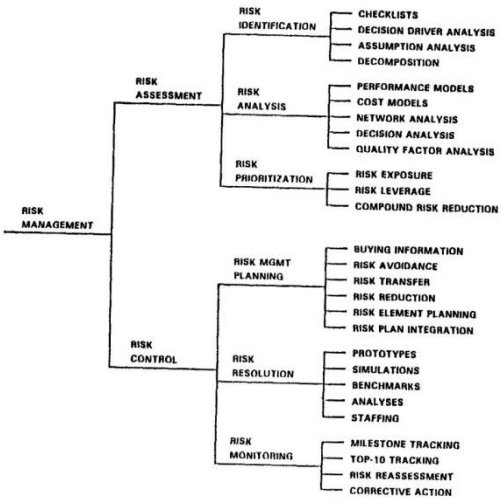


Figure 11: Risk management phases[34]

3.6.6 Risk identification

As said before the first task of risk management is to identify the risks that are present during a project. This is also emphasized in literature by Well-Stam, who applies the RISMAN method [35].

Risk identification is the mapping of possible risks that may occur within a project. Normally the goal of this process would be to determine all the risks that could occur during a project. There are several risk management methods that also incorporate a risk identification phase. It is not uncommon that risk identification happens multiple times during a project. Some risk identification techniques are:

- Check lists
- Decision driver analysis
- Assumption analysis
- Problem decomposition
- Interviews
- Swat analysis

Each of these methods can reveal risks and can thus be used to identify risks that are present in a project. However, the risk identification process as a whole is not being questioned in this research. This research focuses more on finding a development method, thus finding a way how to adept to known risks rather than identifying these risks. As turner states it, it is important to focus on the top 20% of the risks that together form 80% of the impact.

3.6.7 Risk prioritization

Another important step for our research is risk prioritization. Risk prioritization appoints a value to a given risk so that each risk can be measured relatively to all other risks. Not all risks have an equal chance or an equal impact. There are many statistic models that can be applied to calculate risks however these are left out of scope. Rather we research models that make use of relative scales.

3.6.7.1 Risk exposure

The first model is used by Boehm and uses the two variables risk exposure and risk reduction leverage. The risk exposure, explained in paragraph 3.6.1, stands for the probability that a certain impact takes place. Risk reduction leverage is a function that is defined by Risk exposure before – Risk exposure after / Risk reduction costs. With the risk reduction leverage the usefulness of a certain risk management solution can be calculated.

However this approach does have its difficulties. One difficulty is the problem of making accurate estimates of the probability and the loss or impact of a risk. Checklists that make use of a qualitative measure such as shown in Table 2 provide some help in assessing the probability, but it is clear that the probability ranges in four categories are far from accurate.[34] As said earlier statistical models might provide an outcome here, however even Boehm deems these models to difficult and states that it might just as well be simpler to use a simpler course in which an estimation between 1 and 10 is given to risk chance and impact.

3.6.7.2 100 dollar bill

Another good model that listens to a relative measure and has had many applications in science is the 100 dollar bill method. With this method the stakeholders divide a hundred points over the set of risks. The more points a risk gets the more important that risk is. If the amount of objects to be evaluated surpasses 20 then the amount of points can be increased to 1000 or 10000, so the points can be distributed more fair. However, increasing the amount of points should be done with care. Too many points will cause the stakeholder to be less accurate.[36]

3.6.8 Risk management strategies

When risks have been identified and prioritized, they can be managed by use of the four risk management strategies. These strategies are Accept, Share, Reduce and Avoid and are discussed below.

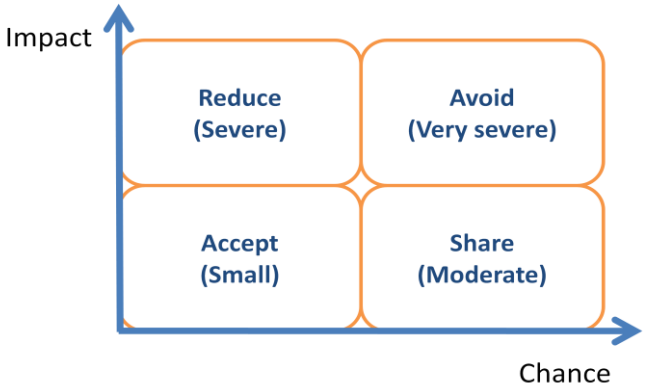


Figure 12: Risk management strategies

Avoid
Avoidance of risks is aimed at eliminating the uncertainty that is driving the risk. In order to avoid the uncertainty it first has to be known what is causing this uncertainty. The removal of a cause or breaking of the cause and effect relation takes away the possibility of the uncertainty of taking effect and thus the risk as a whole.
An indirect avoiding approach entails that the project is directed along a different route, in which possible impact of the uncertainty does not matter anymore. For example if a project is planned in a year that a company has bad results it could be postponed to a later year when the company has better results. This would avoid the risk of getting a low budget on the development project. Avoidance is however not always possible as it can lead that main targets are not reached or that new even bigger risks occur. Risk avoidance always happens before the risk has actually occurred.[35]

Share
The risk strategy of sharing is aimed at sharing or shifting the ownership of the risk to the customer or to a third party. This strategy can be used for financial risks such as planning exceeding or budget exceeding. It is important to realize that carrying a risk does have its financial benefits. The party that carries the risk is likely to be paid a fee in the contract or is compensated in another way. It should be noted that if risks are shared, all sharing parties should have a risk procedure in case the risk occurs. Otherwise the risk might ruin the project.
Examples of risk sharing are the risks of additional travelling costs that are paid by the customer. Or the risk of when a party exceeds a delivery deadline because of external influences.
Risk sharing is usually part of a fixed price contract as in a time & materials contract almost all risks are carried by the customer.

Reduce
In practice most risks cannot be avoided or shared with other parties this is why reducing risk is the most applied risk management strategy. It is a strategy that is aimed to contain the impact or the chance of a risk so that its effects are reduced. The development method that is introduced in our framework in paragraph 4.1 is mostly based on the reduction of development risks by adapting the development method. Reduction can take place both before the risk takes place as after it has occurred.

Accept
After all risks have been covered there are some risks left that are either too improbable or just have no impact. These risks are to be accepted, no adaption or change is done. However, these risks should still be contained by identifying them early to ensure that they do not get out of control in case of occurring. If the impact of a risk might become larger than initially thought additional measure are required. When accepting risks it is important to have a risk aware culture and a risk management program at hand. [35]

3.7 Project characteristics

Every software project is unique in a way that it has a different combination of characteristics like size, customer experience, requirements ambiguity. The variation of these characteristics is what this research is about, in this paragraph we will explore the characteristics of the customer that might influence the development process or the form of contract.

3.7.1 CRM concepts

As with any engineering product the customers form the backbone of a software development project. They provide the developer with requirements on the to be developed system and often more. However, what fits best to a certain customer is not always clear. How can one recognize a customer from another and which properties and wishes from a customer have affect on the development of software and the contract. To achieve this some concepts of customer relationship management are applied. Galgreath en Rogers defined a customer relationship leadership model in which they show how management can facilitate the implementation of customer relationship management (CRM) concepts. They define CRM as follows:

"*Activities a business performs to identify, qualify, acquire, develop and retain increasingly loyal and profitable customers by delivering the right product or service, to the right customer, through the right channel, at the right time and the right cost. CRM integrates sales, marketing, service, enterprise resource planning and supply-chain management functions through business process automation, technology solutions, and information resources to maximize each customer contact. CRM facilitates relationships among enterprises, their customers, business partners, suppliers, and employees.*"[37]

In this context we are especially interested in the identification of a customer which can later be used to qualify them for a certain form of contract and development method. To identify the customer a customer profile can be used. A customer profile is a set of data describing specific customer characteristics.[38] Hence that it is possible to have multiple profiles on different data of the same customer. To find a fitting contract and MDD project method a customer profile can be used to identify and then qualify the customer.

3.7.2 Characteristics of customers in Software engineering projects

To identify a customer a set of characteristics is needed on which the customer can be identified. To determine what characteristics are important we need to take a step back and first look at what is important for a software project.

To define what is important for a software project we take a look at the literature of success criteria. Critical success factors for a software project provide the most important aspects that are required for a successful project. Literature formulates multiple factors that can influence the success of a project.

The Standish Group published a report in 1994 called the CHAOS report that among other things identifies the 10 main success factors in software engineering. This report was updated in 2000 and contained the following factors: Executive support, user involvement, experienced project manager, clear business objectives, minimized scope, standard software infrastructure, firm basic requirements, formal methodology, reliable estimates, other [3].

A comparable research was conducted by Tsun Chow and Dac-Boo who analyzed critical success factors in agile projects [39]. Because model driven development seems much better fitted with agile development methods then with plan driven life cycles, we have chosen to apply the list of Chow.

They identified five categories of success factors namely organizational, people, process, technical and project. Because this research is focused on the customer, only the categories people and organizational will be used. Table 3 shows all failure and success factors in both categories. A full list of all success factors can be found in appendix C.

Category	Failure factors
Organizational	
	Lack of executive sponsorship
	Lack of management commitment
	Organizational culture too traditional
	Organizational culture too political
	Organizational size too large

	Lack of agile logistical arrangements
People	
	Lack of necessary skill-set
	Lack of project management competence
	Lack of team work
	Resistance from groups or individuals
	Bad customer relationship
Category	**Success factors**
Organizational	Strong executive support
	Committed management team
	Cooperative organizational culture instead of hierarchal
	Oral culture placing high value on face-to-face communication
	Organizations where agile methodology is universally accepted
	Collocation of the whole team
	Facility with proper agile-style work environment
	Reward system appropriate for agile
People	
	Team members with high competence and expertise
	Team members with great motivation
	Managers knowledgeable in agile processes
	Managers who have light-touch or adaptive management style
	Coherent, self-organizing teamwork
	Good customer relationship

Table 3: Success and failure factors[39] [15]

The success factors that the customer introduces to a project can be used to identify a customer. When one of these factors is not present it introduces an uncertainty to the project. How to deal with this uncertainty will be discussed in chapter 3.6.4.

3.7.3 Other characteristics

We identified other characteristics that can be used to identify a customer in a software engineering project however, they are more abstract and hard to measure or they fall into a different area of research. Because of this they will only be mentioned for the sake of completeness.

3.7.3.1 Customer Incentives

Whenever a customer initiates a software project, he has a certain goal in mind. That is the client had his reason to pursue this project and to let the developer develop it. This could be simply because the developer is the cheapest or because it is a well known developer or because the developer develops in the neighbourhood. Whatever the reason to pursue the project is, It is not uncommon for these goals to be opposite between the developer and the customer e.g. the developer wants to earn as much money as possible where the customer wants to pay as little as possible. Whenever these incentives of both parties are not clearly stated it becomes impossible to reach the optimal agreement i.e. contract. We omitted these incentives from our research as we find them too hard to measure. It is near to impossible to know the true incentives of a customer for conducting a project under a certain condition. And even if it is known it is even harder to check whether or not the known incentive is true. In a small book called Getting to YES, Roger Fisher explains this phenomena and provides some tips on how to cope with aligning incentives during negotiations[40].

3.7.3.2 Customer perceptions

Other aspects of a customer that can influence the development of software are the psychological senses of an individual. Usually in any projects, software projects not excluded, a developer or builder deals with a contact person which has the following sense that he uses to interpret the progress of the project. These senses are the following:

- Visual memories (visualisation)
- Auditory memories (sound)
- Kinaesthetic memories (experiences)
- Olfactory memories (smells)
- Gustatory memories (taste)

Although most of these can affect the judgement of the contact person, we choose to ignore them as viable input parameters in our thesis. We have chosen to leave these out of scope as we find them too hard to measure and because they form more of a psychological profile of a person.

3.7.4 Characteristics of the customer in software contracts

Besides the influence of the customer on the development of software we also want to investigate how the customer influences the choice of contract during a software project and which contract fits best with a certain customer.
Gopal conducted research to the likeliness of a fixed price or time and materials contract in a offshore software project outsourcing [41]. Although software outsourcing does provide its own endeavours we do think that some of their conclusions are applicable in this research. They made an eleven fold of hypothesis of which the following were found true:

- Perceptions of higher client management information system (MIS) experience are associated with higher probabilities of a fixed price contract.
- Larger clients are associated with higher probabilities of a fixed price contract
- Perceptions of greater project importance to the client are associated with higher probabilities of a time and materials contract.

From these, three important client characteristics can be distilled that can be used in determining the type of contract. These are client experience, client size and clients interest in project.

3.7.5 Concluding

From the text above we can conclude the following. In software project there are multiple properties of a customer that show an influence on a software development contract and project. These factors can be used indirectly to determine what kind of customer a developer is dealing with. They are split into two different categories, organizational and people, which stand for the organization as a whole and the people that work for the organization, usually the contact persons or the persons that are part of the development team.

Because the failure and success factors are often opposite to each other they listen to one common property, this allows the two factors to be taken together. When adding the customer characteristics for a contract to the list we get Table 4.

Category	Properties
Organizational	
	Management support
	Organizational structure
	Organizational culture
	Organizational size
	Agile logistical arrangements
	Interest in project
People	
	People skill set
	Customer relationship
	Resistance from groups or individuals

Table 4: Customer properties

Management support

Management support is often trumped as one of the most important success factor in information system implementation. Even though it seems a very important point in implementation, literature is not univocal about its meaning and its influence on the implementation.

Upper management support is simply the support that the upper management shows for the implementation of a certain solution or application. It seems apparent that upper management is supporting a project, otherwise it probably would not have been initiated in the first place. This is usually the case in small companies where upper management is always involved with most projects or at least with the financial flows in the company. However, in medium to large companies this becomes less apparent. KPMG shows in a literature study that involvement of upper management together with a solid business case, is one of the foundations for good management support.[42]

Two important effects of management support on a project are the selection of a project team and on the financial support of a project. Because on a lower level the developer needs to confer often with his client, it can lead to failure if he is speaking with the wrong business owner. This in turn can lead to wrong requirements ultimately leading to a non fitting software solution.
Furthermore, the executives often provide the funds that are required for the project. If they do not support the project they might not provide all necessary funding for the developer to realise the project as customer might want it. Or upper management might force the developer into a fixed price contract, for the developer it is thus important to know whether or not they have management support at their client. The Chaos report also marked this as the most important success factor in software projects [3].

Organizational structure
The organizational structure of a company is a hierarchy that mainly consist of managerial entities or layers that are present in a company. It is often depicted as an organizational chart which can display many things ranging from managers to departments. In this research however we are only interested in the flatness of the structure. With the flatness we mean the amount of managers that are involved with the authorization of decision taking but are not directly involved in the project. We find that the organizational structure can either be flat or hierarchical. As model driven development is about having a lot of interaction, the development is slowed down by large hierarchical structures. When a large hierarchical structure is present it usually leads to bureaucratic decision making which can influence the speed of the development cycle.
For example, when a request for change (RFC) is ordered by the customer, the developer will reply with the cost of the change. If the customer has a very bureaucratic style of governance than the contact person will first have to query his superior which will have to query his superior, before it can be approved.

Organizational culture
Culture is a very broad concept and while it seems logical that it might influence the development approach of a developer in some way it is not evident how the culture influences the development. As Nerur remarks that the Organizational culture has a big impact on the social structure of organizations, which in turn influences the behaviour and actions of people. They also state that the organizational culture has considerable influence on the decision-making process, problem-solving strategies, innovative practices, information filtering, social negotiations, relationships, and planning and control mechanisms. Agile methods depend on quick, responsive and cooperative teams where face-to-face communication will often be preferred over other means of communication [43] , thus the way the customer communicates within its own organization influences the development.

Agile logistical arrangements
Model driven development is applied best when the customer and developer have a high level of cooperation. As mentioned before this can be achieved by iterative development which requires good communication and many confers between the developer and the customer. This requires that the customer and the developer need to be able to confer at the location of the customer as on the location of the developer. Examples of good agile logistical arrangements are conference rooms and open spaces where the whole team can work together to create a fitting solution. The agile arrangements also include network limitations and limitations to employees throughout the building.

Interest in project

During the bargaining process over the contract the choice of contract depends on multiple inputs. One of these is the relative bargaining power of the two parties. This decreases for the customer when the developer knows the amount of interest that the customer has in the project. Assuming that the parties are risk averse, a high interest would increase the chance of a times and material contract.

However, this situation assumes that the parties are always as much risk averse as possible. Rather than applying a contract that might actually fit best to a customer it assumes that the developer will always lobby for a times and materials contract which might not be true in all cases.

Organizational size
Large organizations often have a hierarchical structure and traditionally a more bureaucratic governance structure. But more over if the client is much bigger than the developer, it will have a lot more to bring to the table. Meaning that when contractual negotiations start, the developer is often pushed into a corner where they are forced to choose for a fixed price contract. Even though fixed price is not necessarily a bad thing, applying a contract without awareness of its consequences can prove fatal to the customer relationship and to the project. What we understand under a fixed price project and what its consequences are, is discussed in paragraph 3.5.

In comparison to smaller companies, large companies often have a medium to high maturity in software development in general. This would imply that large companies often perform better during a software development project. However they also have some values that contradict with the main drive behind agile development. As mentioned in the agile manifesto, being agile is all about being quick and responsive to the customer which of course is hard to achieve when you need to confer every week with 250 people. This is also concluded by Alistar and Cockburn who conclude that “agile development is more difficult for larger teams”, but they do cite occasional successful projects with over 250 people involved showing that the size is not critical to the development.[44]

People skill set
The skill set of the people at the customer who are working with the developer is a very important characteristic, and probably one of the most hardest to measure because it can be measured from many different angles. Among others some examples are the experience with MDD projects, experience with requirements analysis and experience with teams.

Because agile and model driven development is focused on collaboration with the customer and working in teams it is important that the people that participate in the project are experienced enough. While people can of course be involved in a project without actively participating, i.e. interviewed or provide requirements.

Tightly coupled with the customer culture, if the contact person or the person who is part of the developer team is known with multi disciplinary teams he is likely to perform and communicate better. This means that more responsibility can be delegated towards the customer.

Customer relationship
Customer relationships provide a lot of input to the way development takes place. For example if the customer has a good relation with the developer communication is likely to go a lot smoother. The opposite however is also true, if a customer and a developer cannot get along or have large conflicting interests, cooperation and communication between the two parties will be stiff.

But the customer relationship also tells us something about how known the customer and his processes are. If the customer is totally new his processes are likely to be unknown as well. CAPE Groep argues however, that whenever a customer performs in the same markets as other customers they do have knowledge about their processes, or at least about their domain. Thus it is not only important if the customer is known, but also if the market the customer operates in is known.

Resistance from groups or individuals
Resistance from groups or individuals is a property that occurs often during big implementations or organizational changes. Whether these are small changes or big ones, as soon as a person feels threatened he might show resistance to the change. However, Dent argues that the resistance that these people show is not against the change perse. People can show resistance to loss of status or loss of job or loss of pay, which is something completely different than the change on its own. He

states that “Strategies for overcoming resistance to change are offered regardless of the change intended.” And that to overcome this, developers should apply targeted actions. “If the anticipated change will result in the loss of status by some employees, then the field must research and develop strategies for dealing with the loss of status. Likewise, if the change will result in the loss of jobs, that issue must be dealt with. Labelling these difficult problems as resistance to change only impedes the change effort.” [45] If this resistance is not challenged it could lead to the loss of productivity and delay in the development or even worse.

4. CONCEPTUAL FRAMEWORK

This chapter proposes a conceptual framework based on the research and state of the art literature provided in chapter 3. The framework is validated and where necessary adapted based on a validation that will be further discussed in chapter 5.

This chapter will combine the knowledge found in chapter three into a framework that can be used to align the development method with the customer and the type of contract. First an overview of the framework is presented. Then, the various steps that are present in the framework are described and explained.

4.1 Framework Overview

This paragraph will provide an overview of the process of the framework that has been derived from experience at CAPE Groep.
The goal of this thesis as discussed in chapter 2 is to improve the efficiency of model driven development projects by aligning the development method with the customer and the contract. In order to achieve this, the process of setting up a development method was derived from experience of CAPE Group. Figure 13 displays the different steps that are taken to determine how to set up the development method and the contract.

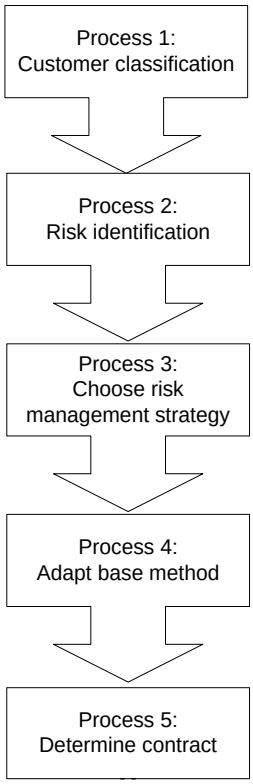


Figure 13: Framework processes

The framework starts with identifying a customer and determining their risk profile. Based on his own business case the developer has to determine how to deal with those risks. When the developer knows how to deal with each risk he can determine how to adapt the development method. The last step is to determine a favourite contract type for the developer that fits the risks of the customer.

As shown in Figure 14, each process step is a transformation of one or more pieces of input. How each of these inputs is transformed will be discussed per process separately. Most of input of each process has been developed during this research. However, the methods to derive the data for these inputs have been proven in previous researches.

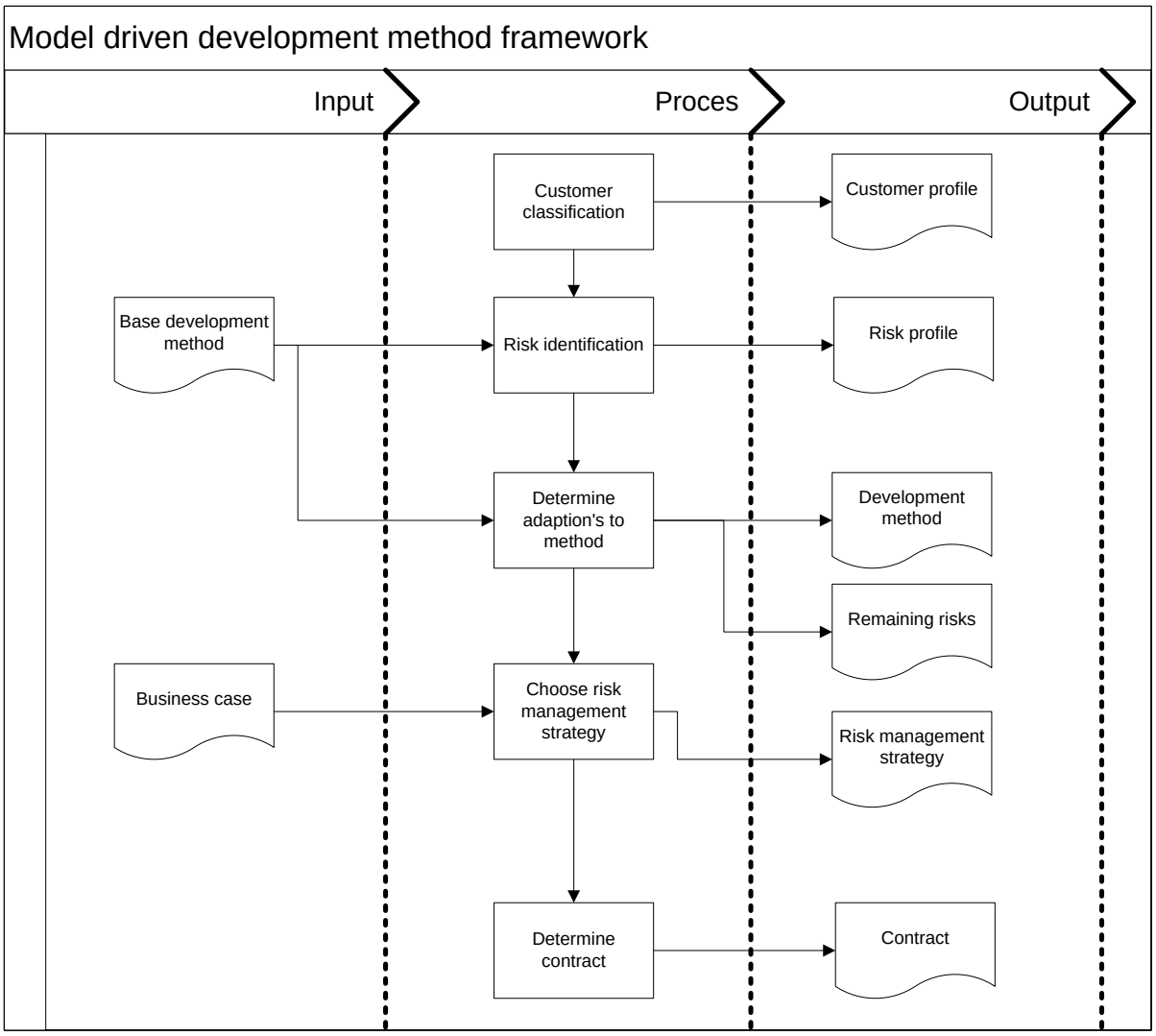


Figure 14: Framework overview

The first step in the framework is the identification of the customer. This is done based on a set of customer characteristics that have been merged during this research. With the help of these characteristics a developer can sketch a profile that should represent a given customer. The method does not introduce customer classes as we think that customers are too diverse to be classified by standard classes.

Based on the customer profile the developer can identify the influence on a set of risks that are present during a model driven development project that uses the base model driven development method. The base model driven development method is a development method that functions as a start off point. It has several parameters that are left open to differ between customers. The risks that are identified all apply to the base method and can be classified by means of their impact and chance on occurrence. There is no need to be exact for each risk as long as they are relative to each other.
When the developer has identified his risks he can determine which risks are to be addressed by the development method and which don't. Based on the risk value that each risk has the developer can determine how to fill in each parameter in the model driven development method. This leaves some residual risks that can be taken into account during the picking of a right form of contract or by other risk management strategies.

Because no project is the same the developer needs to determine how to deal with each risk separately. The method supports this by suggesting four risk management strategies out of which the developer can choose. A driver for the developer to pick a strategy for each risk is his business case. The business case describes the reason and goal for the project and is thus a good direction of how

the developer wants to develop. The method does not support a risk management measure for each type of risk, these will have determined in future research.

After the risk management strategy has been determined and the development method has been chosen a form of contract that fits both can be selected. First 3 new types of contract are explored that are applicable for model driven development projects.

4.2 Model driven development method

Before the processes are explained we first explore the basis of this framework, the model driven development method. This method will be used as a start off point to which the risks that the customer introduces to the project. The method is based on the agile mindset but it also incorporates the possibility to change to a less agile or more rigid approach.

4.2.1 A base model driven development method

Model driven development is still in its early phases and while many projects have adopted a form of model driven development, no agile approach to model driven development could be found. Thus a base development method is introduced based upon existing practices. First an identification of the whole development process is made based on the MDA guide. These processes are supplemented with roles and work products that need to be present in an agile approach. The base development method assumes some basic techniques that are always need to be present in a model driven development project.

4.2.2 Overview

A development method is a very abstract concept that can filled in many different ways. It can provide a standard project setup or a large decision tree specifying each thought of scenario. This development method is based on the agile mindset which is about specifying less and thus only provides the layout of the development trajectory. It does not provide any details on how a work product should look like or what methods are best applied to get the best results.

Figure 15 shows the model driven development method that we propose. It is based upon the agile development methods XP, Scrum and Crystal [25, 28]. The MDA guide is used for its work products in MDD projects and for the activities required to run a model driven project [18]. The roles that are required during an MDD process are gathered from experience from CAPE Groep and from the agile methods.

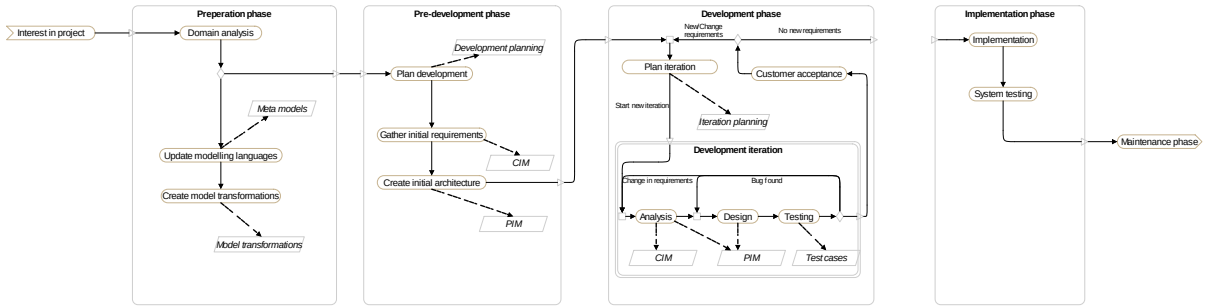


Figure 15: Model driven development method

Presumptions and common practices

The development method is based on techniques and practices common to model driven development and agile development. Agile methods are based around the idea that by doing fast deliveries mistakes are detected early and that by close collaboration requirements and communication are improved.

When comparing the most used agile methods, they all show a development trajectory that consists of roughly 3 phases: an initial identification phase, a development phase and a closing phase. Each development starts with an initial identification of requirements that is expanded during the project. Model driven development can augment this process by storing the requirements directly in a CIM. In our development method we presume that each MDD project has a way of storing requirements in

one or more models. After the initial identification of the requirements the development cycle starts. In Scrum and XP these cycles are called pair programming iteration and sprint [27]. At the end of each development cycle the application is presented to the customer who can then accept an implementation or choose to continue the development in a new iteration. Agile development is also known as iterative development for this reason.
When an iteration is complete, the product is judged and accepted by the customer. If the product is deemed complete it will enter the implementation phase. It should be noted that a product does not necessarily have to be the whole project, rather the project can be implemented in multiple sub products. Both XP and Scrum press the importance of implementing software projects in multiple phases. This is also supported by the agile principle that states that "Working software is the primary measure of progress."

In addition to the development of the application, the development of the models needs to be managed as well. This is why an additional phases next to the three standard phases has been added, the preparation phase.

Below a full description of all phases is given. Each phase has a certain amount of activities in which an actor with the accompanying role participates. Thus each phase has actors with roles, activities and responsibilities. Furthermore, each activity can produce one or more deliverables.

4.2.3 Phases

This paragraph will explore all phases that are present in the model driven development method.

Preparation phase

When a developer starts out with a model driven project he will require a type of modelling language, or a tool to model in. We assume that the developer is already in possession of such a language and thus our development method does not describe how one should create a language. However, modelling languages can get outdated or unfit for a certain solution and thus do require maintenance, which is what is done in the preparation phase.

Models, which are the basis of model driven development, are modelled in a modelling language. This modelling language is the limit of that which can be modelled in a model. Because models are applied to abstract part of a reality, they are often limited and only allow to model that part of reality. For example, a model with only rooms and doors cannot be used to model a bridge. Thus a certain model is applicable to a certain domain. Even a general modelling language such as UML can sometimes be insufficient which is why it will need to be extended. Some criticize that UML is to general which makes it unfit for specialized solutions. However in our method we do not care whether the modellers use UML, DSL factories or another form of modelling. Figure 16 shows the layout of the preparation phase. It starts with a domain analysis in which the modelling languages is analysed whether or not it is sufficient or not. If the modelling language is not sufficient enough to create the application, the developer will require updating its Meta models and model transformation definitions. When these are up to date the developer can continue to the pre development phase if the models cannot be updated in time or are not able to model the needs of the customer the developer has to decide whether or not to continue with the project.

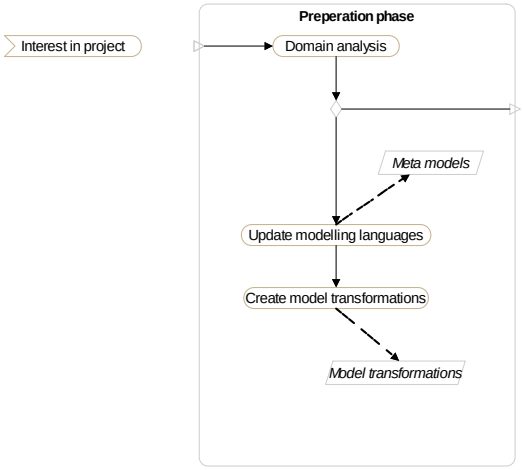


Figure 16: Preparation phase

Domain analysis
During the domain analysis the modellers and business analysts determine whether the existing modelling languages are sufficient enough to model the application. If not is not so, the modellers will have to work together with the DSL expert to create a new modelling language which is sufficient for creating the application.

Update modelling languages
When the modelling languages are not sufficient enough the modellers will have to specify their exact needs as to why the current modelling languages are not sufficient and what they would like to have added. The DSL expert can then update the modelling languages accordingly.

Update model transformations
After the modelling languages have been updated the transformations between the different modelling languages will have to be updated by the DSL expert. These transformations ensure that the new functionality is linked between the different modelling languages.

Pre development
After a project is acquired and it is certain that the modelling languages are able to model the needs of the customer, the development trajectory needs to be planned. An overall planning is made to map out the outlines of the development trajectory. How this planning is made should be determined by the developer, this will not be discussed in this thesis.

Then the initial requirements need to be mapped in a computational independent model(s). It is possible to have multiple models to store the requirements in. This will be done by the business analysts so that the model architect can then create an initial architecture based on the requirements. Existing agile methods such as Scrum and XP allow the customer to update the CIM equivalents. They have a continuous process that keeps updating the requirements. In our method this task is handled by the computational independent models that store the requirements and can be accessed at any time. The task of filling the initial requirements models can be executed by both customer as the developer. However because the modelling does require some modelling experience this introduces a risk to the project. This risk will be addressed later in paragraph 4.3.

The initial architecture forms the basis of the application, where it will fit in the IT infrastructure and what the application will look like. With the help of model checking these requirements can later be mapped to the actual application in a platform independent model.

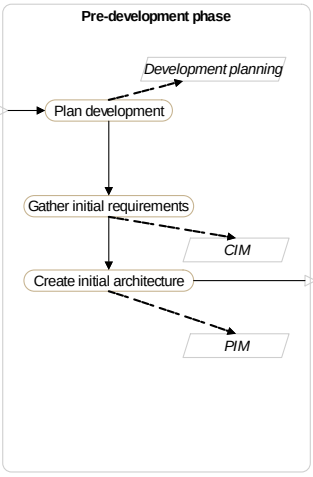


Figure 17: Pre-development phase

Plan development
The pre-development phase starts with the plan development. During this activity the whole development is planned e.g. the milestones are planned. This activity is executed by the project manager of the developer and the project manager of the customer.

Gather initial requirements
When the development has been planned the initial requirements can be gathered, these requirements can be stored directly into a computational independent model. If the model driven development environment does not support this, it should be done separately, for example in a standard office requirements document. The gathering of initial requirements is executed by the business analysts.

Create initial architecture
When the first requirements have been provided the architect can create the initial architecture. The architecture depicts the basics of the domain model, or the application design. In the object oriented world this would compare to the database design which forms the basis of the application. Decisions that need to be taken into account are the place of the application in the application grid of the customer and the amount of users.

Development
During the development phase the actual development takes place. It is during this phase that the application is modelled and tested. As in all agile methods an iterative approach is chosen, that enables feedback from the customer so that the development can cope with changing requirements and increase their quality. An iteration can vary in length depending on the amount of features that need to be delivered but also on the wishes of the customer. Quick deliveries allow for quick feedback and thus better requirements and it also provides more support among as results are shown early. However, very short iterations also provide an increase in management overhead. Both Scrum and XP define a maximum of 1 month per iteration, thus on average we would suggest a 2 week iteration that might shift depending on the customer and the amount of functionality that needs to be implemented.

The development phase is split into three separate processes analysis, design and testing. During the analysis process the requirements are interpreted and translated into models. The process then flows over into the design process which is a combination of the programming and design phase. Model driven development allows programmers to program using models, eliminating the need for a separate coding phase. Model driven development also provides the ability of model checking which checks if fields are filled and if the models meet their specification. This allows the modellers to see whether or not the requirements are met and if a model is technically sound. The latter allows the

decrease of the amount of technical testing, functional testing still is required though and thus use cases need to be made to test the application.

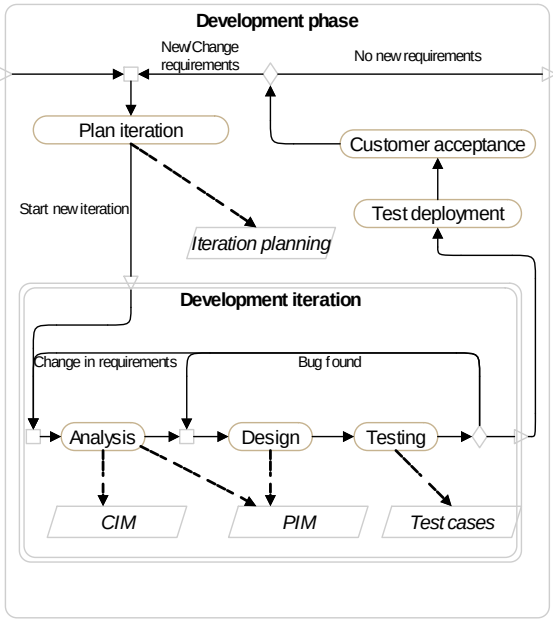


Figure 18: Development phase

In many agile software development methods the customer is often part of the development team. After a part of the application is found well enough to be tested by the customer, the customer tests the application to provide new requirements and feature requests. These can then be taken into account in a new iteration planning. During the whole development cycle the customer is allowed to gather requirements that can be discussed during the plan iteration activity.

Plan iteration
During the planning of the iteration the requirements that are to be covered during the iteration are discussed. The iteration planning that is produced describes all of these requirements. The planning of the iteration is executed by both the project managers of the project. If the model driven environment allows it and the customer has enough experience the requirements could also be directly stored into a computational independent model. This would drop the need for the analysis activity.

Analysis
When the iteration has been planned the requirements that were gathered need to be translated to the computational independent model. The requirements that are stored in the iteration planning are then put into the model. This activity is executed by the business analysts.

Design
As soon as the computational models or requirements models are filled, the modellers can start modelling the logic and layout of the application. The architecture should be nearly finished as this is largely covered by the architect during the activity create initial architecture.

Testing
Whenever a piece of functionality is finished during an iteration it needs to be tested by a tester preferably someone else next to the modeller. Whenever someone tests their own code they are not likely to perform any unexpected behaviour and thus not really test the application. It is also possible to let the customer test the application. If any bugs are found or if during the iteration small requirement changes are made the design can be altered and should be tested again. If the customer is not testing the application they should provide test cases to ensure that all functionality is fully tested.

Test deployment
Should the testing in the iteration not be performed by the customer, the customer will need to be able to perform tests locally. Thus a test version of the application needs to be deployed as soon as an

iteration is finished. This test deployment will also allow the customer to further test the application during the next iteration. The test deployment can be performed by a modeller or by the architect.

Customer acceptation
When all requirements of the iteration are covered and the application is deployed on the test environment the customer can accept or reject the iteration. When it is accepted the next iteration can be planned or if it was the final acceptation the application can be deployed. Customer acceptation is executed by both the project managers.

Implementation

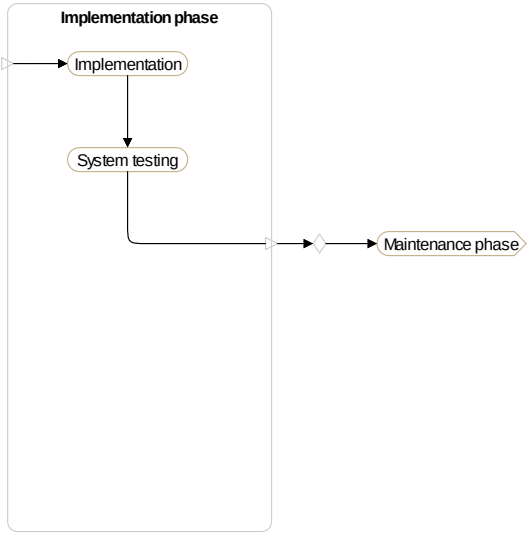


Figure 19: Implementation phase

The implementation phase starts when part of application is accepted and ready to be implemented at the customer. This requires extra testing and checking of the performance of the system and the integration with other systems. Depending on the development planning multiple implementation phases can occur.

Implementation
The application is implemented partly or as a whole on the production environment of the customer. This activity requires extensive testing and is often executed with a fade-out time. How this is executed should be discussed with each customer separately. The implementation should be executed by the architect or by an experience modeller.

System testing
As soon as the implementation is finished the system and system integration should be tested to ensure that they are working. The type of tests and the extensiveness of the tests is different among different customers and should thus be discussed with each customer separately. System testing can be performed by the architect or a modeller but can also be outsourced to the customer.

4.2.4 Roles and responsibilities
The roles described in each of the activities above are further explained in this paragraph. Each role performs at least one core activity that cannot be performed by any other role. The roles are based on practices of MDD and MDA and agile projects [17-18, 44, 46].

Project manager
A project always has two project managers, one at the side of the customer and one at the side of the developer. In a smaller development team the project manager is part of the actual development team where in larger projects he might have a separate role. The main activity of the project manager is managing of the project. All activities that cover planning are performed by the project manager.

Model architect

The architect is an experienced modeller. The core activity of the architect is to provide the initial architecture of the application, additional tasks of the architect would be the implementation or deployment of the application and the system testing. A project should have one model architect at the side of the developer.

Business analyst
The business analyst is the main communication channel between the modellers and the business. They should be able to use the modelling tool to model the requirements in the computational independent models and understand the modelling language of the platform independent models. The role of business analyst can be performed both by the developer as by the customer. Each project should have at least one business analyst.

Modeller
The modeller models and designs the application in the platform independent models. They can be seen as the programmers of classical development projects. The modellers are technical persons that make design decisions about the application. Each project should have at least one modeller. While it would be possible to let the customer assume the role of modeller, we would not suggest this. Letting the customer make design decisions might change or even ruin a project unexpected. Often modellers are also testers. If the project team is very a small a modeller might even be a business analyst.

Tester
The tester does functional testing to check whether the functionality meets the requirements. Both the customer and the developer can fulfil this role. Every project should have at least one tester.

DSL expert
The DSL expert is responsible for keeping the modelling languages up to date. Whenever a modelling language requires an update this will be done by the DSL expert. This also includes the transformations between the modelling languages. Each project should have one DSL expert or someone responsible for updating the modelling languages. As model driven tools are sometimes purchased rather than developed the DSL expert could be also a third party.

RACI
To get a better picture of the role that each role plays during the development method we introduce a RACI table shown in Table 5. RACI stands for responsible, accountable, consulted and informed and provides a separation of responsibilities for a set of tasks. The responsibilities drawn are a recommendation based on responsibilities as drawn by Warsta. [27]

	Project manager	**Model architect**	**Business analyst**	**Modeller**	**Model programmer**	**Tester**	**DSL expert**
Domain analysis	A	C	R	C			I
Update modelling language			I	C			A,R
Update model transformations							A,R
Plan development	A,R	C	C	C	I		
Gather initial requirements	I		A, R				
Create initial architecture		R					
Plan iteration	A		R	C			
Analysis			R	R			
Design	A			R	R		
Testing			I	I	I	A,R	
Deploy test						A,R	
Customer acceptation	A,R	I	C	I	I		

Implementation	I	A,R					
System testing		A				R	

Table 5: Responsibilities in the base development method.

4.2.4.1 Deliverables

The Development method as shown in Figure 15 shows 4 phases each containing several activities of which some produce several work products. This paragraph further explains each deliverable, what they contain and how they are used.

Meta models

Whenever the current modelling language is not sufficient enough to provide the application that the customer wants the modelling language or meta-models need to be updated. Assuming that the developer wants to pursue the project with the model driven approach and not develop with a different technology. The meta-models that are produced during the preparation phase ensure that both the business analysts as the modellers are able to produce the CIMs as the PIMs. The Meta models are created or updated by a DSL expert.

Model transformations

Once the Meta models are complete the transformations between the Meta models are likely to be updated as well. This will ensure that any of the new content that is used is linked between the different layers i.e. from CIM to PIM to PSM. As the Meta models the model transformations are created by the DSL expert.

CIM PIM

The CIM and PIM models form the code of the application where the CIM models store the requirements and the PIM models model the architecture and the logic behind the application. During each activity has a work product CIM or PIM, multiple models can be generated. It should be noted that the actual code or PSMs are not modelled because they are not created by human activity, rather they are generated by the model driven tools. For a full explanation of what these models are see paragraph 3.3.2.

Development planning

The development planning is the main project planning that contains all important deadlines for work products to be delivered. The planning is agreed upon by the project manager of both the customer as the developer.

Iteration planning

Each iteration starts with the planning of the iteration. During this planning the project managers of the customer and developer agree upon the requirements that are to be covered in the iteration.

4.2.5 Parameters of the development method

A development method in its own way can be seen as a form of risk management or risk prevention. It is often based on bad experiences from the past or opportunities that were missed. This is also the reason that many companies apply equal development methods in a different way. If one important lesson can be taken from this fact, it is that a development method is never perfect and cannot be followed to the letter in all situations. This is why we introduce dimensions in which the development method can be adapted to different types of customers and contracts. To realize this we have to define a set of dimensions along which the development method can shift between developers and customers.
When looking at the definition of our development method we specified three parts: the roles, the activities and the deliverables. Any adaption's that happen to the development method have to relate directly to any of those three. Based on the agile success factors and the model driven development practices we explored in paragraph 3.7.2 and 3.3 we came up with the following dimensions.

Agile development should be performed with the customer especially now that model driven development provides an opportunity to close the gap between business and IT. However, sometimes the customer might not be up to this, or might not want to free his resources to work in such a way. This means that the customer cannot always be part of the actual development team and thus cannot always perform the modelling of requirements or test the application during development. Another

example is when the customer wants to perform some tasks but the developer thinks the customer is not ready to do so. Thus the ability of the customer to assume an active role in the project also depends on his capabilities to fulfil them. It is also possible that some responsibilities of the developer are directed to the customer. If the customer is not up to the job there are two choices the developer can take, one is to provide training the customer to perform the job and the second is to perform the job themselves.

The second dimension is the location at which the development takes place. The agile manifesto states that one of the key aspects of agile development is face-to-face communication. But in order for this communication to happen frequently it would require the developer to develop on site at the customer. While this on its own is easily reached it can be hindered by numerous factors. We call these factors agile arrangements. A good location can provide quicker communication and thus if the customer cannot provide an agile location the developer could choose to develop at another location.

The third dimension is the time that the development takes. While a maximum of one month per development cycle or iteration is agreed upon by most agile methods, they do not agree on an average cycle time. Within this month however it can take longer or shorter. Shorter development iterations allow for quick feedback and a higher support among the customer employees. But the customer needs to be able to keep up and shorter iterations also provide a higher amount of management overhead. By default our development method assumes an iteration time of 2 weeks.

The last dimension is the amount of requirements documentation that is created at the start of the project. In general agile is renowned for the art of documenting less. While to some documenting might be seen as a waste of time, others swear by full documentation. Both approaches are preferred in different situations. In our method we let the developer decide on how much to document. We do propose that they decide on this with help of the FURPS+ method.

It can be noted that each parameter can be filled with either a customer directed solution or with a developer directed salutation. This can also be seen as the same distinction that exists between agile development and classical development. In basis these parameters thus allow one to define how agile his method should be. Table 6 shows a summary of the parameters.

Method parameters	Agile	Classic
Role distribution	Customer	Developer
Iteration time	1-2 weeks	2-4 weeks
Location	Customer	Developer
Requirements documentation	Few documentation	Full documentation

Table 6: Method parameters

Even though agile is favoured above classical development in general, it might sometimes be needed to adept to a more classical approach to avoid certain risks or simply because it would be too expensive to do it otherwise. To choose between the different possibilities of the parameters we require a way to determine if the customer and the contract are suitable for the agile approach. A grounded decision between both ways can only be made if the outcomes or the risks of both ways are known. Thus the possible outcomes of both approaches need to be discovered. The following paragraph will explore the customer properties and couple these to known software risks. Those risks can then be mapped to the parameters. When both mappings are made it becomes possible to determine the risks by means of the customer properties.

4.3 Risk identification

Like in any software project there are many risks that can occur during a model driven software project. Some are specific to model driven development, where others are more applicable to MDD. Also some occur very often but with minor effects while others occur almost never but with great effects. This also entails that some risks are very dangerous to a project while others can be neglected. Because it is simply too hard and too costly to manage all risks we set up a check list of the most important risks for each method parameter. This will be done equally for both the agile setup as for the classical setup. In the ideal situation none of the given risks would occur. It should be taken into account that in practice this is near to impossible as some risks are bound to happen when others are shut out. Table 7 displays an overview of all the risks for each development parameter.

Method parameters	Risks
Business analyst role	Missing requirements, unclear requirements, badly modelled requirements, bad cooperation
Tester role	Bugged release, missing functionality, bad cooperation, refuse solution
Iteration time	Changing requirements
Location	Bad facilities, bad cooperation, large travel distance
Initial requirements	Gold plating, non fitting end product, unknown modelling restrictions

Table 7: Model driven development risks

4.3.1 Business analyst risks

The business analyst does the analysis and documentation of the requirements during a project. Thus if his role is performed poorly it is likely that the requirements he produced are also poor. Hence, that most risks that are connected to the business analyst affect the requirements. Requirements could be missing or unclear, but also they could be badly modelled. With MDD as we defined it, the requirements are modelled in the CIM. If the business analyst has never modelled before it is likely for him to model the requirements wrong. Another risk that is connected to a role description is bad cooperation. When a team configuration is not setup well it will lead to bad cooperation between the customer and the developer. Especially if the developer and customer have different ideas on how to lead a team it could lead to problems within the project team.

4.3.2 Tester risks

The testers in a project team ensure that functionality is present and is working as intended. If testing is performed poorly it can result in bugged releases and missing functionality that need to be restored it should be noted that no matter who performed this task it will result in delay of the project. Also because this parameter is a role it can result in bad cooperation depending on the how well the customer and developer can work together.
At last there is also the factor of resistance, if a customer is part of the development because he can test the application they feel involved with the process. In overall involvement causes less resistance and thus reduces the chance to refuse the solution.

4.3.3 Iteration time risks

Iteration time depends how long it takes until the developer delivers part of functionality and hence how often he will have feedback from the customer. When iteration times change it influences the chance and impact of changing requirements. The shorter iterations become the more likely it is that the developer will introduce changing requirements. As Whang states it, "*Prospective users cannot evaluate a system before they actually 'see' the system.* "[30] While we think that it might be possible for users to evaluate a system before hand, we also think that showing them is surely to reveal some shortcomings of a system. Thus creating shorter iteration times will increase the amount of errors found early on but will also increase the amount of changing requirements. Discovering errors early on will reduce the impact of changing requirements, however due to the shorter iterations the likeliness of the occurrence will increase. It thus becomes important to determine up front if a user is able to specify his requirements into detail so that the risk of faulty specifications and accompanying changing requirements can be determined.

4.3.4 Location risks

Agile developers are keen of developing at the developer as this would result into better requirements. If a requirement is unclear the developer is able to instantly ask the customer what a certain requirement entails. However, this weights heavily upon the available facilities and employees. When the facilities are so that the developer is separated from the customer he will not be able to request the information and hence the development would only be slowed down. This is worsened if the customer is geographically located far away. The longer the distance the more travelling hours a developer makes which are associated with high costs and also with increased development time, as time spend travelling is not spend developing.

4.3.5 Initial requirements risks

The point that is often seen as the main difference between agile development and classical development are the initial requirements. Where agile development makes use of a small initial document that only identifies the main requirements, classical development tries to identify all requirements so that no changes will be required afterwards. Both versions have their down sides, but the most important risks connected to this choice are gold plating and a non fitting end product. If the

requirements are not documented well from the beginning and the customer changes wishes some extra functionality it could lead to the developer constantly building extra functionality that is not needed, this is called gold plating. The second risk is that if all requirements are fixed up front then the developer might deliver a product that the customer does not need. As stated by Whang, a customer often does not know what they actually need before they see it. Thus fully fixing the requirements can also be risky.

Last, the initial requirements are also influence by the risk of MDD restrictions. When the requirements are known, it is often known to the developer how to develop the application. Thus the developer will know up front if the application can be modelled with the current models and transformations. If the requirements are not stored however, the developer might have a hard time envisioning the end product and thus is not able to see if the project might require an addition of models and model transformations.

4.4 Customer identification

As described in paragraph 3.7.1 a customer profile can be used to identify and categorize a customer. Based on the customer properties we explored in paragraph 3.7.5 we introduce a customer profile that allows a developer to measure the customer and identify the risk that the customer introduces to a model driven development project. Rather than making each parameter measurable we introduce a relative measure, so that each developer can decide on their own if the parameter is sufficient or not. Each parameter is thus good enough for the developer to apply the development method as presented in the following paragraph. Because the desired values for these parameters can differ between different customers we did not specify a targeted value.

For each parameter risks are identified and examples are provided on how to measure the parameter. The possible outcomes allow the developer to make a grounded decision on how to fill in his development method. In order to create a full customer profile first a risk analysis is done to find the most common risks that might be solved by the development method. These risks have been gathered from experience at CAPE Groep and from literature describing agile practices and software risks.[47] [48]

Management support

Management decides about a lot of things in the project, it is thus important for them to support the project. When this is not present management can appoint the wrong business owner to the project, because he might have time off, or they could limit the budget leading both to a decrease of efficiency and possibly to failure of the project. It is not without cause that in software development literature, many sources appoint management support as a top tier risk [3, 47-48]. Another direct effect is that when top management is not supporting the solution it might show on the end users, discouraging them to use a solution that management is not even supporting. However, if management support is present the budget is likely to become larger as is the support among employees of the customer. A good way to measure if management support is present is by using interviews and by testing how well the business case is set up. As Whittaker concludes, the involvement of management and the business case determine whether or not management support is present [42].

Organizational structure

The risks of an organizational structure is that when this becomes too large and hierarchical the structure delays the communication and hence the agility of the project. Furthermore, as was discussed in the initial problem identification, extra communication reduces the efficiency of projects.

Because the model driven development method we introduce is agile, it requires being fast and working in collaboration with the customer. The organizational structure pictures the hierarchy of the company. When doing a project in a large company with a hierarchical structure it can delay the taking of decisions and the development in overall. While no changes or agreements can be made to change the organizational structure, its disadvantages can be compensated by having people in the development team who are allowed and dare to take decisions. Having good management support is also required for this as management decides on who is in the project team what their authority is. When the organizational structure is flat and allows for easy communication between layers, it becomes more useful to develop together with the customer. Also developing at the customer becomes more profitable. While the communication does not have to be bad in hierarchical

companies it is important to identify this. This can be done based on the amount of managers per employee.

Organizational culture
Unlike the organizational structure the culture of the customer is not easily measureable. It cannot be detected by the amount of managers or their behaviour rather, it has to be observed.

What can be detected is the level of trust and authorization that the business owners have during a project. The less authorization he has the higher level of bureaucracy. A higher level of bureaucracy leads to slower development. As Nerur remarks, the Organizational culture has a big impact on the social structure of organizations, which in turn influences the behaviour and actions of people. They also state that the organizational culture has considerable influence on the decision-making process, problem-solving strategies, innovative practices, information filtering, social negotiations, relationships, and planning and control mechanisms. Agile methods depend on quick, responsive and cooperative teams, face-to-face communication will often be preferred over other means of communication [43] , thus the way the customer communicates within its own organization influences the cooperation and communication that takes place during the development.

Agile arrangements
Model driven development is applied best when the customer and developer have a high level of cooperation. As mentioned in chapter 3.7 this can be achieved by iterative development which requires good communication and many confers between the developer and the customer. This requires that the customer and the developer need to be able to confer at the location of the customer as on the location of the developer. Examples of good agile logistical arrangements are conference rooms and open spaces where the whole team can work together to create a fitting solution. The agile arrangements also include network limitations and limitations to employees throughout the building. If these are not present they will slow down development at the location of the customer. Also the communication is disturbed as face-to-face communication might not be possible or only partly be possible.

General IT experience
When the customer has no or not much general IT experience he is not likely to have done any requirements gathering before. When the customer has not provided any requirements before it is likely that his requirements are going to change during development. Furthermore, if not enough requirements are frozen during the development a customer with no general IT experience might constantly change his wishes leading to endless development. General IT experience also influences the risk of under-funding, when the customer has no IT experience at all he might want to much for what is actually possible during the development. The amount of IT experience cannot be changed but can be measured by the presence of an IT department as the previously done projects by the business owner.

Modelling experience
Model driven development is often able to provide quick results when it comes to layout. This however could make a customer believe that the application is nearly finished when it is actually only a shell. If a customer has no modelling experience his expectations might become abnormally large because of the seen progress. This is worsened if the customer has no knowledge of the modelling technique. Thus the modelling experience of the customer introduces a danger of having false expectations which could result in a lower budget.

Another risk that is introduced by the modelling experience is that the customer will model the requirements wrongly. This is only the case when the customer is given the role of business analyst. This could be countered by providing the customer with training or by letting the developer do the requirements analysis during development.

Teamwork
The risk of working in a team when the customer is not used to this, is that the communication in the team is bad or that the cooperation between the customer and the developer is bad. For example, if the customer decides to introduce new requirements in the model but does not notify the developer of this update it will not be noted and the development will be delayed. Furthermore, if decisions taken by management of the customer are not relayed to the developer it can cause overhead on meetings.

The more experience the customer has with teamwork projects the less the risk of overhead, bad communication and cooperation. Teamwork can be measured by a simple interview asking if the customer ever works in multidisciplinary teams.

Customer relationship
As with teamwork the relationship that the developer has with the customer determines the amount of trust the developer has in the customer. However, it determines more for example, a customer could be a known but entering a new market and thus develop new processes. The processes will also be new to the developer and hence the development might take longer. The other extreme would be an unknown customer in an unknown market. This would require a lot of devotion from the developer to get to know the market and the customer. If both the developer and the customer would operate under such circumstances it is important to gain some domain knowledge during development otherwise poor requirements could be the result.

Besides the customer being new or not the relationship with known customers also differentiates between different customers. While one customer might be considered as an almost a strategic ally, another one could be known for deliberately holding back on information or for always driving a hard bargain. All of these factors influence how the cooperation fares between the customer and the developer.

Resistance from groups or individuals
Resistance is a hard to measure parameter as it is not likely to be noticed upfront. Never the less the parameter should still be considered. If any form of resistance occurs it could cause a refusal of the solution by its end users. Thus it is important that the developer and the customer together identify any resistance and deal with it properly. One example would be to involve the end users and enlighten them with the goal of the application. Users can also be involved during the testing period so that they can actually provide feedback during the development. Thus additional end users might be involved during the testing period.

Customer properties	Customer characteristics	Risks
Management support	• Business Owners • Business Case • Project team	• Low support among employees • Low budget
Organizational structure	• Size • Hierarchy	• Slow decision making
Organizational culture	• Working hours • Work style • Decision power	• Bad cooperation • Slow decision making • Slow communication
Agile arrangements	• Facilities • Employee access • Network access	• Slow communication • Slow development • Slow analysis of requirements
General IT experience	• Amount of similar IT projects done	• Unclear requirements • Low budget • Frequently changing requirements
Modelling experience	• Experience with software modelling	• Wrongly modelled requirements • Wrong expectations

Teamwork	• Projects in daily operation	• Slow communication • Bad cooperation
Customer relationship	• Market • Experience with customer	• Not enough domain knowledge • Bad cooperation
Resistance of groups or individuals	• Amount of users • Type of Users • Management support	• Refuse solution

Table 8: Customer profile

Table 8 provides an overview of the customer profile that has be derived from the customer properties and their characteristics. Each of these properties has been supplemented with risks that are a direct result of those properties.

Now that both the risks of the customer as those from the development method have been introduced in Table 6 and Table 7 they can be linked to see which customer properties influence what development method parameter. With help of this link a developer should be able to make an educated guess about each risk based on the characteristics of a customer. It is not surprising that some risks are unique and it thus seems that they do not influence the development method or they are not caused by the customer. It could also be that they are not identified in one of the two identifications. Table 9 shows the mapping between the customer properties and the development parameters.

Customer property	Risks	Development parameter
Management support		
Organizational structure	Bad cooperation	Tester role, Business analyst role, Location
Organizational culture	Bad cooperation	Tester role, Business analyst role, Location
Agile arrangements	Bad facilities, Bad cooperation, large travel distance, Unclear requirements	Location, Business analyst
General IT experience	Changing requirements, Unclear requirements, Missing requirements	Iteration time, Initial requirements
Modelling experience	Badly modelled requirements	Business analyst role
Teamwork	Bad cooperation	Tester role, Business analyst role, Location
Resistance of groups or individuals	Refuse solution	Tester role

Table 9: Mapping of customer properties to development parameters

The first thing that one might notice when looking at Table 9 is that management support is left empty. During the pairing of risks between the properties and the parameters none of the risks matched each other which left us to believe that the management support does not directly influence the choice of development method. However, because most checklists in literature do identify it as a mayor critical success factor in the development we will leave it in our customer profile.

The next thing revealed, is that the following are not mapped: risks bugged release, missing functionality, gold plating, unknown modelling restrictions and non fitting end product. Thus for each of these risks a deeper identification has to be made to identify a possible customer property for them.

Bugged release
A bugged release is caused by bad testing. As stated by Davis not detecting errors early may contribute to skyrocketing software costs. [49] Thus releasing a bugged version will cause an increase in costs due to extra labour in digging up bugs that might not be repairable. In order reduce the costs of software projects it is important to test the application well before release. Besides functional testing it is also important to test the application technical. However, due to the nature of model driven development it can be greatly reduced. Whenever a model and its model transformation have been proven it will only be reused thus decreasing technical flaws compared to a total new design. Furthermore, technical testing should always be done by the developer as he is responsible and is likely to have the most knowledge of this. Functional testing however can be performed by both the developer and the customer but as said above is a point that should not be underestimated. It will increase the involvement of the customer but can decrease the level of tests. In order to get a good idea if the customer is up to the job the developer could ask for experience with comparable projects placing the risk at general IT experience.

Missing functionality
As with the bugged release, missing functionality is also caused by bad testing. Whenever a tester does not perform his job well it might be that functionality is overlooked by both the developer and the tester. Likewise, this chance will increase if the customer has never before performed a comparable project. Thus missing functionality can also be added to general IT experience.

Gold plating
Gold plating has always been a great risk in software engineering. Already in 1989 did Boehm identify this as a top 10 software risk. Gold plating is caused when the customer constantly requests new functionality or when they constantly request upgrades of existing functionality. In either way the software project will not near its end and has a chance of complete failure. Another possibility is when the developer is not given any hard limit in his budget. This could result in the developer loosing track of his true scope what could also result in gold plating. Because this framework is based on the risks that the customer introduces the later is left out of scope but we do emphasize that developers should always keep track of their scope. Taking that the former into account, gold plating can be determined by general IT experience, one could even say that it is directly caused by the chance that requirements are going to chance during development. The more likely they are to change, the higher the chance of gold plating. If the requirements are set it is easier to keep track of what should and what should not be developed. This does however not completely remove the chance of gold plating as the developer could still lose sight of the scope.

Unknown modelling restrictions
Model driven development is based on the idea that components or models can be reused during projects. This leads to a more robust form of development and also leads to faster development as less testing is required over time. Furthermore, adaptation's can be made quicker due to the fact that alternatives already exist. But as soon as a part is discovered that cannot be modelled yet the question arises whether this should be added to the models or not. If an application needs to be created that currently cannot be modelled the model languages or Meta models needs to be updated. This will increase development time as the developer cannot build the functionality for the application yet. If the requirements are known up front then the developer is more likely to reveal any shortcomings of his model driven development method then when the requirements are determined iterative. Determining if the customer is sure about his requirements can be done based on his experience with comparable projects.

Non fitting end product
When a product is developed with fixed requirements it has a chance to become an unused product because it is not useful to the customer. This is the case when the customer thought he needed something when he actually needed something else. Like the other unmapped risks a non fitting end product can also be measured by the general it experience of a customer. The more often they have performed comparable IT projects and the more often they have been involved with implementations the more knowledge they are likely to have of this. More knowledge will result in a higher chance of providing good requirements that will not change too much to make a product unusable.

Customer property	Customer	Risks	Development

	characteristics		parameter
Organizational structure	• Size • Amount of managers	Bad cooperation	Tester role, Business analyst role, Location
Organizational culture	• Working hours • Work style • Decision power	Bad cooperation	Tester role, Business analyst role, Location
Agile arrangements	• Facilities • Employee access • Network access	Bad facilities, Bad cooperation, large travel distance, Unclear requirements	Location, Business analyst
General IT experience	• Amount of similar IT projects done	Changing requirements, Unclear requirements, Missing requirements, Missing functionality, Gold plating, Unknown modelling restrictions, Non fitting end product	Iteration time, Initial requirements, Tester role, Business analyst
Modelling experience	• Experience with software modelling	Badly modelled requirements	Business analyst role
Teamwork	• Projects in daily operation	Bad cooperation	Tester role, Business analyst role, Location
Resistance of groups or individuals	• Amount of users • Type of Users • Management support	Refuse solution	Tester role

Table 10: Final risk mapping

4.4.1 Conclusion

A mapping between the customer properties and the development parameters has been made based on the risks that both share. Table 10 shows the final risk mapping. The following paragraph will explain how these risks can be calculated and how this influences the choice of the development parameters.

4.5 Risk determination

The customer and risk profile presented in the previous paragraph provide the ability to selective judge a customer based only on the characteristics that influence the development method. In order to calculate the risks a choice has to be made whether to use the 100 dollar bill method or the risk exposure method. The later is chosen as the 100 dollar bill method makes it more subject to the developer. Furthermore, by choosing for the risk exposure method this framework could be expended with actual models to calculate the risk rather than determining them.

To determine the risk with help of risk exposure the developer has to determine the impact and the chance of each risk separately for each situation. For example, if the customer is going to be a business analyst how much risk is there on missing requirements? This can be determined by the amount of similar IT projects done by the customer. If the customer has performed similar IT projects then the chance of him missing any requirements will likely be low. The same risk has then to be determined by the classical counterpart. Here we assume that the developer knows out of his own experience how well he is in determining requirements thus how likely it is for him to miss a requirement.

Say that in this case the customer has a lot of experience and thus the chance of him missing a requirement is estimated at 15%. The impact of missing a requirement is estimated at € 2000 this comes to a risk exposure of 2000*0, 15 = 300. Say that the developer is also an experience developer who has the same odds as the customer then this would result in the following.

Method parameters	Risk	Agile risk exposure	Classical risks exposure
Business analyst role	Missing requirement	300	300

Table 11: Risk exposure

However, now we need to add the chance of unclear requirements. Unclear requirements are determined by general IT experience as the agile arrangements. Because both the developer and the customer are evenly experience the chance of unclear requirements remains the same. However, if the agile arrangements are poor at the customer, it is likely that the customer will have less trouble with unclear requirements. As he is likely to have less trouble with asking question internally rather than the developer. Thus the impact in case the developer is larger, let's say that the customer has an impact of €2000 where the developer has an impact of €3000 with a chance of both 15%. This will result in the following:

Method parameters	Risk	Agile risk exposure	Classical risks exposure
Business analyst role	Missing requirements	300	300
Business analyst role	Unclear requirements	300	450
Total		600	750

Table 12: Risk exposure calculation

In this case the classical choice becomes more risky then the agile choice and thus less attractive for both the developer and the customer.

4.6 Risk Management

When the risks have been identified and their exposure has been determined a grounded decision can be made on how to fill in the development parameters. However, this still leaves us with the remaining risks as each choice in the parameter still has its own risks.
In order to manage these residual risks, certain counter actions or agreements have to be determined and executed. One way to steer these counter actions is by use of the project dimensions identified in paragraph 3.5.2. These project dimensions Money, Scope, Time and Quality can be adapted to cover certain risks. In order to clarify this each dimension has been supplemented with an example.

Money
If a risk has a risk exposure of €300,- the developer could ask the customer to pay that €300,- in order to cover the risk. In reality risks will never be fully covered this way because otherwise projects would become way too expensive.

Scope
If the developer has a large chance of deadline overruns due to an undefined scope, he could fix the requirements and with that the scope to reduce the risk.

Time
If the developer is short of time, he could push back the deployment schedule and so create more time to release his deadline.

It should be noted that all of these measures are bound to the agreements which have been laid down in the contract between the developer and the customer.

For each risk the developer has to determine how he wishes to handle and manage them. Depending on what is acceptable for the developer, he will react in a certain way. By determining what is acceptable and thus when a risk is properly managed or is being accepted there are numerous factors that play a role. The following are the most important ones: (Inter) national laws, politics, image and a personal risk attitude. The personal risk attitude is often dependant on the business case that the developer has on a certain project.

4.6.1 Business cases

The business case captures the reason for initiating the project. Of course there are an unlimited amount of business cases to define. However in this context we have identified 3 different types of business cases that we found suitable. These are: earning money, earning a customer, and learning from a project

Earning money
When the developer has a long term relationship with a customer this will result in mutual trust but often higher costs as the customer is likely to pay for a service.

Earning a customer
Often when a developer wishes to grow via a customer they want to gain a customer's trust by accepting lower prices for a project and by accepting more risks.

Learn from project
Whenever and a customer want to achieve something new and innovative or when the developer wants to allow new employees to learn from a project they are likely to agree upon longer development times or increased costs.

Each of these business cases can be mapped to one of the four risk management strategies introduced in paragraph 3.6.4: Prevent, reduce, accept and share.

4.6.2 Risk management strategies

When a developer wants to earn money he is likely to accept as few risks as possible and thus avoid them as much as possible. If risks cannot be avoided he will try to share them, and then reduce them before finally accepting them.

When a developer wants to learn from a project he likely does so in collaboration with the customer. It is not uncommon that a customer also wants to learn something from a project. This will make it easier to share the risk together with the customer by and thus if the risk would ever come to pass the costs would be split. For every risk that the customer is not willing to share the developer can then use the same sequence as with the earning money business case.

When the developer wants to gain a customer or earn the customer he is likely to accept more risks. However, assuming that no one is keen of risks we do assume that he will still try to reduce them for his own gain as much as possible. If it is not possible to further reduce a risks that is still too high a customer should try to avoid the risk. If that is not possible he should try to share the risk. When trying to gain a customer, the customer should be least troubled by risks and hence why sharing is used as last resort.

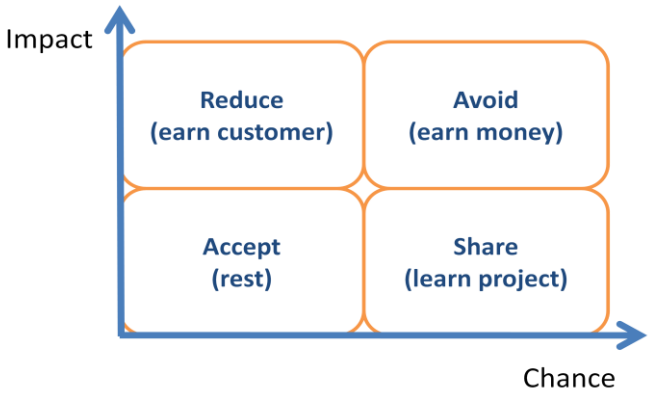


Figure 20: Risk management strategy.

When a developer has selected his risk management strategy he is able to determine how to deal with each risk separately. As this thesis is about aligning the development method and the contract

with the customer, proposing a full risk management for each theory is left out of scope and is left for future research.

4.6.3 Conclusion

A developer can choose between three business cases for each development project that allow him to steer how he manages his risk. For each business case, earning money, earning customer and learn from project a risk management strategy is proposed that gives him a guideline on how to manage his risks.

4.7 Selecting a suitable contract

Now that a development method for MDD projects is clear and a way has been found to identify the customer, a preference for a type of contract needs to be determined. Customers often have their own preference for a type of contract even for the start of a project. I.e. they often prefer a fixed price contract. But this might not be the best applicable contract. Because the customer is the project initiator and the ultimate decision maker, it's preference for a form of contract influences the final contract greatly. This is why we will introduce a framework that allows the developer to determine a type of contract based on presented risks. With this framework he might be able to persuade a customer of a more fitting and fair contract. It should be kept in mind however, that a customer does not always have such a preference or that he will not reveal it.
In order to select a contract we first need to make a distinction between them. To make a distinction between the different types of contract we revisit the framework of Turner.
Looking back at the framework from Turner we see that it proposes three types of contracts for three quadrants. However, it also possesses a fourth quadrant which is left empty. Turner states that most of the software projects are likely to fall in this quadrant.

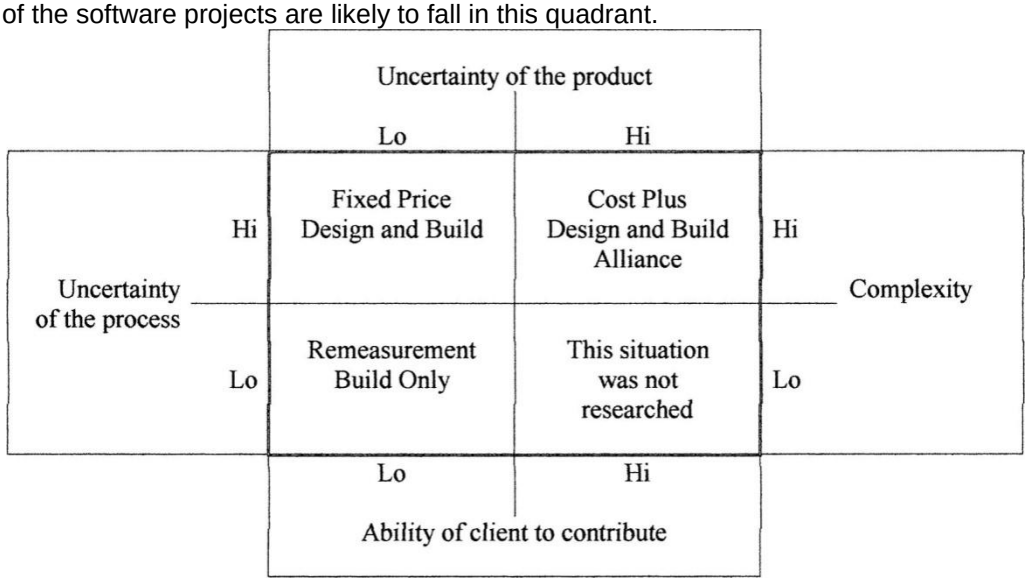


Figure 21: Four quadrants of Turner

While we do not dispute this, we do think that the other contract forms are viable for some software engineering projects.

Both re-measurement contracts as fixed price contracts can be applied as long as the customer is really sure what he wants and the developer knows what he is doing. This is because software can also exist out of standard packages or custom of the shelves products. While these will still need some configuration these are still standard products and thus will most often have a low uncertainty to their requirements. Thus if an application is specified upfront as a standard product a re-measurement contract or fixed price contract can be applied. This of course depends, if the developer is aware of the process how to deploy and what techniques to use. For example, a client wants a customer management system however he wants to use a different programming language then the developer is used to work with. This introduces an uncertainty to the process as the developer does not know how a development project with a different programming language goes. Hence that if the developer is unknown to the development process he cannot specify the rates and thus it is not recommended

to use a re-measurement contract then. What turner does not encompass in his model is that where he applies a re-measurement contract it would be perfectly possible to use a fixed price contract instead. Because all factors are known, it would thus be an easy task to specify the maximum costs in a fixed price contract. Another example would be if the developer would engage in a project that he has never done before. This would also increase the uncertainty about the development process.
The third quadrant has uncertainty for both the process and the product, meaning that the customer is not truly sure about his requirements and neither is the developer aware of how to undertake the development. If this is the case there are two options: either the customer finds a new developer or they cooperate under a Time and Material contract. The time and material contract will ensure that the customer will put all his effort into getting the requirements clear while the developer has all the resources he needs to gain insight into the development processes. This also entails that the developer will not be hold back. However, such a type of contract requires a lot of trust and according to turner, it requires that both parties have the availability and capability to undertake the project. Last Turner also states that the project must be risky otherwise it can best be managed by another type of contract.

Each of those three contracts can thus be applied in a software engineering contract, but most of the time a software developer will find himself in the fourth quadrant. Often customers have no clue of what it is that they exactly want or need, while the developer has a standard set of development rules to which he develops. Because of that, the fourth quadrant is the focus of this thesis.

4.7.1 Agile software development contract

The fourth quadrant has a certain process and an uncertain product. Thus it is known to both the developer and the customer how the development will take place but not what the application should look like. Thus the methods to tackle the problem are known (i.e. model driven development tools and methods) but the definition of this problem is not. Due to this nature, software projects can most of the times be seen as wicked problems. However, during a project a customer often learns from examples and input and which provides him the ability to create stable requirements. Thus over time the problem becomes less wicked. This is exactly what thought of agile development is about.
Many practitioners still believe that fixing all requirements is the best way to go. However this would not strive with the agile approach we suggest to use for model driven development. As the agile approach is based on the idea that requirements can and will change. But as identified in 4.3 constantly changing requirements can form a hazard to MDD projects as well as it will lead to a delay of development.

On the other hand, not fixing any requirements would provide the flexibility to the developer but would leave the customer in the dark about the actual costs of the application, assuming that he will sit out the project until he has a result that satisfies his needs rather than his wishes.

Thus the important thing to do is to find a way in between where some requirements are frozen while others can be bend. This is also what we proposed in the development method. As said in chapter 3.5 two aspects of a contract are that it can be seen as an agreement between two or more parties and as a setup of a project organization. Thus the contract is a guide to the finish of a project. The project square defines the dimensions among which a project manager can steer the project. Thus a contract must define what these dimensions are, or how they influence the project. The four dimensions are Time, Money, Scope and quality.

When looking at the contract types of time & materials and fixed price from the perspective of the project square one can conclude that the fixed price has everything fixed while times and materials has nothing fixed. *It should be noted that the dimension quality is dropped from the comparison. Quality in the project square stands for quality of the end product. We assume that any developer will fix quality at all time. If quality would not be fixed, it would be left to the developer whether or not the quality of an application is suitable which could leader to not working applications.

	Time	**Money**	**Scope**
Fixed price	Fixed	Fixed	Fixed
Time & materials	-	-	-

The disadvantage of having nothing fixed is obviously that the customer has no control on the total cost. While disbanding during the project is an option available to the customer this is the worst scenario for both parties. On the other hand the disadvantages of having everything fixed are that the developer has no vent to reduce the pressure on his resources if any changes occur in the requirements. This can be handled by introducing a risk fee that covers possible additional costs, but as customers are likely to claim more if they have to pay more this will end up in a vicious circle.

The solution to this is actually quite simple and already applied in some agile contracts. By not fixing all project dimensions but only allow two of the dimensions to be fixed the developer will have his steam vent. This provides the following three types of contracts:

	Time	**Money**	**Scope**
Time boxing	Fixed	Fixed	
Scope boxing	Fixed		Fixed
Scope budgeting		Fixed	Fixed

Time boxing
During a time boxing contract, a certain amount of money is agreed upon to be spend up to a time limit. However, how much is developed during this time can be variable. This would seem unfair to the customer as in theory the developer could develop nothing during this time period and still get paid. Thus in order to keep this contract attractive for both parties it is important to keep small iteration loops in which the customer can see the progress of the development. It should be noted that the feedback cycle of the contract is not the same as development iteration. However they are not mutually exclusive and could be merged for practical purposes.

Scope boxing
A scope boxing contract is based upon the idea that during a certain time period a certain scope has to be developed. The amount of resources used to develop this scope can vary. Thus increasing the resources increases the costs but will allow for a quicker development. This form of contract comes in handy when the customer wants the development to be done before a deadline. To ensure that the delivery of the application is not postponed both the contracting parties need to agree upon a deadline fee. If the developer cannot deliver the application at the given deadline he will have to pay that fee. How high that fee is should depend on the value of the application and on the days that the application is late.

Scope budgeting
Just as in a fixed price contract scope budgeting deals with a fixed scope for a fixed amount of money. However, the time that the project needs to be delivered can vary. While this form of contract might not seem viable in many situations, it could be that the developer is running low on capacity and thus needs to postpone the delivery of the project. While this might seem an unreliable form of contract the customer should realise that the developer does not like to postpone the project as it will also delay the payment of the project.

4.7.2 Choosing a contract type
In order to make a choice between the contract types we need to determine which contract type is most attractive for both the developer as the customer. To do this we can link the contract types with the risk via the steering mechanisms or project dimensions: Time, Scope and Money.

The risks that are identified in the risk profile can also be mapped to the project management dimensions, just like the contract forms. This will allow the developer to make grounded decisions about which contract form is the most suitable in a given situation. The project dimension that has the most risks associated with it should not be fixed. This will ensure that these risks can be reduced or changed by shifting the steering mechanism.

To realize this we need to link all risks to a steering mechanism. The following figure shows a list of all risks connected to a steering dimension. If the risk would occur it would result in a negative effect on that steering element. For example, bad facilities would cause development to go slower due to communication delays or because it is harder to develop due to a lack of a good network connection. This thus directly increases your development time and the chance to exceed the deadline. If the

chances are highest that the deadline will be exceed it would be wise not to set a deadline and thus not fix that dimension of the contract.

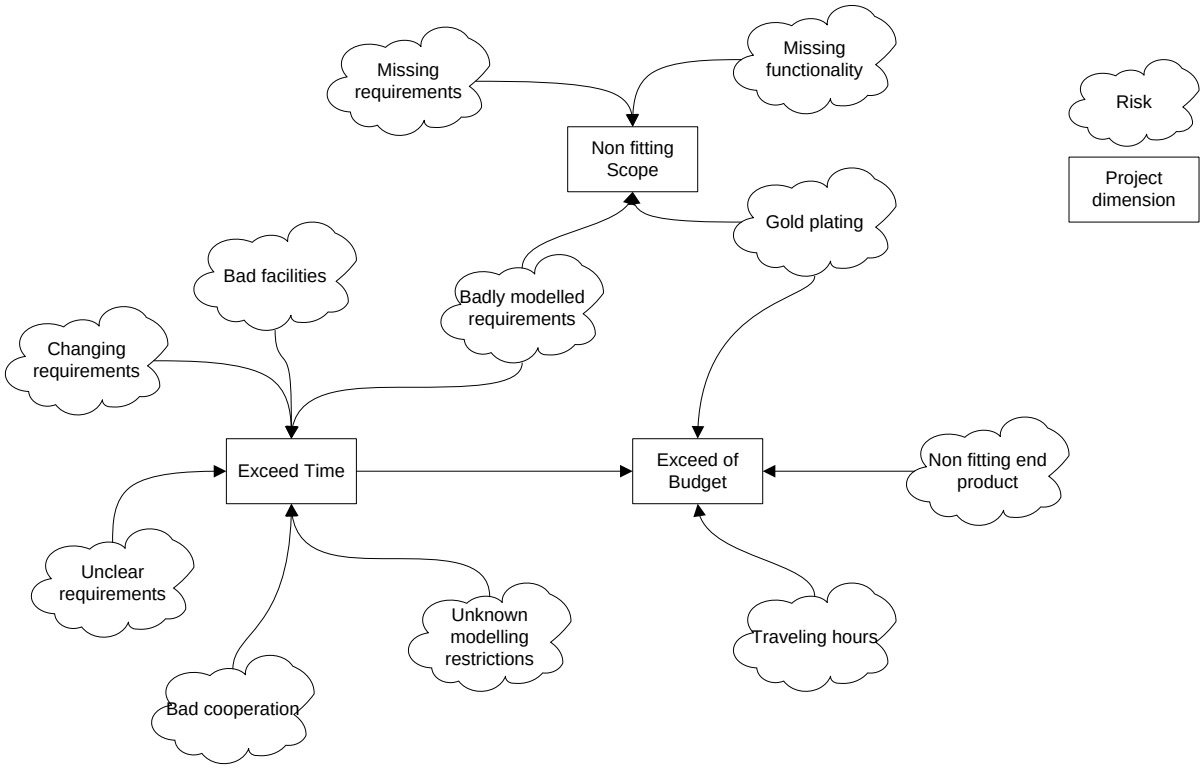


Figure 22: Risk mapping to project dimensions

4.7.3 Conclusion

The framework of Turner suggests 3 types of contracts spread over 3 quadrants. He explicitly stated that his framework was only viable for construction and that in because of large uncertainties about the problem software engineering required a different type of contract. To cross this gap we provided three new types of contracts Time boxing, Scope boxing and Scope budgeting. The new contract forms are based on the project dimensions scope, money and time. Each contract form fixes two out of the three dimensions. This will ensure that the customer will get what he wants while the developer has a vent to loosen the pressure when insecurity is at hand.

5. VALIDATION

In the previous chapter the framework to decide on how to fill in the development has been explored. Together with the model driven development method this framework should allow a developer to adapt the base development method to the characteristics of the customer. Also the framework suggests a method to determine what type of contract is best suitable in case of the customer and the development method. In this chapter the framework is validated on its usefulness in practice as on soundness and it's process

5.1 Internal and external validity

There are two types of validity, internal validity and external validity. [2] The external validity of a solution points to the extent to which the solution is valid under different circumstances.

According to Wieringa a solution is internally valid if "it is true that interaction among the solution elements and domain elements provide a certain outcomes" and that "The outcomes will take the stakeholders closer to their goal".

The internal validity thus points to the soundness of the solution where the external validity points out the applicability of the solution. In our case this means that our framework has to be found usable by developers at CAPE Group and it has to be found valid so that the method will produce a model driven development approach that will increase the efficiency of the project.

5.2 Validation approach

The goal of this research was to develop a framework that would prescribe a development method that was aligned with the customer and the contract, to increase efficiency. In order to align the customer with the development approach and the contract, the framework makes use of risk exposure. By determining how much risk a customer introduces to a project the developer can align his development method and propose a fitting contract.

To truly measure whether the risk exposure can be used to increase efficiency of model driven development projects multiple use cases or real projects would be required. However, due to a lack of time we choose for to validate the framework via illustration and an expert panel. These methods were merged into one large interview to save time. The interview can be found in APPENDIX D: INTERVIEW.

5.2.1 Validation through illustration

Validation through illustration is a technique that applies small examples and illustrations of the solution. Based on these examples a selected group can validated whether the results are wanted or correct. Also the usability can be checked by determining if the users are able to follow the steps taken in the solution and they understand how the solution is produced.

5.2.2 Validation through experts

Validation through an expert panel is comparable with the first validation method but differs in the fact that it is used to check the soundness of the solution. For this part in the validation the experts were asked to check if they could find any steps missing from the framework, as for the identified risks. Besides on completeness the experts were also questioned to see if all steps taken are logical.

The expert panel consisted of a group of 5, ranging from medior to senior software consultants who all have performed multiple model driven development projects. Validation on opinion is seen as less useful than a case study. However, due to deadline reasons we choose for the first.

5.3 Validation process

Due to a lack of time it was not possible to do two validation sessions with all experts. Thus both methods were merged into one large interview. To ensure that the experts understood the framework, a detailed explanation was given prior to the interview. Also during the interview an explanation was given of each input, process and output element of the framework. Questions about these elements were asked both before and after the explanation to also check the applicability and usefulness of the framework.

The interview consisted of 94 question, these questions can roughly be split into two types of questions: Yes/No questions and, explanatory questions. The yes/no questions were asked to confirm the usability soundness and logic of the process. If one of these was answered with an unexpected answer the interviewee was directed to an explanatory question.

In total there were 24 questions on the usability of the framework, 15 questions on the soundness of the framework and 10 on the logic of the framework. In total five interviews were held, all with experts from CAPE Groep.

5.4 Results

The interviews provided some new insights in both the usability as on the soundness of the framework as a whole. We will first describe the feedback in general before we will discuss the results in depth.

5.4.1 General results

In general all interviews provided good feedback on the usefulness as on the ease of use of the framework. Most experts did not understand the framework out of the blue, but after some small initial explanation they immediately understood the purpose of the framework. Furthermore, most experts also understood and agreed upon the process of the framework.

In terms of soundness the interviews showed varying results. Some experts stated that they just did not know whether the set of risks was complete. While others stated that they did not miss anything and that the most important risks are covered. But in overall

The applicability also showed some shortcomings, unlike the expectations the experts did not mind at all judging the risks with only some characteristics given to guide them. However, they did not fully understand how these risks lead to the actual changes in the development method. Also some did not understand how the development method was influenced by the project dimensions.

In overall they found the big advantage of the framework to be:

- Insight in the customer
- A guide line for the model driven development process
- A guide line to choose a contract type

One issue with the interviews itself was that due to the fact that each interview contained a lot of explanation about the framework itself they were very lengthy which resulted in a loss of concentration in the end.

5.4.2 In depth results

The interviews were structured around the different steps that exist in the framework. Were the first part checked the general idea and process of the framework. As said above, this was found to be useful and sound. The next part was the development method.

5.4.2.1 Development method

This part of the framework is the core of the framework as it is the main output that is presented to the developer. It is thus needs to be validated thoroughly on all three aspects.

The development method is based on existing developments such as scrum, XP and RUP. But rather than using 3 phases a fourth phase is included that tries to detect if any adaptions on the models or model transformations are needed. All experts deemed this a logical step but three of the five experts thought that this phase required an end point if there was chosen that the models were not updated and that the project was not done due to a lack of technical excellence to solve the problem.

This results in process as found in Figure 23.

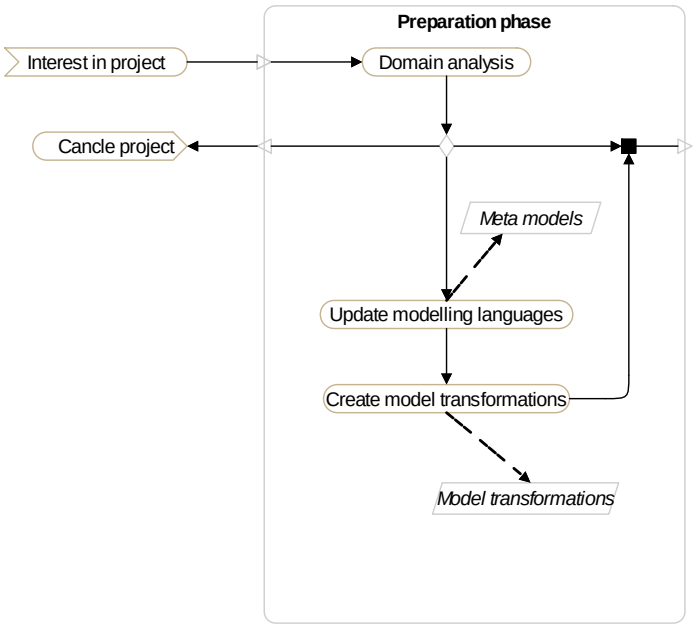


Figure 23: New preparation phase

Besides the preparation phase also four out of the five experts agreed that during the pre-development phase the requirements should be extracted before a development planning is made. Because the planning is based on the content that needs to made, the requirements are needed to plan the actual development. This creates the following process:

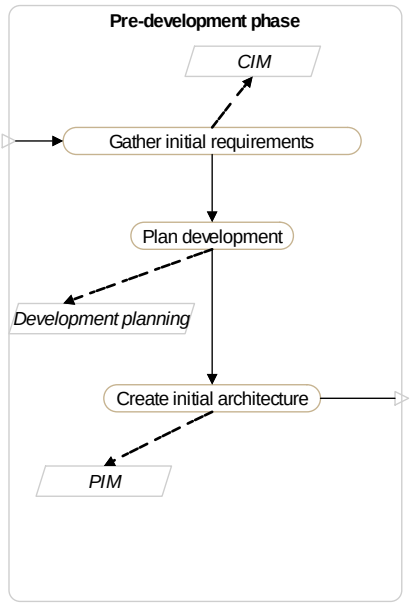


Figure 24: Improved pre-development phase

One expert also suggested renaming the phase to planning phase. However, because the phase not only plans but also builds architecture we kept the name.

Two experts also suggest that the current process around testing was not performed optimally because planning was not required after the system went live. Also they suggested that customers do normally not accept a unit but rather a whole deployment. Thus acceptation was put outside of the loop and system test in front of the customer acceptation. This allows the client to first test the system before they have to accept it and results in the following process.

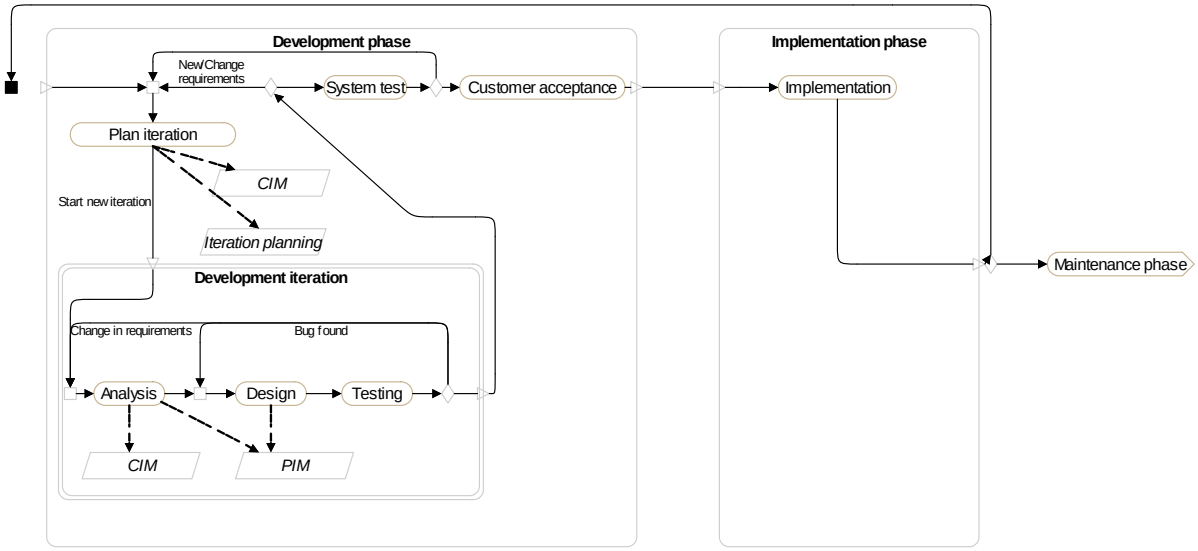


Figure 25: New development and implementation phase

All experts agreed upon all the deliverables of the development method. Only one suggested that the application that is developed also needed to be added.

The roles were also not complete according to 3 out of 5 experts. They all missed the business owner or process owner. A business owner is a customer role who is responsible for the requirements of a certain process. He is likely to provide the most requirements to the business analyst.

5.4.3 Customer profile

The experts were glad with the possibilities that the profile gave them but were not sure on how to apply it. Most of them found the profile complete enough but did suggest some improvements on the customer characteristics. They thought that amount of managers should be hierarchy and that culture required a maturity level. Two also thought that the amount of working hours that an employee is allowed to spend on a project should be part of culture.

While the customer profile will never be an exhausting profile of all the aspects of the customer that influence the development we do feel that these aspects are important and added them to the list. This results in the following changes to the customer profile

Customer properties	Customer characteristics
Organizational structure	• Size • Hierarchy
Organizational culture	• Working hours • Flexibility • Decision power • Available project hours

Table 13: Changes to customer profile

5.4.4 Risks

After the customer profile the risks and risk identification process were validated. Surprisingly most experts agreed that the most important risks were covered. One stated that he had trouble validating the completeness of the risks. However, because there are an unlimited amount of risks we do not worry nor state that this list is complete. As long as the most important risks to model driven development are covered.

The experts also agreed with the risk identification in which the risks are identified relatively of each other. Some even stated that this would be “the only way to make these kinds of estimates”. While we do think that there are other possibilities we have left our risk identification process as is due to its ease of use. However, we do see possibilities for future research.

Because there are multiple definitions of risk strategies we also validated this to see if they were found complete. Two of the five pointed out that at CAPE Groep they used a different model that has a 5th category of risk management called contingency risk management. However, one of those also stated that this could be seen as a reduction of the risk and was thus not that important to the framework.

5.4.5 Business cases

The business case provides a guideline to the developers to choose a risk management strategy. Most experts like the idea of using such a guide line and also thought it helpful. However, they also thought that these business cases were far from exhaustive and required some further research to determine what the other possible business cases can be. They did say that these were good enough to start with.

5.4.6 Remaining risks and contract types

The remaining risks are coupled to the project dimensions to be able to determine a type of contract. Most experts really liked this idea and thought that it was clear and even applicable. They were then asked if the contract types that are developed during this research are applicable and logical. All experts agreed upon these types of contracts applicable in software engineering. They even found it logical that quality is always fixed. One expert made the additional comment that this was “especially the case within CAPE Groep because CAPE strives to deliver the highest quality, with an exception on a proof of concept or a pilot project.”

5.5 Conclusion

We have validated the development method and the framework by interviewing five experts and providing them with examples and illustrations of the framework. This showed that even though the development method had some flaws in its process the framework itself is very useful in aligning the development method and the contract with the customer. It also showed that the customer profile is not exhausting and that more research is required to find a good way to determine the risks that the customer introduces.

6. CONCLUSION & RECOMMENDATIONS

In this final chapter all results of this research will be combined in a set of conclusions and recommendations. Paragraph 6.1 gives an overview of the conclusions that can be drawn from this research and tries to answer the research question that have been asked at the start of this research. This will be followed by the recommendations to users and future researchers in paragraph 6.2.

6.1 Conclusions

During this master thesis research has been conducted to aligning a customer in a model driven development project to the development method and the contract. The problem statement that was central to this research was the following:

> *"How can model driven software projects become more efficient by aligning development with the customer and contract type?"*

In order to answer this research question the four central questions have been answered that together resulted in the framework that has been developed.

What types of contracts are used in the software engineering field?
There are many different ways to define a contract. In this research we chose to define a contract as an agreement between two parties is used to set up a project organization. A contract has three main elements, a payment type, intellectual property and a product definition. Because in software engineering a product might not always be well defined from the beginning it is important to be able to adept a contract to certain changes. This can be done by mapping the contract to the project dimensions: Scope, Time and Money. In this research we defined three different types of contract that each fixed two out of the three dimensions. This provided the following types of contracts:

	Time	**Money**	**Scope**
Time boxing	Fixed	Fixed	
Scope boxing	Fixed		Fixed
Scope budgeting		Fixed	Fixed

Other contract types that can be applicable to software engineering but not in combination with our method are: Fixed price, Re-measurement and Time and Materials.

How does the customer influence software development projects?
To determine how the customer influences the software project we researched critical success factors that had to be present at the customer. These critical success factors were coupled to characteristics that on their own provide handles to the developer to identify a customer. The customer profile consists of the following properties and characteristics:

Customer property	**Customer characteristics**
Organizational structure	• Size • Hierarchy
Organizational culture	• Working hours • Flexibility • Decision power
Agile arrangements	• Facilities • Employee access • Network access
General IT experience	• Amount of similar IT projects done
Modelling experience	• Experience with software modelling
Teamwork	• Projects in daily operation
Resistance of groups or individuals	• Amount of users • Type of Users • Management support

It is notably that these characteristics are not made measureable and that these characteristics are not exhausting either. It was concluded during the validation that while they are neither exhausting nor measureable they are still useful in the process of identifying a customer.

How can a development method of MDD be aligned with the customer?
In order to align the customer with the development method first a base development method has been introduced. This development method has five parameters that can change between an agile approach and a more classical approach. These five parameters are:

Method parameters	Agile	Classic
Role distribution	Customer	Developer
Iteration time	1-2 weeks	2-4 weeks
Location	Customer	Developer
Requirements documentation	Few documentation	Full documentation

To map the development method with the customer, a risk exposure level can be used. The risk exposure level is a function of risk impact * risk chance. Thus for each parameter a set of risks has been identified which were mapped to the customer profile. This mapping was done based on the risks that were identified as results of failing the critical success factors that made up the customer profile. This mapping resulted in the following:

Customer property	Customer characteristics	Risks	Development parameter
Organizational structure	• Size • Hierarchy	Bad cooperation	Tester role, Business analyst role, Location
Organizational culture	• Working hours • Flexibility • Decision power	Bad cooperation	Tester role, Business analyst role, Location
Agile arrangements	• Facilities • Employee access • Network access	Bad facilities, Bad cooperation, large travel distance, Unclear requirements	Location, Business analyst
General IT experience	• Amount of similar IT projects done	Changing requirements, Unclear requirements, Missing requirements, Missing functionality, Gold plating, Unknown modelling restrictions, Non fitting end product	Iteration time, Initial requirements, Tester role, Business analyst
Modelling experience	• Experience with software modelling	Badly modelled requirements	Business analyst role
Teamwork	• Projects in daily operation	Bad cooperation	Tester role, Business analyst role, Location
Resistance of groups or individuals	• Amount of users • Type of Users • Management support	Refuse solution	Tester role

It should be noted that the risks that are identified are measured based on the characteristics that the customer profile has. However, because these characteristics are not made quantitative they are hard to measure.

How can the efficiency of software projects be measured?
To define how efficiency is used in software engineering we first have to look at the problem that was identified within CAPE Groep. We identified that customers often requested a non fitting contract form

and due to that more discussions arise. Efficiency can thus be defined as the time and money spend developing a certain scope. To measure the efficiency within a software project the project steering dimensions can be used. The more scope that has been made with the same amount of money and time the more efficient a project is.

6.2 Recommendations

In this paragraph recommendations will be given on two different levels. The first will be on the use of framework and development method. The second paragraph will contain recommendations regarding future research.

6.2.1 Use of framework

We performed an extensive research to develop a framework that allows the alignment of the development method with the contract and the customer. The main recommendations regarding the use of this framework are the following:

Train project managers in determining the risks, let them apply the framework first in a mirror project that runs parallel to a real project. This way they will not only learn the framework but are also able to validate the framework more extensively.

Create standard questionnaires that can be used during the initiation of the project to determine the risks. While it is hard to determine standard questionnaires due to large differences between customers, it might be possible to determine a type of customer or a type of market and create a questionnaire for this market.

Validate the framework more extensively by using the framework in upcoming projects and compare them to results of previous projects. This allows the efficiency increase to be measured based on the used resources in earlier project compared to the new projects.

Make use of the risk exposure levels not only to determine what the development method should be but also to persuade a customer into choosing the right contract type.

Make the project member aware of the risks involved in a project and enable a culture that talks about risks to make it easier to identify possible risks in the future. This does not stop at the developer, when the customer is more open about his limits it will be easier for both of them to determine risks. The more risks identified the better they can be managed.

6.2.2 Future research

This research has delivered a theoretical model that has been validated by 5 different experts. Based on their opinion the framework has been adapted to reflect a more realistic model. However as no real cased studies have been performed neither a lab demo it is too soon to conclude that the framework will always work. To further improve the framework, it will need to be put to use by practitioners. By applying it to real life examples and projects new risks can be found. The list of risks is far from exhausting and can be updated by use. Besides on the actual event of a risk future research should also focus on finding out why these risks occur and how they can be determined up front.

Another reason to further apply this framework is that it has only been validated by 5 experts who where all working for the same company. While not necessarily, it could be that information has been missed because of that. This is strengthened by the fact that the interviews all were very lengthy which led to a decrease in concentration of the experts. In overall more validation is necessary to assure the quality of this research.

One of the initial goals of this research was to increase efficiency by increasing the alignment between the customer, the contract and the development method. While the latter three have all been achieved within this research, no actual results about the increase of efficiency could be determined. This could either be because they are not present, but more likely it are the time limitations that prevent any results to show. Future research could focus itself on finding out if efficiency is indeed increased and if so, how much it increases.

Next to the efficiency increase also more studies are need to determine a good way to identify the risks more accurate. Currently the risks are only determined but if a better link could be made between the characteristics of the customer and the occurrence of a risk, the risks might actually be calculated which would provide a more reliable observation of the customer.

BIBIBLIOGRAPHY

[1] Mendix. (2009, *Mendix platform*. Available: http://www.mendix.com
[2] R. J. Wieringa, *Problem analysis and solution design chapter 6*. Enschede: University of Twente, 2008.
[3] Standish_Group. (2001, *Extreme Chaos*. Available: http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf
[4] H. Verschuren and P. Doorewaard, in *Het ontwerpen van een onderzoek*, ed Utrecht: Lemma, 1995, pp. 19-79.
[5] J. McCarthy, *Dynamics of software development*: Microsoft Press, 2006.
[6] P. Naur and B. Randell, "Software Engineering," Garmisch, 1968.
[7] W. S. Humphrey, "The Software Engineering Process: Definition and Scope," *ACM SIGSOFT Software Engineering Notes,* 1989.
[8] IEEE, "Standard glossary of software engineering terminology," ed, 1990.
[9] A. Macro and J. Baxton, *The craft of software engineering*: Addison-Wesley, 1989.
[10] H. v. Vliet, *Software engineering principles and practice*: Wiley, 2007.
[11] W. W. Royce, "Managing the development of large software systems," in *Proceedings IEEE WESCON*, 1970.
[12] B. Selic, "Model-Driven Development: Its Essence and Opportunities," presented at the Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2006.
[13] E. Seidewitz, "What Models Mean," *IEEE Software,* vol. 20, pp. 26-32, 2003.
[14] A. M. Starfield, "A Pragmatic Approach to Modeling for Wildlife Management," *Wildlife Management,* vol. 61, pp. 261-270, 1997.
[15] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer,* vol. 39, pp. 25-31, 2006.
[16] T. Mens and P. Van Gorp, "A Taxonomy of Model Transformation," *Electronic Notes in Theoretical Computer Science,* vol. 152, pp. 125-142, 2006.
[17] O. Pastor, *Model-driven architecture in practice : a software production environment based on conceptual modeling*, 2007.
[18] OMG, "Model driven Architecture Guide," in *The Basic Concepts*, ed: Object Management Group, 2003, p. 62.
[19] J. Bézivin, "In Search of a Basic Principle of Model Driven Engineering," *Novatica Journal,* 2004.
[20] B. Hailpern, "Model-driven development: The good, the bad, and the ugly," *IBM Systems Journal,* vol. 45, p. 451, 2006.
[21] S. Beydeda, *Model-Driven Software Development*, 2005.
[22] J. den Haan. (2009, October). From Process Design to Process Automation. *The enterprise architect*. Available: http://www.theenterprisearchitect.eu/
[23] CARE-Tech. (2009, *OlivaNova*. Available: http://www.care-t.com/
[24] M. Jackson, "Software development method," in *A classical mind: essays in honour of CAR Hoare*, A. W. Roscoe, Ed., ed: Prentice Hall, 1994, pp. 211-230.
[25] A. Qumer and B. Henderson-Sellers, "An evaluation of the degree of agility in six agile methods and its applicability for method engineering," *Information & Software Technology,* pp. 280-295, 2008.
[26] W. Cunningham, "Manifesto for agile software development," in *Manifesto for Agile Software Development*, ed, 2001.
[27] J. Warsta. (2002, *Agile Development Methods: Review and Analysis*.
[28] P. Abrahamsson, Salo, O., Ronkainen, J., Warsta, J., "Agile software development methods review and analysis," *VTT Publications,* vol. 478, pp. 3-178, 2002.
[29] "Allwords," in *Allwords*, ed, 2009.
[30] S. Whang, "Contracting for Software Development," *Management Science,* pp. 307-324, 1992.
[31] J. R. Turner and J. S. Simister, "Project contract management and a theory of organization," *International journal of project management,* pp. 457-464, 2001.
[32] J. R. Turner and R. A. Cochrane, "Goals-and-methods matrix: coping with projects with ill defined goals and/or methods of achieving them," *International journal of project management,* vol. 11, pp. 93-102, 1993.

[33] W. B. Barry. (1991) Software Risk Management: Principles and Practices. 32-41. Available: http://doi.ieeecomputersociety.org/10.1109/52.62930
[34] B. Boehm, "Software risk management," ed, 1989, pp. 1-19.
[35] D. v. Well-Stam, Lindenaar, F., Kinderen, S. van, , *Risicomanagement voor projecten; De RISMAN-methode toegepast.* . Utrecht. ISBN 2003.
[36] P. Berander and A. Andrews, "Requirements Prioritization," ed, 2005, pp. 69-94.
[37] J. Galbreath, "Customer relationship leadership: a leadership and motivation model for the twenty-first century business," *The TQM magazine,* vol. 11, p. 161, 1999.
[38] P. Schubert and P. Risch, "Customer profiles, personalization and privacy.," *In Proceedings of CollECTeR Europe,* 2005.
[39] T. Chow and D. Cao, "A survey study of critical success factors in agile software projects," *The Journal of Systems and Software,* pp. 961–971, 2008.
[40] R. Fisher, *Getting to yes*, 1981.
[41] A. Gopal, "Contracts in Offshore Software Development:An Empirical Analysis," *Management Science,* 2002.
[42] B. Whittaker, "What went wrong? Unsuccessful information technology projects," *Information Management & Computer Security,* vol. 7, pp. 23-30, 1999.
[43] S. Nerur*, et al.*, "Challenges of migrating to agile methodologies," *Commun. ACM,* vol. 48, pp. 72-78, 2005.
[44] A. Cockburn, "Agile software development, the people factor," *Computer,* vol. 34, p. 131, 2001.
[45] E. B. Dent and S. G. Goldberg, "Challenging "Resistance to Change"," *Journal of Applied Behavioral Science,* vol. 35, pp. 25-41, March 1, 1999 1999.
[46] B. Selic, "The pragmatics of model driven development," *IEEE Software* vol. 20, pp. 19-25, 2003.
[47] J. J. Jiang, "A measure of software development risk," *Project Management Journal,* vol. 33, p. 30, 2002.
[48] L. Wallace and M. Keil, "Software project risks and their effect on outcomes," *Commun. ACM,* vol. 47, pp. 68-73, 2004.
[49] A. M. Davis, *Software requirements: analysis and specification*: Prentice Hall Press, 1990.

APPENDICES

Appendix A: Software manifesto principles
Appendix B: Agile methods
Appendix C: Critical success factors in agile projects

APPENDIX A: SOFTWARE MANIFESTO PRINCIPLES

Our highest priority is to satisfy the customer
through early and continuous delivery
of valuable software.

Welcome changing requirements, even late in
development. Agile processes harness change for
the customer's competitive advantage.

Deliver working software frequently, from a
couple of weeks to a couple of months, with a
preference to the shorter timescale.

Business people and developers must work
together daily throughout the project.

Build projects around motivated individuals.
Give them the environment and support they need,
and trust them to get the job done.

The most efficient and effective method of
conveying information to and within a development
team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development.
The sponsors, developers, and users should be able
to maintain a constant pace indefinitely.

Continuous attention to technical excellence
and good design enhances agility.

Simplicity--the art of maximizing the amount
of work not done--is essential.

The best architectures, requirements, and designs
emerge from self-organizing teams.

At regular intervals, the team reflects on how
to become more effective, then tunes and adjusts
its behaviour accordingly.

Appendix B: Agile methods

Method	Key points	Special features	Identified Shortcomings
ASD	Adaptive culture, collaboration, mission-driven component based iterative development	Organizations are seen as adaptive systems. Creating an emergent order out of a web of interconnected individuals.	ASD is more about concepts and culture than the software practice.
AM	Applying agile principles to modelling: Agile culture, work organization to support communication, simplicity.	Agile thinking applies to modelling also.	This is a good add-on philosophy for modelling professionals. However, it only works within other methods.
Crystal	Family of methods. Each has the same underlying core values and principles. Techniques, roles, tools and standards vary.	Method design principles. Ability to select the most suitable method based on project size and criticality	Too early to estimate: Only two of four suggested methods exist.
DSDM	Application of controls to RAD, use of time boxing, empowered DSDM teams, active consortium to steer the method development.	First truly agile software development method, use of prototyping, several user roles: “ambassador”, “visionary” and “advisor”.	While the method is available, only consortium members have access to white papers dealing with the actual use of the method.
XP	Customer driven development, small teams, daily builds	Refactoring – the ongoing redesign of the system to improve its performance and responsiveness to change.	While individual practices are suitable for many situations, overall view & management practices are given less attention.
FDD	Five-step process, object-oriented component (i.e., feature) based development. Very short iterations: from hours to 2 weeks.	Method simplicity, design and implement the system by features, object modelling.	FDD focuses only on design and implementation. Needs other supporting approaches.
OSS	Volunteer based, distributed development, often the problem domain is more of a challenge than a commercial undertaking.	Licensing practice; source code freely available to all parties.	OSS is not a method itself; ability to transform the OSS community principles to commercial software development.
PP	Emphasis on pragmatism, theory of programming is of less importance, high level of automation in all aspects of programming.	Concrete and empirically validated tips and hints, i.e., a pragmatic approach to software development.	PP focuses on important individual practices. However, it is not a method through which a system can be developed.
RUP	Complete SW development model including tool support. Activity driven role assignment.	Business modelling, tool family support.	RUP has no limitations in the scope of use. A description how to tailor, in specific, to changing needs is missing.
SCRUM	Independent, small, self-	Enforce a paradigm shift	While Scrum details in

	organizing development teams, 30-day release cycles.	from the “defined and repeatable” to the “new product development view of Scrum.”	specific how to manage the 30-day release cycle, the integration and acceptance tests are not detailed.

Appendix C: Critical success factors in agile projects

Failure factors
Organizational
1. Lack of executive sponsorship
2. Lack of management commitment
3. Organizational culture too traditional
4. Organizational culture too political
5. Organizational size too large
6. Lack of agile logistical arrangements
People
7. Lack of necessary skill
8. Lack of project management competence
9. Lack of team work
10. Resistance from groups or individuals
11. Bad customer relationship
Process
12. Ill-defined project scope
13. Ill-defined project requirements
14. Ill-defined project planning
15. Lack of agile progress tracking mechanism
16. Lack of customer presence
17. Ill-defined customer role
Technical
18. Lack of complete set of correct agile practices
19. Inappropriateness of technology and tools
Success factors
Organizational
1. Strong executive support
2. Committed sponsor or manager
3. Cooperative organizational culture instead of hierarchal
4. Oral culture placing high value on face-to-face communication
5. Organizations where agile methodology is universally accepted
6. Collocation of the whole team
7. Facility with proper agile-style work environment
8. Reward system appropriate for agile
People
9. Team members with high competence and expertise
10. Team members with great motivation
11. Managers knowledgeable in agile process
12. Managers who have light-touch or adaptive management style
13. Coherent, self-organizing teamwork
14. Good customer relationship
Process
15. Following agile-oriented requirement management process
16. Following agile-oriented project management process
17. Following agile-oriented configuration management process
18. Strong communication focus with daily face-to-face meetings
19. Honoring regular working schedule – no overtime
20. Strong customer commitment and presence
21. Customer having full authority
Technical
22. Well-defined coding standards up front

23. Pursuing simple design
24. Rigorous refactoring activities
25. Right amount of documentation
26. Regular delivery of software
27. Delivering most important features first
28. Correct integration testing
29. Appropriate technical training to team
Project
30. Project nature being non-life-critical
31. Project type being of variable scope with emergent requirement
32. Projects with dynamic, accelerated schedule
33. Projects with small team
34. Projects with no multiple independent teams
35. Projects with up-front cost evaluation done
36. Projects with up-front risk analysis done

Appendix D: Interview

Onderdeel	Vraag	Antwoord	Aantekeningen
Algemeen			
	Is het framework zonder introductie duidelijk?		
	Mist er op het eerste oog iets in het framework?		
	Zo ja, wat dan?		
	Is het framework na korte uitleg nog steeds duidelijk?		
	Zo nee, wat niet?		
	Volgt het framework een logisch process of zijn er stappen die je anders zou doen?		
	Zo nee, wat is er niet logisch en waarom niet?		
1. Base development method			
	Is het doel van de stap basis ontwikkel methode duidelijk?		
	Zo niet wat niet en hoe zou dit duidelijker kunnen?		
1A. Fase plan			
	Is het doel van de **Preparation** fase duidelijk?		
	Zijn de activiteiten duidelijk in de preparation fase?		

	Zo niet wat niet en hoe zou dit duidelijker kunnen?		
	Is de volgorde van de activiteiten logisch?		
	Zo nee waarom niet?		
	Missen er nog activiteiten?		
	Zo ja welke ontbreken er?		
	Is het doel van de **Pre-development** fase duidelijk?		
	Zijn de activiteiten duidelijk in de Pre-development fase?		
	Zo niet wat niet en hoe zou dit duidelijker kunnen?		
	Is de volgorde van de activiteiten logisch?		
	Zo nee waarom niet?		
	Missen er nog activiteiten?		
	Zo ja welke ontbreken er?		
	Is het doel van de **Development** fase duidelijk?		
	Zijn de activiteiten duidelijk in de Development fase?		
	Zo niet wat niet en hoe zou dit duidelijker kunnen?		
	Is de volgorde van de activiteiten logisch?		
	Zo nee waarom niet?		
	Missen er nog activiteiten?		
	Zo ja welke ontbreken er?		
	Is het doel van de **Implentation** fase duidelijk?		

	Zijn de activiteiten duidelijk in de implementatie fase?		
	Zo niet wat niet en hoe zou dit duidelijker kunnen?		
	Is de volgorde van de activiteiten logisch?		
	Zo nee waarom niet?		
	Missen er nog activiteiten?		
	Zo ja welke ontbreken er?		
	Ontbreken er in de gehele methode activiteiten die gedurende een MDD traject plaats moeten vinden?		
	Zo ja, welke stappen ontbreken er en waarom?		
1B. Rollen			
	Is het van iedere rol duidelijk wat zijn of haar verantwoordelijkheden/taken zijn?		
	Zo nee, van welke rol niet?		
	Zijn deze rollen dekkend of ontbreken er nog rollen of verantwooredelijkheden?		
	Zo Nee, wat ontbreekt er?		
1C. Deliverables			
	Is het van iedere deliverable duidelijk wat deze precies inhoudt?		
	Zo nee, welke is onduidelijk en waarom?		

	Missen er nog deliverables of onderdelen die gedurende het ontwikkel traject opgeleverd dienen te worden?		
	Zo ja welke ontbreken er en waarom?		
2. Customer identification			
	Is het doel van de deze stap duidelijk?		
	Zo nee, waarom niet?		
	Is het duidelijk waarom er gebruik is gemaakt van deze klant eigenschappen?		
	Is de keuze voor deze klant eigenschappen logisch?		
	Zo nee, waarom niet?		
	Ontbreken er nog klant eigenschappen?		
	Zo ja, welke eigenschappen ontbreken en waarom?		
	Zijn de gekoppelde klant karakteristieken duidelijk?		
	Zo nee, welke niet?		
	Zijn de gekoppelde klant karakteristieken dekkend?		
	Zo nee, welke ontbreken er?		
3. Risk identification			

	Is het doel van de deze stap duidelijk?		
	Zo nee, waarom niet?		
	Zijn de Risico's dekkend genoeg of ontbreken hier nog belangrijke risico's?		
	Zo ja welke ontbreken er en waarom?		
	Is het duidelijk hoe deze risico's beoordeeld moeten worden?		
	Zo nee, wat ontbreekt hieraan en waarom?		
4. Business case			
	Is deze stap logisch en benodigd?		
	Zo ja waarom is deze stap belangrijk?		
5. Risk management strategy			
	Is het doel van deze stap duidelijk?		
	Zo nee, waarom niet, wat was er verwacht?		
	Zijn de vier type risico management strategien duidelijk?		
	Zo nee, welke niet en waarom niet?		
	Zijn de vier type risico management strategien compleet of ontbreekt er hier nog een?		
	Zo nee, welke ontbreekt en waarom moet deze er bij?		
6. Adaption base method			

	Is de manier waarop deze stap wordt uitgevoerd duidelijk?		
	Zou je de stap op de zelfde manier uitvoeren?		
	Zo nee hoe zou de stap anders uitgevoerd kunnen worden?		
	Is de mogelijke aanpassing voor ieder risico duidelijk?		
	Zo nee welke niet?		
	Zijn de mogelijke aanpassingen voor ieder risico logisch?		
	Zo nee welke niet?		
	Zijn de mogelijke aanpassingen compleet?		
	Zo nee, welke ontbreken er hier en waarom?		
7. Remaining risks			
	Is deze output duidelijk?		
	De risico's worden hier uitgezet tegen de project management variabelen is deze stap logischerwijs te volgen?		
	Zo niet waarom niet en hoe zou dit anders kunnen?		
8. Fixed project dimensions			
	Met het bepalen van de project dimensies wordt het contract nader ingevuld, is dit duidelijk en logisch?		

	Zo nee, waarom niet?		
	Het bepalen van de project dimensies wordt gedaan aan de hand van de overgebleven risico's. Deze worden opgeteld en degene die het meeste risico overheeft heeft de minste voorkeur om gefixed te worden. Is dit een juiste benadering of kan dit niet zomaar gesteld worden?		
	Zo nee, waarom niet?		
	Kwaliteit wordt altijd gefixed, is dit een juiste stelling?		
9. Afsluiting			
	Ontbreekt er iets aan dit framework		
	Zo ja, wat ontbreekt er nog?		
	Heb je nog vragen gemist in het interview?		
	Zo ja, welke?		