BACHELOR THESIS

Elicitation and analysis of current approaches to adopt Agile Development methodologies for mechanical and mechatronic engineering

Fakultät für Wirtschaftswissenschaften

des Karlsruher Instituts für Technologie (KIT)

from

Andrea Figueras Vall

[Matriculation Nr.1797482]

Delivery Day: 09/07/2015

Expert: Prof. Dr. Dr.-Ing. Dr. h. c. Jivka Ovtcharova

First adviser: Dipl.-Wi.-Ing. Matthes Elstermann

Second adviser: Dipl.-Ing. Hendrik Schuck

Karlsruher Institut für Technologie

IMI Informationsmanagement im Ingenieurwesen
Prof. Dr. Dr.-Ing. Dr. h. c. Jivka Ovtcharova

Kaiserstraße 12 | Zirkel 2
76131 Karlsruhe | Geb. 20.20

Betreuerer: Dipl.Wi.-Ing. M. Elstermann

Telefon: 0721 608- 46635
Fax: 0721 608- 43984
E-Mail: matthes.elstermann@kit.edu
Web: http://www.imi.kit.edu

Unser Zeichen: ME
Datum: 8. July 2015

Bachelor Thesis

Topic:

Elicitation and analysis of current approaches to adopt Agile Development methodologies for mechanical and mechatronic engineering

Initial Situation:

In classical as well as software engineering there are many different development methodologies and approaches. Many ideas or base concepts are almost the same, like waterfall-concepts or the V-model. Yet, in Software Engineering there has been a move away from those classical approaches, despite the fact that these models have been used for longer time. For various reasons, that will be research in the thesis, there has been a move towards so-called agile development approaches to govern development projects. As these agile concepts have been applied with quite some success, nowadays many companies use Agile Development methodologies to develop their software. Due to their apparent success, the use of agile concepts has sparsely been spread to other domains, but it is unknown to what extend and with how much success.

Goal/Proceeding:

The main goal of this thesis is to find, describe and analyze sources of recent researches or case studies and examples about adopting agile approaches to manage development beyond software engineering, if any such sources exist. Based on the findings it will be tried to analyze whether Agile Development methodologies can or should be applied to the domain of mechanics or mechatronics. In order to do so, it is necessary to research and understand what Agile Development actually is, the main reasons behind its usage, and how agile methods compared to more classical development approaches and their purpose.

Prof. Dr. Dr.-Ing. Dr. h. c. Jivka Ovtcharova

Declaration

This dissertation is the result of my own work and includes nothing, which is the outcome of work done in collaboration except where specifically indicated in the text. It has not been previously submitted, in part o whole, to any university of institution for any degree, diploma, or other qualification.

_____ _____

Place, Date Signature

Abstract

The purpose of this study was to conclude if it would be possible to use Agile Development methodologies with mechanical and mechatronic engineering.

Agile values and practices have revolutionized the software development industry in the last decade, and therefore it was of interest to figure out if other engineering disciplines could also take profit of Agile. The research has been mainly focused on finding literature and also finding some case studies about how to adapt agile approaches to mechanic and mechatronic products. The objective was to conclude if it was possible, and in that case, which adaptations were needed to in order to fit mechanical and mechatronic engineering.

The literature that has been found about the concerned issue is all based on experiences, commentaries and interviews of users, due to the lack of other available or existing sources. Specifically, most of the founded experiences are referred to hardware or mechanical development using Scrum.

The results of the research indicate that it is possible for mechanical development team to use agile approaches, but they require some adaptations. Moreover, during the research some advices have been found, which might be useful for starting applying Agile with mechanical or mechatronic engineering.

Index

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Both in business and in engineering, product development is the whole process of creating a new product. Ulrich and Krishnan in 2001 described “new product development as the transformation of a market opportunity into a product”.

Product development is a complex process, which has been an issue of improvement since the beginning of the engineering. Each type of engineering needs different types of approaches in order to develop their products. In this thesis the concerned disciplines are mechanical, mechatronic and software engineering.

In the beginning of the 21st century, a new way of managing software development came up and revolutionized the way of developing software. In 2001, a group of developers formed the Agile Manifesto and defined the nowadays known Agile Software Development methodology. These methods, which are basically based on self-organized cross-functional team, rapid response to changes and are goal-orientated, have succeeded in software industries. Many large companies use agile approach manly because agility can contribute to decreasing the development time for new processes and increasing flexibility for existing processes, where modification and implementation are required.

In a complex and permanently changing environment, organising the project with agility is no longer a need but a condition if the companies want to remain or even access into the market. For this reason, mechanical and mechatronic teams are also willing to take profit of Agile Development methodology.

1.2. Aims and objectives

By the moment it seems that agile approaches are only applicable with software development. The purpose of this thesis is to conclude if agile approaches can also be applied to other engineering disciplines, beyond software development, focusing preferably on the domain of mechanics and mechatronics.

The main goal is to find, describe and analyse sources of recent researches or case studies about adopting agile approaches in non-Software environments. However, the available literature about non-Software environments using Agile methods is very

limited. This fact makes the research more complex, and for this reason the main information sources will be experienced engineers who have tried to apply agile approaches to their projects.

1.3. Methodology and limitations

The methodology used to reach the goal of the thesis is based first on consulting all the known literature where agile approaches are concerned, such as conferences, books, newspapers, etc.

However, after revising the literature, these sources did not contain much information about the concerned issue. For this reason the sources are mainly blogs, forums and other public groups available in internet, where people can express their thoughts and opinions about different topics.

The thesis is limited for its duration of approximately three months, but moreover it is limited for the mentioned lack of resources and information to find companies which are using agile approaches in non-software projects.

1.4. Structure

In order to analyse whether Agile Development methodologies can or should be applied to the domain of mechanics or mechatronics, before it is necessary to research and understand what Agile development actually is.

Firstly the thesis presents a short introduction of what software development is and which are the more classical development approaches and their purpose. This introduction will help to understand why there was a need of a new software development methodology, and how these methods can be compared and differentiate of agile methods.

In order to reach the purpose of the thesis, Agile Software Development methods are properly explained, as well as the main types of these methods. Once agile values have been understood, there is an analysis of how to adapt and modify these methods in order to suit mechanic and mechatronic engineering.

The research presents two different ways for trying to conclude how to adapt agile approaches. The first one is the study of different commentaries and theories about how agile could success with mainly hardware engineering, all them based on the own experience of agile users. The second way is the review of some case studies of companies which have already used Scrum with mechanical projects. These cases apart from allowing us to extract some conclusions can also help to understand better agile methods.

2. Development methodologies in engineering

2.1. Introduction

Back to 2500 years BC, large projects must already have had some sort of development methodologies, such the Pyramid of Giza or Pyramid of Cheops. It is difficult to believe that those types of projects were possible without some sort of planning and project management. (Stober & Hansmann, 2010)

The evolution of the machines and the associate techniques of design made more remarkable the need of some methods which could help to develop faster the projects and with a lower rate of failure.

Nowadays, it is common to distinguish different methods or skills inside the field of product development process. These methods allow solving specific problems in engineering, especially when there are disciplines that affect more than others. However, each type of engineering is very particular; they have their own characteristics such as specific development, design and implementation process. For this reason, each domain of engineering has its own development methodologies which guide projects through the way of success.

Agile Software Development, as well as many other methods, are examples of these development methodologies in the field of Software engineering.

2.2. Software development methodologies

Software development models can be described as various processes or methodologies, which have been selected and organized in order to develop the project according to its objectives and functionality. Software development processes help to organize properly the development process, and strengthen the software quality.

All software development models follow the general phases of the Systems Development Life Cycle (SDLC). SDLC is the term used in software engineering to describe the activities performed in each stage of the software development process. This cycle basically describes how the project is going to be developed and maintained. The *Figure 1* shows the mentioned general phases that all the models have to follow for the purpose of success.

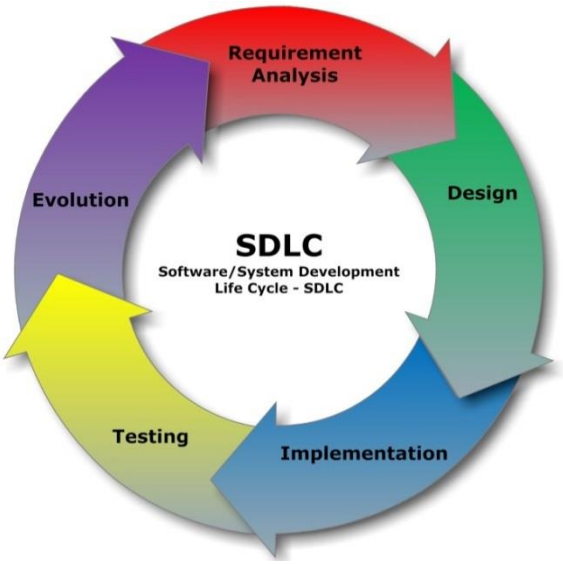

Figure 1. System Development Life Cycle

There are several models for the software development life-cycle; besides software development is also a field that is constantly researching for new methods. Indeed the last 25 years, a large number of different approaches have been developed but few of them are nowadays being used. (Abrahamsson et al., 2002)

Each model has been developed to achieve certain objectives. The most common and traditional software development models are Waterfall, Spiral, Iterative, V-shaped and Prototyping. These methods are following briefly explained, just for understanding their operating method and the main differences between them. (Mohammed & Govardhan, 2010)

2.2.1. Spiral model

The spiral model is a combination of top-down and bottom-up concepts which emphasizes risk analysis, especially with large complex systems. The spiral model has four phases: *determining the iteration, evaluating alternatives and risking analysis, developing and verifying deliverable,* and *planning the next iteration*.

The model is called Spiral due to the fact that the project repeatedly passes through these phases in various iterations.The software project enters into the next iteration, which is based on the customer's evaluation, and follows a lineal approach to implement the feedback suggested by the customer. The structure can be understood in the *Figure 2*.

It is said that Spiral model emphasizes risk assessment and minimizes project risk because it focuses on breaking a project into small segments. This way the provided segments are easier to change during the development process, and they also provide the opportunity to evaluate risks throughout the life cycle.

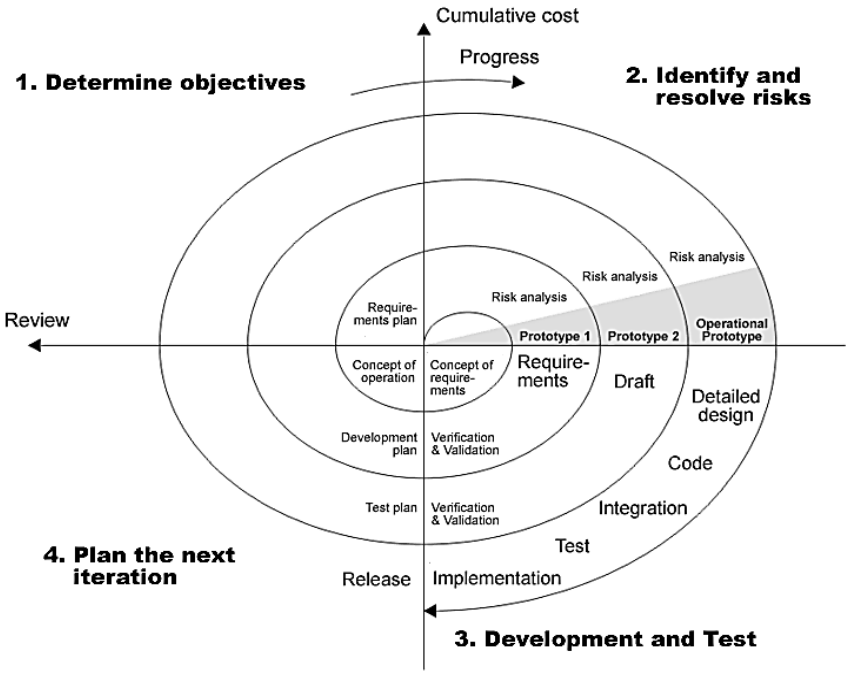

Figure 2. Spiral model process

2.2.2. Iterative development

The Iteration model was created as a variation of the Waterfall model due to the problems that this has. Iteration model usually consists in the division of the project into small parts, as observed in *Figure 3*; this way the demonstration of a result is allowed earlier. Each part of the project is developed in a different iteration, and each iteration is actually a mini- Waterfall process.

The main purpose of iterative development is to allow flexibility for later allowing changes. Then, the project team can evaluate within each iteration, which changes are needed in order to produce a satisfactory product.

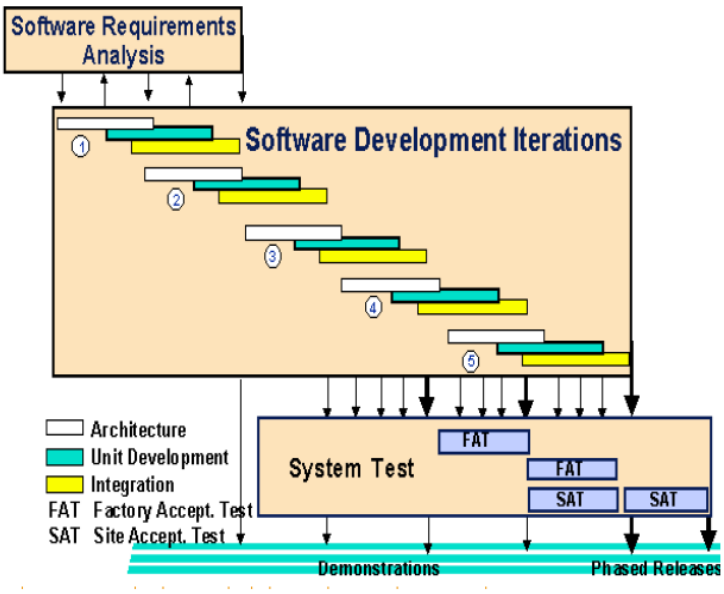

Figure 3. Iterative Development

This model is usually combined with the Incremental built model for software development, creating this way the **Iterative and Incremental development.** Incremental development slices the system functionality into increments (portions). In each increment, a slice of functionality is delivered through cross-discipline work, from the requirements to the deployment. (Wikipedia)

Iterative and incremental are keywords for Rational Unified Process, Extreme Programming and generally for various agile software development frameworks.

2.2.3. V-shaped model

The V-shaped model is an extension of the waterfall model. The typical waterfall moves lineally downwards, meanwhile V-shaped model phases are turned upwards after coding phase to form the V shape. As in waterfall model, the life cycle model is begun with the requirements and each phase must be completed before moving forward.

The biggest difference between waterfall and V-shaped model is the emphasis that V-shaped has in testing. The testing procedures are developed early in the life cycle before any coding is done, during each of the phases preceding implementation. The V-Model shows the relationship between each phase of the development life cycle and its associated phase of testing.

In the *Figure 4,* the main structure of this model can be appreciated, where the horizontal axe represents time whereas the vertical axe represents the level of abstraction.

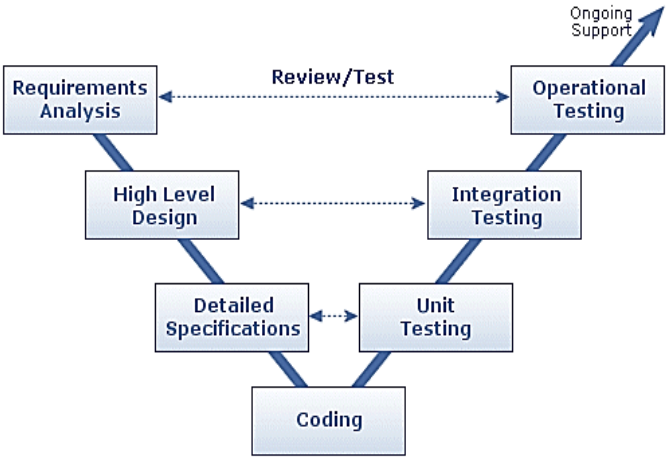

Figure 4. V shaped model

2.2.4. Prototyping

Software prototyping is the creation of prototypes of software applications. The main idea is to freeze a code or design before it can proceed, this way the requirements can be better understood. Prototyping is an attractive idea for complex systems for which the requirements are difficult to understand or determinate. The prototype is usually an incomplete system, but it provides an overall view, both to the project team and the client, of how the system will work. It permits to detect errors earlier and identify easily missing functionalities. (ISTQB exam certification)

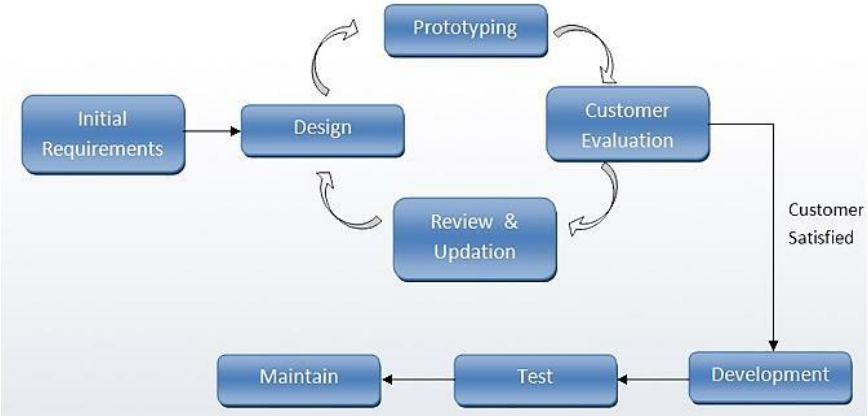

Figure 5. Prototyping cycle

2.2.5. Waterfall methodology

The waterfall model is one of the most popular and oldest system development life cycles for software engineering, and is widely used in government projects and in many major companies. As it is the most generic and used method, this methodology is explained more deeply than the others.

The first formal description of the waterfall model is often cited as an article by Winston W. Royce in 1970. Royce presented this model as “an example of a flawed, non-working model”. (Royce, 1970)

The waterfall model describes a development method which proceeds sequentially through different phases. Each waterfall stage or phase is assigned to a different team in order to control the deadline and to obtain a better project. As this model emphasizes planning in early stages, it ensures design flaws before they are developed (Mohammed & Govardhan, 2010). Feedback loops exist between each phase, like this, it is possible to start again a phase and make the appropriate modification. We can say that “the progress flows from one stage to the next, much like the waterfall that gives the model its name”. (Contributor, 2006)

Waterfall model is a *rigid and linear method*, which means that waterfall development has distinct goals for each phase of development and where each phase has to be completed before the next one is started. Once the next phase is started, there is no turn back to the previous phase.

The main goal of waterfall methodology, besides getting a great project, is to *deliver the project on-time*, this way man can say that the classic waterfall method stands for predictability.

In order to understand well the waterfall methodology it is necessary to explain the distinct phases that the model presents: (Stober & Hansmann, 2010) (Contributor, 2006)

1. ***System requirements***: This is the first step of the method and is also the most important, as it involves the description of the software which is going to be developed. This description has to establish the components for building the system, laying out functional and non-functional requirements. These requirements are predicated on understanding the customer’s business context and the functions the product must perform. The results of the analysis are

typically captured in formal requirements specification, which are used as input to the next step.

2. ***Design***: This step consists in defining the hardware and software architecture, specifying performance and security parameters, designing data storage containers and constraints, choosing the IDE and programming language, and indicating strategies to satisfy the specified requirements.
3. ***Implementation***: In this phase the product is built based on the design specifications developed in the previous step. The design documents are translated into code, component by component, built according to a pre-defined coding standard and debugged. Before they are integrated to satisfy the system architecture requirements, each component is tested and reviewed on its own.
4. ***Testing*****:** The goal of this phase is to identify bugs in the software by executing the program before it is released to the costumer. In this stage, both individual and integrated components are tested to ensure that they are free of errors and fulfil the requirement. Defects, if found, are logged and feedback is provided to the implementation team to enable correction. This is also the stage where product documentation is prepared, reviewed and published.
5. ***Maintenance*****:** This phase starts as soon as the solution is delivered to the costumer or the main user. It involves making modifications to the system or to an individual component, modifying attributes or improving performance. These modifications come up either due requests of the customer, or defects uncovered during live use of the system.

Its structure can be observed in the *Figure 6,* as well as its phases.

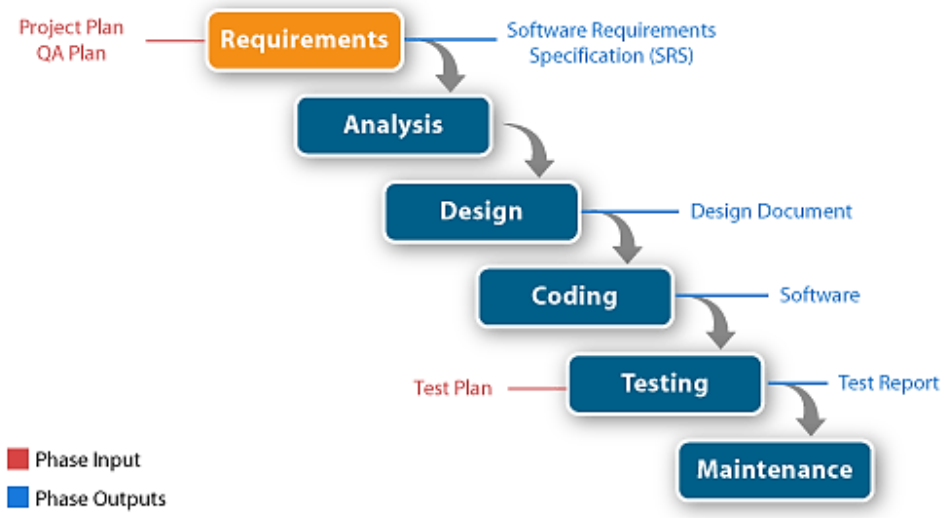

Figure 6. Waterfall methodology

Once it has become clear what the waterfall methodology is, we can describe some situations when is appropriate to use it:

- The product definition is stable and its requirements are well known, clear and totally fixed. It is also advisable the project to be simple.
- The technology that will be used is well understood and we have access to all the needed resources.

2.3. Need to move forward

As has been mentioned, traditional methodologies, as waterfall methodology, *work with a detailed plan*, where all the tasks and their delivery dates are determined at the beginning of the project. Any change that needs to be included, will assume a delay in the delivery date and a remarkable increase in the final cost of the project. The increase of the cost depends on the stage where the project is, as we can observe in the *Figure 7.* This way, A. Awad, 2005, says that "traditional methods freeze product functionality and disallow change". (Awad, 2005, p. 21.)

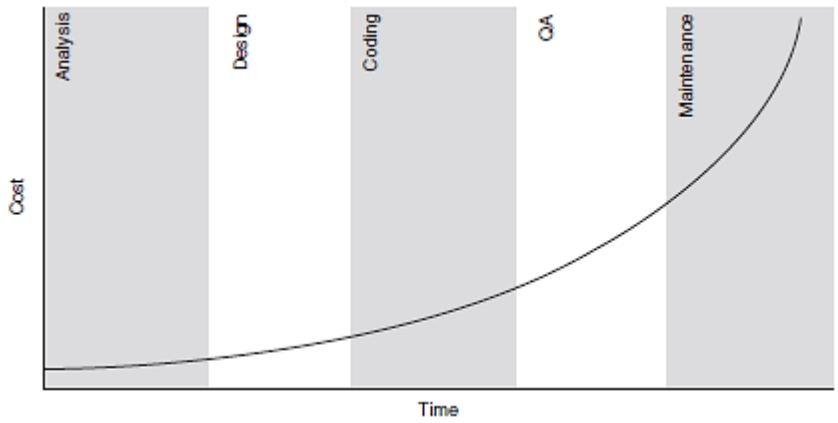

Figure 7. Cost vs. Time

Nowadays the complexity of our technology, and specially the dynamics of market-driven needs, may cause more problems. It becomes extremely hard and expensive to elaborate a precise long-term plan, which takes all these constantly changing constraints into account. It is evident that successful projects must be organized in an adaptive and flexible way in order not to fail. From now on it has to be assumed that change is the only thing that will happen for sure. Because of this, there is a need to move forward, to move into adaptive approaches.

The agile model uses an adaptive approach where there is *non-detailed plan* and only the tasks related to the main characteristics of the final product are previously established. The project may achieve success easily when it is developed in much short iteration, instead of being focused on meeting the fixed target milestones. It is better not to waste time planning future details which probably will need to be changed over time. Iterations will help to structure a comprehensive project into smaller manageable units. Stober and Hansmann, in their publication in 2010, argue that "going forward with short iterations rather than planning for the future in advance will allow focusing on the present and continuously adapting to incoming requirements". (Stober & Hansmann, 2010, p. 7)

Another aspect that needs to be improved is the way that the teams are managed. Innovative and efficient teams cannot be managed in a centralized top-down management approach. Teams need to feel themselves as the main users, instead of workers, this way they will be more motivated and the project will have more probabilities to succeed. Agile methodologies focus on the talents and skills of individuals, and processes are assigned to specific people, there are no predetermined roles.

In conclusion, in my opinion there is a need to move forward to adaptive methodologies for at least two main reasons: the limitation of traditional methodologies handling complex projects, which cannot be planned in long-term, and the difficulty that these traditional methods have to accept changes.

3. Agile software development methodology

3.1. Background

Software development has been an issue of discussion for decades. Many methodologies have been proposed in order to be able to develop software in an organized way and to deliver it faster and cheaper than the existing methodologies.

Lately, most of the suggestions for improvement have come from experienced practitioners, who have labelled their methods as *agile software development*. These methods of developing software have been revolutionary and well introduced in this field because of the innovation of their ideas. However, Dyba and Dingsoyr, 2008, argue that though there are many agile methods, little is known about how these methods are carried out in practice and what their effects are. (Dyba & Dingsoyr, 2008)

The first mention of adaptive software development process was in 1974 by E. A. Edmonds. Concurrently and independently, the same methods were developed and deployed by the New York Telephone Company's Systems Development Center under the direction of Dan Gielan. In the early 1970s, Tom Gilb started publishing the concepts of evolutionary project management (EVO), which has evolved into *competitive engineering*.

Since mid-1990s, in reaction of traditional methods, which are plan based, some lightweight software development methods came up such as Unified Process and Dynamic Systems Development Method in 1994, Scrum in 1995, Crystal Clear and Extreme Programming in 1996, and Adaptive Software Development and Feature-driven Development in 1997. Although these methods were originated before the publication of the Agile Manifesto in 2001, it was in 1998 when these methods were started to been called *agile*. (Wikipedia)

Many sources refer to the following methods as inspiration for agile development: (Dyba & Dingsoyr, 2008)

- **Agile manufacturing**: it is seen as the next step after Lean manufacturing in the evolution of production methodology. Managing change, uncertainty, complexity, utilizing human resources, and information are key concepts in this system. “An enabling factor in becoming an agile manufacturer has been the development of manufacturing support technology that allows the marketers, the designers and the production personnel to share a common database of

parts and products, to share data on production capacities and problems". (Wikipedia)

- **Lean software development**: it has been rooted in the Toyota Production System from the 1950s as a translation of lean manufacturing. Lean is most popular with start-ups that want to penetrate into the market, or with ideas which are tested for making a viable business. Some of the core ideas in this system are eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build quality in and see the whole.

Both of these methods share many ideas with Agile Software Development, especially Lean, which some experts consider it as one of several approaches of Agile. The issue about if Lean Software Development is an Agile Toolkit for Software Development or if there are many differences between these approaches is really complex, but, due to the time limitation, it is not going to be discussed in this thesis.

3.2. Methodology description

Agile software development is a group of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to change. (Wikipedia)

The first time that the concept of agile was introduced in software development was in 2001 when 17 software developers published the "*Manifesto for Agile Development*", also known as the Agile Manifesto. The values presented in the Agile Manifesto can be understood with the following publication: (Manifesto, 2001)

"*We are uncovering better ways of developing software by doing it and helping others to do it. Through this work we have come to value:*

- ***Individuals and interactions*** *over Processes and tools*
- ***Working software*** *over Comprehensive documentation*
- ***Customer collaboration*** *over Contract negotiation*
- ***Responding to change*** *over Following a plan*

That is, while there is value in the items on the right, we value the items on the left more."

These central values that the agile alliance adheres to are, first of all, the importance of the interaction between the developers. Human factor is more important than processes and development tools, communication helps to improve the work.

They also remark the main objective of the project team is to focus on working software, on testing and improving it in order to keep it simple, insetad of documenting all the work that is done. The idea is to profit the time by working software. This value must not be missunderstood, documentation is importan but it has to be kept on the appropiate level, not exceed it.

The relationship and cooperation between the developers and the client is given the preference over contract negotiation. Maintaining the client well informed gives advantage the project to succeed, the client can apreciate how the project developes and posible changes are informed earlier. This also envolves the last value, which means that both developers and customers are ready and capable to introduce any change into the project, despite the fact that then established plan could change.

As Stober and Hansmann suggested in 2010, it is also important to emphasize that "agile software development is not just a single set of rules to follow in order to be successful. Agile methodologies comprise leadership concepts, lightweight project management processes, engineering best practices, development techniques, as well as supportive tools". (Stober & Hansmann, 2010, p. 12)

This basically means that an agile project will not be able to pick a process or a set of rules and just execute it. The process will need to be shaped by the members of the team, using their own experience from old projects, and adapt it to the current project. Agility is a pool of thoughts, and then the team's way of work is a mix of practices. The resulting process will be determinate by the needs and context of the project. It is also a need that the development process doesn't become too complex, or it won't be able to be executed; flexibility and simplicity are the key to success.

3.2.1. Agile Methods' Structure

"Critics of agile software development are very apprehensive about agility leading to chaos and anarchy: If you were not creating a detailed schedule with expected work items and estimated efforts throughout your project, you would have nothing to measure your progress against". (Stober & Hansmann, 2010, p. 93)

Despite the effort of these critics of agile, agile development has a structure of planning. This development doesn't focus on planning all the iterations of the project since the beginning; with agile development the plans are made according with how far in the future is the delivering. That means that the team focuses on planning well detailed the current iteration, but it also has a rough plan for the entire project.

This way of planning allows developers to know which tasks are planned for the current iteration, and to have an idea of which are the priority tasks that will need to be done to reach the main goal.

The *project plan,* which is the one that includes the entire project, must include some aspects for defining the general way for reaching the main goal. A project plan must include the high level goals and priorities, assigned teams and their mission, main areas of work, backlog with use cases, major milestones, the number of iterations and the general objectives of each iteration. Basically, it needs to include how the team will be organized and how will the developers work.

In the other hand, the *iteration plan*, which only covers the current iteration, needs to detail aspects as the team iteration objectives, iteration backlog with detailed use cases, updated effort estimates and the evaluation and acceptance criteria. It is a well detailed plan, which defines properly all the needed aspects of the current iteration.

The architecture idea of agile development can be better understood in the *Figure 8,* which shows the difference between a low level and a high level of detail:

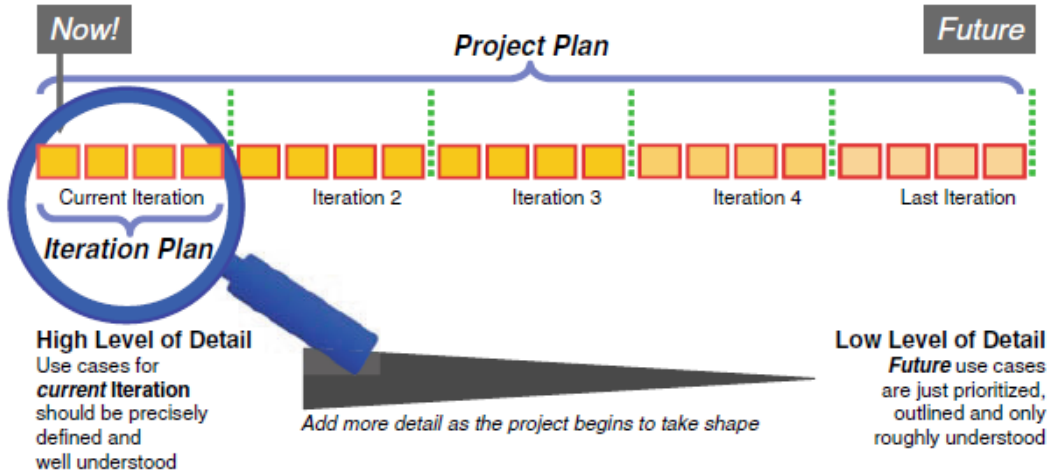

Figure 8. Difference between high and low level details

Observing the structure that each methodology follows for planning the project, we can notice the difference between the traditional methods, such as waterfall, and the agile methods. Meanwhile the traditional methods follow a tough plan which implies unchanged requirements and fixed objectives for each iteration; agile development gets to know the aspect of continuous planning, which implies dynamics way of moving and continuous improvement. In the *Figure 9* can be noticed how the projects are driven depending on which type of methodology they are following.

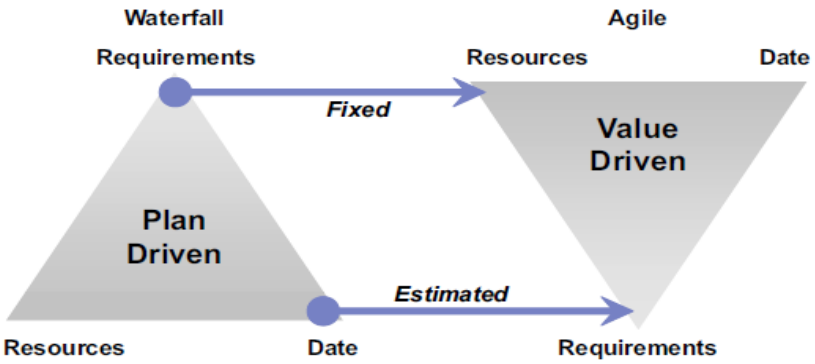

Figure 9. Projects' structure according to the used methodology

3.2.2. Agile Management

So far it is understood that agile development methods consider change as a fact, it is also need to consider a different way of teaming and leadership. Agile development methodologies need "motivated and innovative teams, prepared to adapt to change with the spirit to perform and deliver customer value without being hindered by confinements of regulations. It is needed to foster a climate of smart collaboration and intensive communication to bring the best skills together and to come up with interdisciplinary creativity". (Stober & Hansmann, 2010, p. 75)

Jim Highsmith (Highsmith, 2009) had studied in depth the topic of agile project leaders. He says that "*A traditional project manager focuses on following the plan with minimal changes, whereas an agile leader focuses on adapting successfully to inevitable changes*". For this reason he encourages agile teams to be managed as "team" management style, what means enable teams to self-manage their own tasks to facilitate the completion of features and assist the team to remain coordinated and effective so that they can succeed.

Too often project managers focus on the usual constraints of time and cost, there are times when value doesn't seem to matter at all. Then there are project managers who

focus on scope and detailed requirements, but not on the end goal of value. In order to succeed with agile management he suggest three key values for agile leaders which consist in "*delivering value* over meeting constraints, *leading the team* over managing tasks and finally, *adapting to change* over conforming to plans". (Highsmith, 2009, p. 54)

Thomas Stober and Uwe Hansmann (Stober & Hansmann, 2010, p. 78) say that "while the strategic orientation of a company is defined to a large extent from the top to the bottom, the detailed planning is done in a bottom-up approach by the individual teams within their scope". Basically this bottom-up management means that the project development is driven from the bottom of the hierarchy.

This way of management includes agreed goals, what needs to be done, but doesn't detail how things will be implemented. "The team inherits the necessary responsibility and authority to pursue suitable tasks and work items on its own. Managers cannot expect teams to perform at a high level without allowing them to be the authors of the solution" (Stober & Hansmann, 2010). Obviously, if the personal goals of an individual are aligned with the project-specific goals, the development of the project will be more successful.

3.2.3. Summary

In Agile Manifesto all this characteristics that have been explained in this thesis, are summarized in twelve clear principles. These principles are: (Manifesto, 2001)

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the principal measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design.
10. Simplicity, the art of maximizing the amount of work not done, is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

In conclusion, a development method can be considered agile when "software development is **incremental** (small software releases, with rapid cycles), **cooperative** (customer and developers working constantly together with close communication), **straightforward** (the method itself is easy to learn and to modiidy, well documented), and **adaptive** (able to make last moment changes)". (Abrahamsson et al., 2002, p.17)

For any interest in knowing more about some cases where agile methods were used, Dyba and Dingsoyr, 2008, in their publication "Empirical studies of agile software development: A systematic review", they identified 36 empirical studies with the purpose of investigating what is currently known about the benefits and limitations of agile methods. (Dyba & Dingsoyr, 2008)

3.3. Agile development vs. Traditional development

Now that both developments have been properly defined, it is possible to compare briefly agile and traditional development in order to clarify the possible doubts between these methodologies. The main characteristics are displayed in the following *Table 1*. (Stoica, 2013, p. 72)

	Traditional development	Agile development
Fundamental hypothesis	Systems are fully specifiable, predictable and are developed through extended and detailed planning.	High quality adaptive software is developed by small teams that use the principle of continuous improvement of design and testing based on fast feed-back and change
Management style	Command and control	Leadership and collaboration
Communication	Formal	Informal
Development model	Life cycle model (waterfall, spiral or modified models)	Evolutionary-delivery model
Organizational structure	Mechanic (bureaucratic, high formalization), targeting large organization	Organic (flexible and participative, encourages social cooperation), targeting small and medium organizations
Quality control Permanent testing	Difficult planning and strict Control. Difficult and late testing	Permanent control or requirements, design and solutions.
User requirements	Detailed and defined before coding/implementation	Interactive input
Cost of restart	High	Low
Testing	After coding is completed	Every iteration
Client involvement	Low	High
Additional abilities required from developers	Nothing in particular	Interpersonal abilities and basic knowledge of the business
Appropriate scale of the project	Large scale	Low and medium scale
Developers	Oriented on plan, with adequate abilities, access to external knowledge	Agile, with advanced knowledge, co-located and cooperative
Requirements	Very stable, known in advance	Emergent, with rapid changes
Primary objectives	High safety	Quick value

Table 1. Traditional vs. Agile development

For having an idea of which way of working is better than the other, the IT Project Success Rates Survey explores how effective the five most common software paradigms are. The survey explored the effectiveness of Lean, Agile, Iterative, Ad-Hoc, and Traditional, by asking respondents whether their organizations had used such developments approaches and if so, how successful were they. (Dr.Dobb's, 2010)

Due to the limitation of the thesis, the survey is just focus in Agile and Traditional developments. For this survey, a project is considered “successful if a solution has been delivered and it met its success criteria within a range acceptable to your organization; challenged if a solution was delivered but the team did not fully meet all of the project's success criteria within acceptable ranges (e.g. the quality was fine, the project was pretty much on time, but ROI was too low); and a failure if the project team did not deliver a solution at all” (Dr.Dobb's, 2010).

The IT Project Success Rates Survey that have been used are from the 2010 and from the 2013, this way can be appreciate the introduction of agile methods in software development. The survey results have been extracted from (Dr.Dobb's, 2010) and (Dr.Dobb's, 2014).

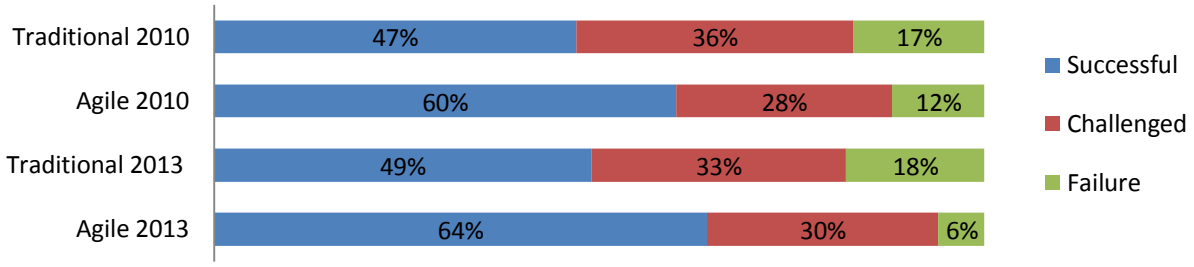

Figure 10. Results of the survey

As it can be appreciate in the *Figure 10*, agile projects are more successful than traditional, and they have also become more successful between the years 2010 and 2013. This evolution of agile software development projects is normal, teams also needed to be trained and learnt how agile values really work.

In the *Figure 10* can be also observed that the percentage of failed projects is bigger with the traditional methodologies, and that they tended to fail more in 2013 than in 2010.

About the challenged projects, the difference between traditional and agile projects was more notable in 2010 than later in 2013.

3.4. Different types of agile methods

Simplicity is the most important issue about agile methods, which also have to allow the deliverance of the software in the faster possible way. Agile methods can success by focusing firstly on the most important functions and by reacting to the feedback.

During the last few years, as it has been comented before, there has been a huge development of new methodologies, both traditionals and agile. The most known agile methods are Extreme Programming, Scrum, Rational Unified Process, Dynamic Systems Development Method, Open Source Software Development, Crystal Family of Methodologies, and Adaptive Software Development.

Due to the limitation of the thesis, Extreme Programming, Rational Unified Process and Scrum are the only methods developed, also in short way, just in order to get an idea about what agile methods consist. These methodologies have been choose to be developed due they are the most common agile methods used nowadays.

For more information about agile methods, it is recommended conferences and books like (Ullah, 2014) and (Abrahamsson et al., 2002).

3.4.1. Extreme Programming (XP)

Extreme Programing was developed by Beck in 1999 due to the problems caused by the long development cycles of traditional developments model. It is called extreme basically because it takes the common practices and brings them to extreme levels.

XP introduced a new way of developing software by introducing concepts like efficiency, change expectation, communication, automated tests, collective code ownership and the more particular one, storytelling culture. (Abrahamsson et al., 2002)

The lifecycle of XP consists of five distinguished phases. The first phase is *Exploration* where customers write out the story that they wish to include in the first release; the stories are specific functions of the program. This phase is also used by the program developers to familiarize themselves with the technology, or some specific tools, that will need to be used during the program development.

The next phase is *Planning*, where the stories are set in order according to their priority, and the programmers agree a schedule and an estimated effort for each story. The *Iterations to release* phase is the one where the schedule, realized during the planning phase, is broken down into small iterations, which take about one to four

weeks. The stories that are used in each iteration are selected by the customer, who is also in charge of creating the functional tests that will be run at the end of each iteration. Once the iteration has been tested, the system is ready for production.

The *Product ionizing* phase requires extra testing before the system can be released to the customer. At this phase, new changes, if wanted, can still be included in the current release. New suggestions can also be documented for later implementation, for example during the maintenance phase.

After producing the first release, the *Maintenance* phase becomes really important, because the XP project must keep the system in the production running while producing new iterations. In this phase is needed to add new people to the team, change the team structure and count on the customer effort.

When the customer doesn't have more stories to implement, the project arrives to the *Death* phase. Then is when the customer needs are satisfied and all the necessary documentation has to be written. Death can also occur if the project isn't able to be delivered, either for elevated cost or for not presenting the desired outcomes.

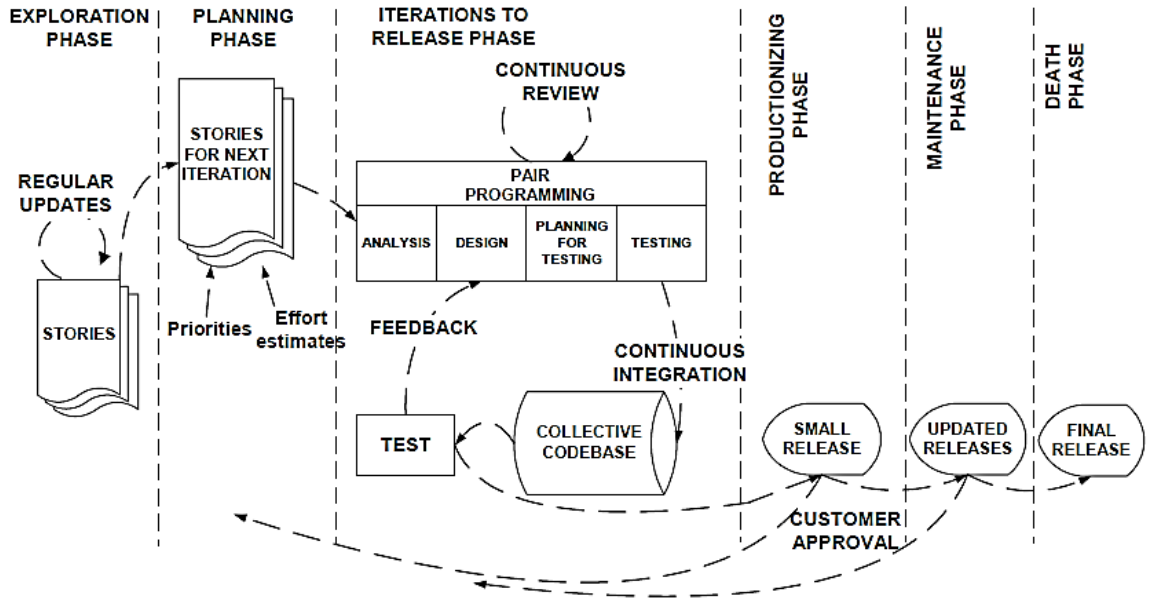

Figure 11. Extreme Programming process

3.4.1. Rational Unified Process (RUP)

The Rational Unified Process was developed by Philippe Kruchten, Ivar Jacobson and others at Rational Corporation to complement Unified Modelling Language, an industry-standard software modelling method.

Using the RUP, the development cycle is divided into four phases: Inception, Elaboration, Construction and Transition. These phases are also split into different iterations, where each of these iterations has the purpose of developing a piece of the software. (Dingsoyr et al. 2007)) (Abrahamsson et al., 2002)

The Inception phase is where requirements and life-cycle objectives are defined, as well as the schedule and costs. During the inception phase, critical use cases and needed resources are also identified.

The next phase is *Elaboration*, “it is where the foundation of software architecture is laid” (Abrahamsson et al., 2002, p.56). At the end of this phase the project has solid software architecture with stable requirements and plans. Also an analysis is made for determining the risks, the reliability of the design and the use of resources.

After having the entire architecture, the solution is implemented and tested during *Construction phase*. RUP considers this phase as a manufacturing phase, where controlling cost, time and quality is really important. At this point, the team needs to decide if the features and the quality of the solution meet the requirements and if it is ready to be released to the user community. The first test is commonly called Beta test.

Finally, during *Transition phase* is when the product is mature enough to be realised to the users, who help the project providing feedback, reporting problems and adding new requirements. Based on user response, some changes are made to correct any outstanding problems. After several iterations the product can be rolled out to marketing distribution and sales teams. In this phase documentation is also produced.

Through the phases, nine *workflows* take place in parallel. The workflows are *Business Modelling, Requirements, Analysis & Designs, Implementation, Test, Configuration & Change Management, Project Management and Environment.* All are very common in software development, except maybe Business Modelling which is for ensuring that the customer’s business needs are catered for.

3.4.2. Scrum

The Scrum process has been developed for managing the systems development process. It also can help to improve the existing engineering practices in an organization.

Scrum is founded on empirical process control theory, which asserts that knowledge comes from experience and making decisions based on what is known. It introduces the ideas of flexibility, adaptability and productivity; this way Scrum teams focus on produce with flexibility in a constantly changing environment. (Schwaber & Sutherland, 2011)

Scrum consists of Scrum teams and their associated roles, events, artefacts, and rules. Each component within the framework serves a specific purpose and is essential to Scrum's success and usage.

Scrum process includes three phases: pre-game, development, and post-game. (Abrahamsson et al., 2002)

During the *pre-game phase*, two parts can be differentiate: planning and architecture. Planning is basically the definition of the system that will be developed. During the planning a **Product Backlog list** is created. This list contains all the requirements that are currently known and is constantly updated during the entire project. Normally this list is realised with "Post-it" in order to make easy the comprehension and the display. Planning also includes the structure of the team, the needed tools, risk assessment and controlling issues, and verification management approval. The second part, the architecture of the project, consists in planning the iterations based on the current items in the Product Backlog.

The *development phase* is the agile part of the Scrum approach. It is developed in **Sprints**, which are iterative cycles where the functionality is developed. Each Sprint contains the traditional phases of software development: requirements, analysis, design, evaluation and delivery.

The *post-game phase* contains the closure of the release. When there are no more items or requirements, the systems is ready to release. The preparation for realising is done during this phase, including the tasks such as integration, system testing and documentation.

As mentioned before, what entirely defines Scrum development are the roles and responsibilities that can be identified during the entire project, and the management practices and tools that are employed in the various phases of Scrum. There are six identifiable roles in Scrum: Scrum Master, Product Owner, Scrum Team, Customer, User and Management. These roles allow having an organised structure inside the project. For better comprehension, some of these roles need to be described: (Abrahamsson et al., 2002)

- **Scrum Master**: is the responsible of ensuring that the project follows the accorded practices, values and Scrum rules. The Scrum Master is the man in charge of interacting with the project team, the customer and management. He is also responsible of securing that the team can work as productively as possible.
- **Product owner**: is the responsible of the project, and is selected by the Scrum Master, the customer and the Management. He makes the decisions of the tasks related with the Product Backlog list.
- **Management**: participates in setting the requirements and goals, and is also in charge of making the final decisions.

The prescribed events and artefacts used in Scrum to create regularity and to minimize the need for meetings, according to Ken Schwaber and Jeff Sutherland (Schwaber & Sutherland, 2011) are:

- ***Sprint Planning meeting***: it is organised by the Scrum Master, and consists in a meeting with the customers, users, management, Product Owner and Scrum Team to decide upon the goals and the functionality of the next Sprint. After this, the Scrum Master and the Scrum Team focus on how is implemented the product increment during the Sprint. (Abrahamsson et al., 2002)
- ***Sprint Backlog:*** it is the start of each Sprint. It is a selection of Product Backlog items that will be implemented in the next Sprint. The items selected to be implemented are selected by the Scrum Team, the Scrum Master and the Product Owner. The difference between the Sprint Backlog and the Product Backlog is that the first one doesn't change until the Sprint is completed. There are no actualisations while the Sprint is being developed.
- ***Daily Scrum meeting****:* these meetings are organised to keep track of the progress of the Scrum Team continuously and they are also used to plan the next meeting.

- ***Sprint Review meeting****:* it is done the last day of the Sprint. The Scrum Team and the Scrum Master present the results to the management, the customers, users and the Product Owner. In this meetings new items come up to be introduced in the Product Backlog.
- ***Sprint retrospective***: It is an opportunity for the Scrum Team to inspect itself and create a plan to improve during the next Sprint. This meeting occurs after the Sprint Review meeting but before initialising the next Sprint.

Scrum development is a complex methodology of how structure a project, and for this reason all the concepts have to be really clear. One good way to clarify the process is with the *Figure 12,* where all the roles and practices are represented.

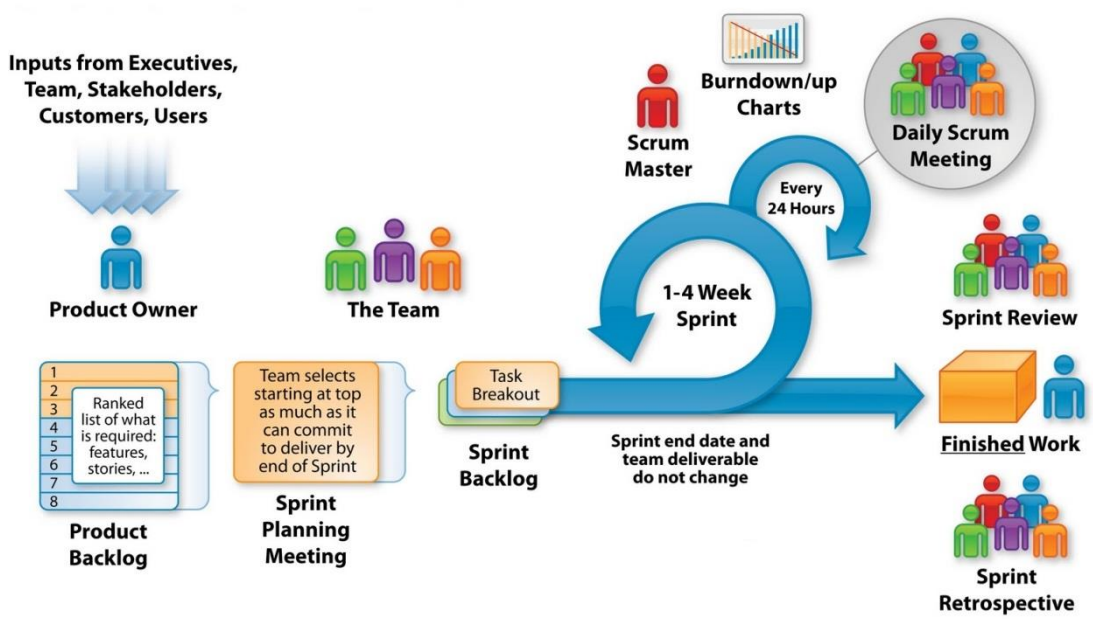

Figure 12. Scrum architecture

4. Adapt agile approaches to mechanic and mechatronic engineering

In the first moment, agile methodologies were thought for developing software, but few years later, after the success of these methods in software, the question is if these methods would suit for mechanic and, maybe also, for mechatronic development.

Back in 2001, 17 software developers signed the now-legendary document, the Manifesto for Agile Software Development. More than a decade later, in 2012, four of the original authors, Ron Jeffries, Jim Highsmith, Arie van Bennekum and Andrew Hunt, discussed the agile manifesto's impact on project management in an interview with Michelle Bowles Jackson. The whole interview can be found in the Annex, *"Interview of agile authors".* (Jackson, 2012)

During the interview they were asked about ***what agile's place is outside of software development*** and their answers were quite optimistic. Ron Jeffries, the author of Extreme Programming Installed, answered:

"*The key ideas apply everywhere. These include, but are not limited to:*

- *Putting people with needs in direct contact with people who can fulfill those needs.*
- *Populating projects with all the needed people and capabilities to get the job done.*
- *Building work incrementally and checking results as you go.*
- *Preparing for and influencing the future but not predicting it.*
- *Making tasks concrete and quickly finishing them.*
- *Giving people work to do and the knowledge to do it, not pushing them around like pawns on a chessboard.*
- *Focusing on providing value frequently and rapidly, not directly on cost.*

Agile ideas are based on how people and organizations work best. There are specialized details we need to know regarding software, just as there are in any other domain. The principles, however, are broadly applicable."

In the same interview Jim Highsmith, the primary developer of the Adaptive Software Development agile method, said that "*In any project that faces uncertainty, complexity, volatility and risk, there is a place for agile practices and principles*".

Arie van Bennekum also agreed with his partners about using agile methods outside software development, furthermore he answered that "*Agile is holistic and applicable everywhere in business or life. I use it as a concept wherever I am and for whatever I do, from defining online strategies to the total refurbishment of my house*".

The last comment was from Andrew Hunt who, with the following answer, mentioned the small paper that agile plays in software development.

"The agile mindset is critical to successful business in the 21st century. At its heart, an agile approach has little to do with software; it's all about recognizing and applying feedback. The definition of "agile" that Venkat Subramanium and I proposed in Practices of an Agile Developer states, "Agile development uses feedback to make constant adjustments in a highly collaborative environment."

Notice there's nothing in there about software. The Pragmatic Bookshelf publishing company, in fact, has been called an "agile publishing company" by our many fans. We seek to embody the principles of agility throughout our business practices.

Charles Darwin famously said, "It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change." We have become keenly aware of this through our work with software. But rapid adaptation is the single most important idea of this century for any business, in any market."

From this interview can be worked out that agile methods *could be applied to other types of engineering*, but the issue now is the lack of information on this field. In this thesis the information about how to apply agile methods to mechanic or mechatronic engineering is organized in two parts:

- The first part consists of the summary of the opinion and commentaries of different experienced people, who have discussed this issue in public blogs or web pages. These blogs contain explanations about both personal practices and theories to apply agile approaches to another type of engineering.
- The second part is the summary of some case studies of Mechanical Products Development Teams using Scrum. These case studies were developed by a student of the Chalmers University of Technology in his Master thesis.

4.1. Opinions and commentaries of experienced users

This first part is a summary of opinions of different users about how agile methods could be applied to mechanical or mechatronic engineering. The opinions and commentaries are divided in three blogs, depending which is their origin source. The first blog is the development of a theory of how agile values and practices could suit hardware development, the second and the third blogs are experiences and advices of users who have already tried to apply agile in their mechanical projects.

4.1.1. Blog 1: How to apply agile in hardware product development

This first blog is about a theoretical solution of how to apply agile principles in hardware product development. This theory is explained in this thesis because, although hardware is considered to be electronic engineering, hardware development combines both mechanical and electronic disciplines, for this reason, this is considered an issue of interest.

Eric Graves, an aerospace and mechanical engineer, is the author of the development of this theoretical solution. The resource is a set of seven posts, all about “How to apply agile in hardware product development”. The blog where these posts can be found is called “PLAYBOOK”, which contains information about Lean, Agile, Flow and Theory of Constraints. (Graves, 2015)

Graves in his first post remarks the rapid growth of Agile development with software products due to the huge financial and cultural improvements that it provides, but in the other hand, he also remarks the slow advance that agile development has had with hardware product development. From his point of view, the reasons why agile doesn’t advance in hardware development is because there are a lot of differences between software and hardware development.

In order to be able to apply agile methods into hardware, first we need to mention the top three differences that have huge impact on planning, managing and executing the projects.

- ***Lead Time:*** meanwhile software teams have a relatively short compile step which resides within the Design-Build-Test cycle; hardware teams have a relatively long procedure step.

- **Component Cost**: the cost of software development is almost all labor; however, hardware products have also a material cost, which is more difficult to minimize.
- ***Non-Homogenous Work:*** software projects are comprised of several people with different skills such as marketing, design, different fields of development, and quality; but hardware projects require even more skills. This means that hardware teams are integrated for more people and more variety of skills.

In his blogs, Eric Graves tries to define the implications of these differences and how agile principles, practices and tools need to be modified to meet the challenges of hardware product development.

Agile values

As defined in this thesis in the point "*4.2. Methodology description"*, the agile values are Individuals and interactions over Processes and tools, Working software over Comprehensive documentation, Customer collaboration over Contract negotiation and Responding to change over Following a plan.

For illustrating these values the author considers "the value of the top goal as a weight being supported by the lower-level requirements. The relative value of the lower level requirement is represented by the strength of the spring which supports the goal. The greater the value of the sub-requirement, the stronger the spring and the more weight of the goal that sub-requirement supports" (Graves, 2015). Software products are illustrated in the *Figure 13,* meanwhile hardware products are described in the *Figure 14.*

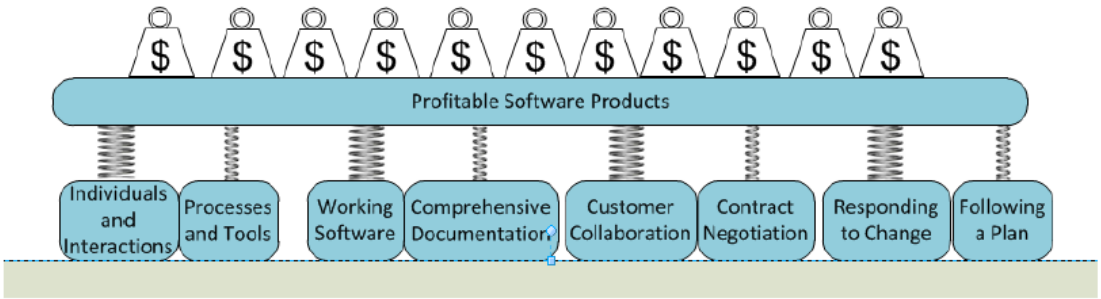

Figure 13. Profitable Software Products

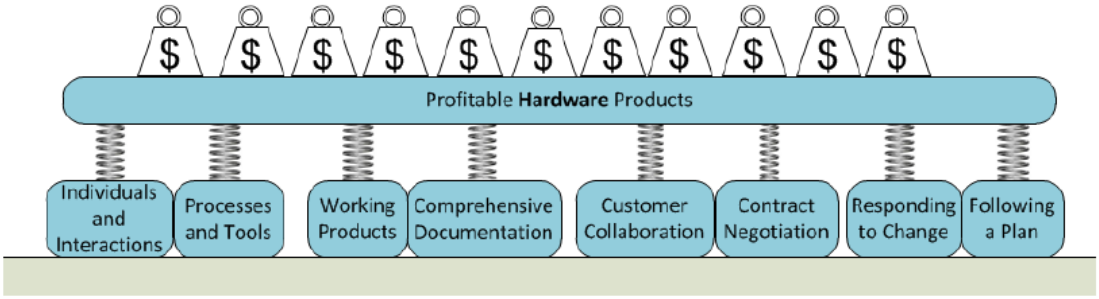

Figure 14. Profitable Hardware Products

The reason why the same springs present different strength in Software than in Hardware is because hardware development conditions create different costs and risks. Facts like procurement times, component costs, large teams and skills variety reduce the value of the practices on the left, and transfer value to the practices on the right.

- **Individuals and Interactions vs. Process and Tools:** in hardware development, as the team size increases, the number of interactions increases exponentially and also the complexity of the system. In order to combat this increasing complexity, a good framework of processes and tools which keeps everyone on the same page, using good practices, and communicating well is even more essential. Although face-to-face is often the fastest way to communicate, not all the implicated people will be able to be present when a change is made. *The key is to minimize the effort involved*.
- **Working Software vs. Comprehensive Documentation:** in this case we can replace "Working Software" for "Working Product". While software code is difficult to understand, often is more difficult to look at a mechanical or electrical design and gain much understanding and knowledge. Knowledge is often better transferred through documentation when face-to-face transfer isn't possible. The clue is to keep documentation effort at a minimum, so the team can focus on getting product working.
- **Customer Collaboration vs. Contract Negotiation:** the cost of customer collaboration is much larger in hardware development, largely because of the cost of the parts. The ability to get feedback is limited by how many test units can be afforded to procure.
- **Responding to Changes vs. Following the Plan:** changing hardware implies more cost than changing software. The procurement time required to get new parts causes delays and involves rework costs. As a result, changing plans is often costly.

Agile principles

Agile principles were generated by software developers for software development, so they are software focused. Some of these principles cannot be used with Hardware Product development, which involves physical components. For this reason, they need to be carefully adapted, in order to not create more problems than the solved.

For using the agile principles for hardware development, the words “software” and “developers” have been replaced for “products” and “engineers”, respectively. In fact, these changes can be applied for all types of engineering. After the changes, agile principles look like this:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable products.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer‘s competitive advantage.
3. Deliver working products frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and engineers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within an engineering team is face-to-face conversation.
7. Working product is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, engineers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

In his blog, Graves makes a review of each principle, one by one, mentioning all the discrepancies between software development and hardware development. The most important conclusions and changes of these principles are following summarised.

The 1st principle says that our priority is to focus primarily on delivering valuable products to customers early and continuously. Even among the most advanced product

development companies, continuous delivery is rarely found to be the most profitable option for hardware products. This is basically because changes must pay back more than what they cost, only this way a business can be sustainable and profitable.

For this reason, the alternative for the first principle for hardware development would be something like:

"Our highest priority is to satisfy the customer early through continuous delivery of value."

The fact that it is not usually profitable to release hardware products continuously to customers does not preclude from *developing* the product more continuously. However, the definition of value must change from being singularly focused on customers receiving products. Instead, the delivery of continuous value comes from continuous reduction of project risk.

About the 2nd principle, in hardware development *some* changes also need to be welcome even late to maximize the profits. In this case, the challenge is identify changes as early as possible, enable the ones which are easier and more cost effective, and know when the effort of changing is worth.

One of the main problems of welcome change is the limited modularity. Because of the cost-and-performance-limited modularity of many hardware products, products tend to be highly integrated. Therefore, when something needs to be changed, it ends up needing to change a larger percentage of the product.

Another fact to keep in mind is that the time between design and test in hardware is necessarily larger, mostly because of the procurement time. This fact increases the delay, which increases the cost and reduces the economic value of the change. As a result, few of the changes turn out to be profitable.

Some advices that would be helpful for being better prepared for changes could be establish more modularity in the products, establish which requirements are believed to change and which components would be impacted by those changes and consequently, establish modularity in those risky places where change is expected.

Now, the purposed alternative for this second principle is:

"Enable and welcome profitable changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."

The 3rd and 7th principles discuss one of the most difficult issue and also the key for succeed with agile, the agile practice known as “sprint”. It seems almost impossible in hardware product development to deliver working product every two weeks. “Working product” refers to a physical object which internal and external customers can use well enough to happily pay for it or to provide good feedback about it.

In order to deliver working product and profits, both hardware and software development follow the Design-Build-Test (DBT) path. In fact, before being able to deliver some working product, the product needs to go through several DBT iterations. The number of DBT iterations is determinate by the number of errors. There are two types of errors: the ones that can be found through internal testing and the ones that cannot. Internal testing finds most of the defects (design and process errors) and some lower-level requirements errors, which can be relatively quickly and easily fixed. When the errors are under the working product threshold line, represented in the *Figure 15,* the product is considered to be working product and can be delivered to the customers. Customers will do the external testing, where the higher-level requirements are usually found.

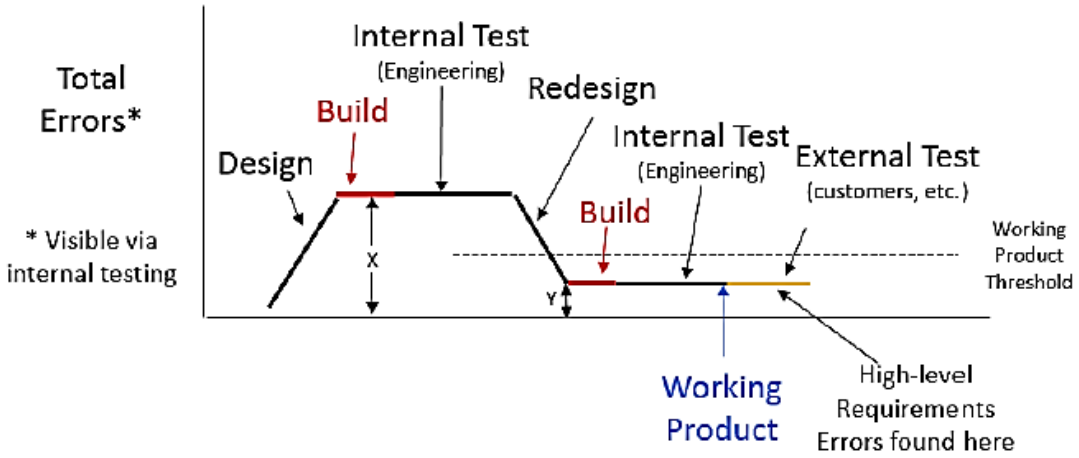

Figure 15. DBT iterations

Due to the existence of physic components in hardware development, the DBT cycle is relatively longer than for software. If the sprint is from 8 weeks, and the procure parts are done in 2 weeks, which is quite fast, that only leaves 4 weeks to do all the design, assembly, test, redesign, reassembly, and retest. This could be possible if the product just needs two iterations for becoming working product, but it usually needs more. Innovation is what sells, and it can rarely be done in only two iterations.

One way to fit into the sprints is reducing requirements, which reduces the test time and sometimes also the build time. However, the redesign time often still involves a

large percentage of highly integrated components, so it doesn‘t usually get proportionally smaller.

Another way to get working product earlier is recognizing that physical prototypes can produce the valuable feedback we need earlier. Even if the product is not ready for sale, it can provide good value from the feedback of the customers. The more “real life” is the test, the better.

Finally, the working product used to measure progress in hardware, especially early in development, is spread across multiple, independently progressing project risks and the mock-ups and tests which provide value. This makes it a little more difficult to measure, but it is still measurable. It can be measured via an overall risk burn-down score, which is the sum of the scores of individual project risks. In fact, burning project risk is what product development is all about, and maximizing the risk burn rate is how projects are completed and profits are created the fastest way possible.

The principles from 4 to 6 and 8 to 12 can be successfully applied to hardware development, they don’t require any change. The 4th, 6th and 11th principles are needed basically because product development is all about information. The key to success is to eliminate unnecessary delays and not waste valuable time on invaluable information. These three principles are about this, helping valuable information flow faster.

The 5th and 8th principles are all about being sure everyone in the team has what is need, including maximum availability to the tasks. Engaged and motivated people are focused and desire to get the work done.

The 9th and 11th principles just remain the importance of having things well done since the first moment and of minimizing the done work in order to achieve the goal.

Finally, the 12th principle just makes team to have a look at the past and learn from it. Products, development processes, and ultimately profits can be improved when the team reflects often on their actions.

In conclusion, as we can see, most of the agile values can be applied for hardware development. However, some of them need to be adapted to fit hardware development properly. This is a very complex topic, which hasn’t been totally developed yet. For further information, the reader can address to (Graves, 2015).

4.1.2. Blog 2: Sharing experiences

In this blog the information source is a Google group, where the main author is a Scrum Master who works with CNC machine development projects. These projects contain different disciplines like design, mechanics, mechatronics, electronics and software. In this group, the author explains that his company has already been successful applying Scrum in software development, and now, in other to go further, he would like to find experienced people willing to share with him their own experiences about applying agile values into another engineering discipline, besides software. (Grugl & Dayley, 2013)

In order to be able to apply agile values into other engineering approaches, he mainly would like to know if scrum is actually the way to move on, how mechanics and hardware developers can deal with delays and how to build machines which easily accept changes. As it is concerned, these are really important questions that would need to be answered in order to move on.

Unfortunately, the author just received one useful answer from a software engineer, who had his first experience into Agile with projects that were developing physical data products for harsh environments. These projects involved software engineers, Field Programmable Gate Array (FPGA) designers, electrical and mechanical design engineers.

His team also started applying Scrum firstly with software development, and this pulled FPGA designers, and after, the hardware engineers to get into agile values. He affirms that they actually didn't get into Agile with mechanical and electrical engineers, though they attend to all the daily meetings and they were aware of all the planning.

One useful advice that was extract from this experience was: "***Get as soon as possible a prototype and an adaptable hardware"***

This way, the team would be able to "welcome changing requirements, even late in development". The problem is that it isn't easy to get an adaptable hardware. In order to get it, he proposes to use ***brassboard*****,** which is "an experimental or demonstration test model, intended for field testing outside the laboratory environment. This model contains both the functionality and approximate physical configuration of the final operational product" (Wikipedia). The design obtained with bassboards, is not a final version but it permits to develop and experiment with hardware designs close to how the final version would look like.

Another aspect that helped mechanical engineers to approach to agile values was the ***tight relationship they had with a local machine shop***. This enabled them to get one-off parts and experiment with designs and materials. This way, the designs were able to be produced in small runs and quickly into full testing.

Although this process was more expensive, it provided a change of mentality on behalf of the organization. He argue that it was worth to spend money on methods like these, because despite the increase in the design budget, they saved money by learning faster and discovering wrong directions before a full production run.

From my point of view, another useful advice can be extracted from this experience: "***Money can be saved by investing in diminishing the wrong directions"***

Finally, we can conclude that a team can be lead into agile practices by striving to get ***more often validated knowledge*** of the mechanical designs, and also by motiving themselves to ***be more creative***.

4.1.3. Blog 3: Pro and contra ideas

The last set of opinions of this issue comes from the online application "LinkedIn", which is a business-oriented social networking service. This application allows people to create debate groups to discuss issues of interest. The interesting debate for this thesis was created by a software engineer who after achieving agile methods in his company, wondered if they would achieve any gains by introducing agile methods in their hardware and mechanical teams. (Linkedin, 2013)

The possibility of applying agile approaches to hardware and mechanical engineering generate two different kinds of reactions: believe and support that it is a good and innovation idea, or that it is an enormous error. For this reason, the multiple answers of the debate are separate and summarised according if they are pro or against it.

Ideas of how it could succeed

Most of the following ideas come from experienced engineers who have already applied, or at least tried to apply, agile values into mechanical projects or projects with differentiated engineering disciplines. This is a summary of some ideas about how agile approach could be adapted to other mainly mechanical and electronic engineering.

- It is quite difficult to involve mechanical and electrical engineers into agile without much software background, so one way to involve them more easily into agile values is changing terminology to be better aligned with these disciplines.
- The 15 min daily stand-up meeting is really valuable for any project; it improves communication between the team, or between the different disciplines of the project.
- The idea of having a cross-functional team working on the same function or sub-function in parallel, rather than in sequence, should be applied in all kind of projects. We can work better together as a team to solve a problem and reduce the risk of problems showing up in a late step of the development sequence
- Iterations and learn cycles can and should be applied to any project with a high level of unforeseeable uncertainly. Concepts like “fail fast”, from the lean product development, are important for start-ups. Every project will be different and for this reason the cycles are always going to be dependent on what needs to be learned. This will define how long the cycles are going to be.
- Time frames and tools used to implement the methodology need to be adjusted. It is not the same changing a line of code and running a regression test than changing a bug in a drawing, machining, inspection, ECO, back to drawing cycle. One way to face up to this problem, or at least to make it easier, is involving the people from the machine and fabrication shop in order to extend the available resources.
- Time boxed iterations are important for having the customer involved during the course of the project and having his feedback. His feedback can help the team to improve the project or detect possible mistakes. So, there is a need to find ways of having this customer involvement.

  Although normally we can think that the customer would be grateful for being in the meetings and being aware about what is going on, sometimes happens that the customer wants to stay out of trouble. It can be a little disconcerting for customer to be in the discussions where the team is blocked, even if they might figure it out early.

  One proposal that has worked well is to schedule a customer call once during sprint planning interval and share with them a bit of retrospective on the completed work and of the plan for the upcoming sprint. Once the team has a “mature” prototype, the involvement of the customer can be amplified.

Ideas against adapting agile approach

Some people, also talking through their experienced minds, think that companies make a mistake when they try to take methods and practices developed for software, included Scrum, and try to use them for other disciplines, including hardware development. Some of the arguments why it is not a good idea to use agile methods for other approaches are the followings.

- Scrum is based upon Sprints of relatively short lengths, with highly defined tasks that must be completed during the Sprint. The nature of software development makes this an excellent framework, but Scrum isn't necessarily the best framework for hardware development. If the products are in a highly regulated industry, then the documentation must follow industry requirements for specification and design, as well as normal testing and functional requirements documentation. This makes it extremely difficult to use Scrum by itself.
- With typical hardware design involving complex electromechanical construction, where fabrication and assembly take significant time and expense, it is not advisable to encourage some of the high iteration rate methods oriented to the relatively "free" cost software applications, which requires purchasing, stocking, fabricating and assembling complex physical parts.
- Choosing a process and trying to force it into a project is not a good way of working. Processes require certain conditions to work well on a project and if these conditions cannot be provided, then the project is driven into the failure. The first mistake is from the management team imposing an unworkable process on the development team given the geographic, logistical, and organizational complexities of the particular development project.

4.2. Case studies using Scrum

This second part is the review of some case studies where different companies used Scrum with their non-Software projects.

Aircraft systems integration projects are the first mentioned case studies, which have been considered an issue of interest due the complexity they present by using diverse domains of engineering. The review basically includes different advices that Carlson

and Tumer extracted from different Aircraft systems integration projects. (Carlson & Turner, 2013)

The Marel GRB case is maybe the most interesting case study because it includes the real point of view of different roles of Scrum. Reynisdóttir, in 2013, realized an internal study in the company and developed it in his Master thesis “Scrum in Mechanical Product Development”. (Reynisdóttir, 2013)

The last case study is SAAB EDS, which has also been extracted from (Reynisdóttir, 2013). This review is also based in an interview to two members of the Scrum team, which has helped to extract some advices about how to apply Scrum with hardware development.

4.2.1. Aircraft systems integration

Carlson and Turner, (Carlson & Turner, 2013), published a review of selected non-software agile case studies, with the intention of finding lessons that are applicable to transform aircraft systems integration process. They state that current aircraft systems integration is similar to the traditional waterfall method in software development.

“In 1993, R. Belie noted that the aerospace industry faced challenges of a rapidly changing world coupled with increased competition and more demanding requirements which threatened to create unaffordable aerospace systems” (Carlson & Turner, 2013, p. 469) . Belie noted that the industry had responded to these challenges by increasing team size and expanding development cycles. The effect of these changes was the opposite, the costs increased and the innovation slowed down. Belie’s proposition was to have small cross functional teams and use computerized tools to increase the ability to iterate. Rapid iteration would facilitate change incorporation and enable the teams to explore a wider range of design options. Belie’s ideas were quite similar to what nowadays we know as agile methods.

In this paper, as aircraft systems integration is still based on a serial development methodology that seeks to address complexity with ever larger teams, Carlson and Turner selected various case studies of agile in non-software industries to “show how to address scalability and draw attention to the need for organizational paradigm shifts as well as the need for a foundational architecture and robust change management” (Carlson & Turner, 2013, p. 470).

The case studies from where they took away some hey lessons were the “Johns Hopkins CubeSat”, “The Nokia example”, “Ozkaya Agile Development and Architecture“, “DOD IT acquisition” and “The Sanova case”. The case of more interest for this thesis is the “Johns Hopkins CubeSat” case, which involved disciplines such as Payloads, Electrical, Software, Mechanical, and Avionics. This case “demonstrates the application of a highly iterative agile process to a complex engineering problem and the resulting benefits increased cycle time and cost as well as increased innovation” (Carlson & Turner, 2013, p. 470).

The key lessons that Carlson and Turner took away from their cases review are the following:

- Having an **architecture** that allows independence of the teams and getting multiple early releases. It is important to formulate the architecture early and document subsequent changes.
- The need of **diverse teams** with knowledgeable members from all areas to encourage shared vision and accountability. Agile methods also work best with **small teams**.
- Follow a rigorous **configuration control process** to keep everyone on the same page.
- **Co-location** can enable teams to facilitate rapid design iteration which enabled flexibility and innovation.
- Having an **Agile coach** for educating the teams on how the process is supposed to work. **Process training** is a worthwhile investment to change the culture, create appropriate expectations, and quickly implement effective methods
- **Prioritize work** helps to support long lead efforts and early architecture decisions. It basically helps to reduce wasted effort and rework.
- **Incremental testing** and **rapid iteration keep** the teams focused on meaningful work and producing functioning objects. This way, deficiencies should be revealed earlier.
- **Prototyping** can reduce the work load on critical areas and get quickly to a tangible product that can provide the best feedback.

4.2.2. Marel GRB

This case study studied a new project developed in 2012 in the industry centre (IC) at Marel GRB. Marel is a leading global provider of advanced equipment and systems for the food processing industry. The IC's development team is composed ox mechanical team and an embedded software team. The case study of this thesis is mainly focused on the mechanical team.

The project consisted in developing a new machine that included a combination of technologies, some which were similar to what Marel was used to develop and others that were new for the company. Only two of the five members of the mechanical team were working on the project full time. The team, before starting the project, did a Scrum training directed by a Scrum coach, who also was the Scrum Master of the mechanical team. This project was set out as an experiment of 7 months, and after that time the team could decide whether or not to continue using the framework.

The Agile Center, which is a competence centre within the Innovation Community at Marel GRB that helps the teams applying Agile, before starting the projected defined the following *Table 2* of things that teams must do, should do and could do meanwhile using Scrum.

Team must have	Team should have	Team could have
• One backlog • One Product Owner • One Scrum Master • Regular rhythm • Sprint Retrospective • Relative Estimating • Col-located team • Sprint of less 3 weeks • Product Owner and Scrum Master is not the same • Sprint Planning	• Product Owner and Scrum Master are not part of the technical team • Use burn-down charts for Releases and Sprint • Visible definition of "Done" • Teams have working contract	• Improvement backlog • Impediment backlog • Definition of Ready

Table 2. Scrum team

After three months of project, the author of the source, (Reynisdóttir, 2013), did an interview with the Scrum Master of the project, and after six months the team and the Product Owner were also interviewed. Following there is a summary of the most important concepts.

The Scrum Master

First of all the Scrum Master said she was happy with the results until that moment, the team might not be following the framework by the book, but she thought that Scum had proven to be effective for that project.

During the implementation of the framework, the Scrum Master noticed that the members were used to work as individuals, not as a team. She also noticed that the project was large and complicated and that the Product Owner was very busy and not always available. She felt that the Product Owner didn't really understand their role, which is mainly to prioritize the work of teams. For this reason she agreed that a *better training for the Product Owners and deeper knowledge about Agile methods was needed*.

Some recommendations that the Scrum Master gave before using Scrum were:

- The Product Owner and the Scrum Master can **never be the same person**.
- There will always be **Reviews** at the end of Sprint. She stated that once a team has to stand up at the Review and say they have nothing to present, teams rarely do that twice. This pushes them to plan better and be more realistic about what they commit to.

She noted that Scrum is well suited in cases where the uncertainty is high, the team doesn't know clearly the work they have to do and then it is difficult to plan far ahead. The reason way Scrum suits in these cases is because it is an iterative process that allows re-planning and re-evaluation of plans.

Finally she remarked that for using Scrum with mechanical teams, in their case, they had needed to *accommodate the variable Sprint lengths* and also *forget about using User Stories*. Otherwise, she agreed that teams should always *use the Daily stand-ups, have a backlog,* and *do planning and tasking stories as well as use relative estimating.*

The Team

The first thing the team mentioned was that the system was really good for keeping track of the project's progress. The fact that they were synchronized with the software team and the product development workshop was seen in positive way. The daily-meetings make sure that everybody knows what everybody is doing and it is difficult to get lost, the communication is much better. They felt that these *daily-meetings are needed.*

The team was very *positive towards the planning meetings*, also known as Sprint planning, and all agreed that they were necessary, although they noticed they were not really doing it by the book.

On the other hand, the *Sprint Review was not so successful*. There was a general frustration regards this meeting. The team agreed too many people were invited to the meeting and few of them showed up. They thought they were getting neither any profit nor any value feedback from the Review. In addition, they didn't feel comfortable with showing unfinished models or discussing technical problems with people from software teams and Product Owners. They suggest that a best way to organize the Review could be either have a Review every other Sprint, were they would have something more substantial to present, or having a big Sprint Review when they have something interesting to present.

About *work breakdown, they all agreed that it was difficult* to break the product and its components further down. The main reason is that they did not have the final and full view of what the product and that made the work harder. The team noted that they were demanded to do a very detailed breakdown of work-packages and stories, and then it might end up not to be used.

Another issue the team didn't see the point was how the Product Owner or anyone else could create *a backlog of tasks for the team*, which they would have to use. According to the team, the backlog has to be technical and not user oriented, for this reason in mechanical development *the Product Owner shouldn't create the backlog items, the team should do it*.

In summary, the team claimed that new framework had some advantages such as: the **Daily stand-up** meetings, **breaking down the functional** aspects of the machines, giving **clarity and an overview** of the project, better **communication and co-operation**, providing **focus** for the team and being able to **plan and discuss** together.

Otherwise, they also thought it had some disadvantages like: the extensive and useless **Sprint Reviews**, the **time consumed** during the meetings and the hard work that the **work breakdown** supposed, besides to find it unnecessary. This way, the team agreed that these issues needed a change in order to keep working with Scrum.

Product Owner

The Product Owner, who in this case was also the Industry Centre's (IC) Product Development Manager, mentioned some problems related with the team. He noted that the team did not take really care about the components from suppliers that were critical to the success of the project. They just trusted that everything would be delivered without problem, so he had to take care of it. Actually, he remarked that it *was unclear who should do this type of work.*

He also mentioned that there were some discussions between those mechanical engineers that worked together due to the fact they used to work alone.

The Product Owner stated that was most valuable for the team was *planning and communicating internally*. *The overviews* from the daily sand-up were also good for him and for the team. These meetings allowed them to address risk factors, try to see what could go wrong.

On the other hand, he said that the new framework had caused negativity within the team, basically because they were taken out of their comfort zone. And, unlike the team, he affirmed that the Review meetings were really good for asking things that otherwise would never been asked. He claimed that mechanical engineers are not used to challenging each other, and for this reason they didn't feel comfortable.

Summary

At the end, all of them agreed that it was certain that they would not have come this far without Scrum. Basically, the team would not have accomplished the goal of sending all parts of the prototype to production by the set deadline if they had been working in the old way.

In conclusion, when all of them were asked if **Scrum could work with mechanical development**, all of them agreed that **yes, but it needs to be adapted**. They also noted that the adaptations they would need, do might not work for all the projects.

4.2.3. SAAB EDS – Scrum-like Method in a Hardware Design Project

This case study presents how a multidisciplinary hardware team at SAAB EDS in Gothenburg, Sweden, used a Scrum-like method in one of their design projects. The project had competences of power and cabling, mechanical and microwaves, and there was no software competences involved. The project involved twelve people, but just three of them were fully dedicated to the project. It was planned to be 22 months long, using Sprints of three weeks; and the product was built up about eight parts, where each part had to go through the phases of design, build of prototype and verification separately.

As mentioned before, Scrum method consists in breaking down work and writing it on notes. This type of work was not fully new to the team; they have worked with visual planning before in other projects.

All the information extracted from the source (Reynisdóttir, 2013) comes from an interview of two employees of SAAB EDS. The interviewed employees were the Line Manager and the Project Manager.

Scrum implementation

Before starting a Sprint, the team had two different meetings. The first one was attended by the Project Manager, Product Owners and one or two team member representatives from each of the different competence areas. There they decided the work items for the next Sprint. The second meeting was attended by all the team, and there they estimated the required time for the work items. If the estimated time was more than the allowed capacity, the remaining work items were left for the "Next Sprint". The Project Manager explained that *they were never able to finish all the work that they had planned; normally 10% of the planned work was left*.

One of the most interesting things of this case study is the re-evaluation of priorities. In the beginning the team was taught that the priorities that were set for each story in the beginning should be kept throughout the project. But that was not their way of work, as they have re-evaluated the importance of the stories in between the Sprints. Furthermore, there were planned stories that became irrelevant during a Sprint because of unexpected events and so they were down prioritized.

There were also other reasons why they had to re-evaluate priorities or change the order of work. Sometimes some parts had not been delivered on time for the team's

verification and therefore there were people waiting who did not have work to do in that Sprint. The solution *was looking for any work, planned for the next Sprint, which could be done during the current one.*

About getting people write notes regarding their work, the Project Manager noticed that it was hard because it meant extra work. After some time, they realised that *it was useful to put down in words what one was going to do*. Moreover, they were also working with a company in India, so having a description of the work which needed to be done was a benefit.

Team's profits

The commitment of the people and the fact they were planning together and more closely, allowed people to have clearer they work. Each member of the team knew what everyone was doing, and if the team members had questions regarding the work items, it was possible to change their description to be clearer and more precise. In addition, if the management had too high goals the team was able to discuss that and adjust the goals accordingly to the current capacity.

According to the Line Manager despite the profits, there was still a problem of people being dispersed over several projects. He noticed that the benefits of Scrum would have been greater if people had been 100% dedicated to the project.

Suggestions or guidelines

At the end of the interview, the interviewed employees agreed that there were some suggestions that would make profitable the way of working.

The Project Manager noticed that it would be better to write detailed goals for the backlog items. The items have to be well defined, and a definition would let clearer what needs to be done for the item to be finished. The Line Manager agreed that is important to find a good Scrum Manager, as that person is going to make the team or system work. It is important that the Scrum Manager can teach the team how to use Scrum. If the person does not have the ability to communicate and work with people, the meetings could turn out badly.

5. Conclusions

As it has been seen, the rapidly changing world with increased competition and more demanding requirements is the main reason for needing agile approaches. The main purpose of this thesis was to conclude if agile approaches were able to be applied to other engineering disciplines, beyond software development, focusing preferably on the domain of mechanics and mechatronics.

After analysing the available sources and case studies, I think that we can conclude that **agile approaches can be used with other engineering disciplines**, but they need to be adapted. However the main problem is that in mechanical and mechatronic engineering each project is really different from the rest, and then it is quite difficult to establish some practices or rules which would help all the projects to succeed.

Although we cannot establish a standard guide, I believe that some good advices have been extracted from this analysis. Moreover, I would say that there are some advices that could *always* benefit the mechanical and mechatronic teams, despite most of these advices have been deducted from Scrum approach.

First of all having an Agile coach for educating teams with the framework would always be the best way for starting with any agile approach. Small and diverse teams which attend to daily or regular meetings would also improve the communication between the team, this way everybody knows what everybody is doing.

There are also some suggestions for facing up the multiple problems which can come up during the DBT cycles within the mechanical and mechatronic projects. Establishing more modularity in those places where change is expected, getting as soon as possible a prototype and having a good relationship with the shops which supply the resources, can help the team to avoid several problems.

Finally, the feedback from the customer is really important, but we also don't want to inconvenience them. On the suggestions, it has been scheduled a customer's call once during the sprint and update them with the completed work. Once the team has a prototype that can be delivered to the customer, the involvement of them can be amplified.

One important thing that I think that needs to be specified is that besides agile approaches may be applied with mechanical and mechatronic projects, it doesn't necessarily mean that these are the best frameworks for some special projects.

Sometimes if the products are in a highly regulated industry, agile approaches may hinder the project.

The first mistake from the management would be imposing an unworkable process or an inadequate framework on the development team. In these situations the fail can be associated with the framework, but that would be an error, given that the first and most important decision before starting a project is to **decide with which methodology the project would be optimized and would succeed**, not just avoid the fail.

6. References

Manifesto for Agile Software Development. (2001). Retrieved from http://www.agilemanifesto.org/

Dr.Dobb's. (2010, August). Retrieved June 11, 2015, from The world of software development: 2010 IT Project Success Rates: http://www.drdobbs.com

Linkedin. (2013, November). Retrieved May 18, 2015, from Agile for hardware and mechanical?: https://www.linkedin.com/grp/post

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods: Review and analysis.* Oulu.

Agile Alliance. (2002, May 1). *Agile Manifesto*. Retrieved from http://www.agilealliance.org

Awad, M. (2005). *A Comparision between Agile and Traditional Software Development Methodologies.*

Carlson, R., & Turner, R. (2013). Review of Agile Case Studies for Applicability to Aircraft Systems. *Systems Engineering Research*, (pp. 469 – 474). Atlanta.

Contributor Melonfire. (2006, September). *Understanding the pros and cons of the waterfall model of software development. Techrepublic*. Retrieved from http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/

Dingsoyr, T., Dyba, T., & Brede, N. (2007). *Agile Software Development: Current Research and Future Directions.* Springer.

Dr.Dobb's. (2014, February). *Dr.Dobb's*. Retrieved June 11, 2015, from The world of software development: The Non-Existent Software Crisis: Debunking the Chaos Report: http://www.drdobbs.com

Dyba, T., & Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. (pp. 834-842). Trondheim: SINTEF ICT.

Graves, E. (2015, January 1). *PLAYBOOK*. Retrieved June 2015, 1, from http://playbookhq.co/blog/agileinhardwarenewproductdevelopment/

Grugl, L., & Dayley, A. (2013, July). *Google groups*. Retrieved from Scrum alliance: https://groups.google.com/forum/#!msg/scrumalliance/gAYsCdCQNoM/Zjftgko2XwYJ

Highsmith, J. (2009). *Agile Project Management:Creating Innovative Products.* Addison-Wesley Professional.

ISTQB exam certification. (n.d.). *What is prototype model advantages disadvantages and when to use it*. Retrieved from http://istqbexamcertification.com

Jackson, M. B. (2012, April 1). *Read periodicals* . Retrieved June 13, 2015, from Agile: A decade in: http://www.readperiodicals.com

Mohammed, N., & Govardhan, A. (2010, September). *A Comparison Between Five Models Of Software Engineering.* Retrieved from IJCSI International Journal of Computer Science Issues Vol. 7, Issue 5.

Reynisdóttir, Þ. (2013). *Scrum in Mechanical Product Development.* Gothenburg.

Royce, D. W. (1970). Managing the development of Large Sotware Systems. *IEEE WESCON 26(August)*, (pp. 1-9).

Schwaber, K., & Sutherland, J. (2011). *The Scrum Guide.* scrum.org.

Software developers. (2001). *Manifesto for Agile Software Development*. Retrieved from http://www.agilemanifesto.org/

Stober, T., & Hansmann, U. (2010). *Agile Software Development: Best Practices for Large Software Development Projects.* Böblingen: Springer.

Stoica, M. (2013). Software Development: Agile vs. Traditional. *Informatica Economică vol.17*, (pp. 64-76).

Ullah, M. (2014). *Comparison and problems between Traditional and Agile software.* Lappeenranta.

Wikipedia: The Free Encyclopedia. (n.d.). Retrieved from https://en.wikipedia.org

7. Annex

7.1. Interview of agile authors

The following interview has been extracted from (Jackson, 2012).

Publication: PM Network
Author: Jackson, Michelle Bowles
Date published: April 1, 2012

BACK IN 2001, lightweight processes came together to 17 advocates of lightweight draft and sign a now-legendary document, the Manifesto for Agile Software Development. Now, more than a decade later, four of the original authors discuss the agile manifesto's impact on project management, what has changed, what has surprised them- and where they'd like to see the approach go in the future.

- ***How has agile impacted project management in the past decade?***

Jim Highsmith: It has shown there is a range of project types and different strategies. In particular, projects that involve uncertainty, varying requirements and shorter delivery times can benefit from agile methods. Projects in which innovation and experimentation are required often in conjunction with uncertainty- also need to be managed in a more iterative, product-oriented way rather than a task-oriented manner. Agile project management embraces both "doing" agile and "being" agile- and the latter is the hardest It defines a different management style: one of facilitation, collaboration, goal- and boundary-setting, and flexibility. Finally, agile is changing the way organizations measure success, moving from the traditional iron triangle of scope, schedule and cost to an agile triangle of value, quality and constraints.

Ron Jeffries: More organizations recognize that conventional project management is not the best way for them to do their work. They are finding the assumptions behind conventional project management are not germane to the business of creating value. In particular, it is common for a project management approach to focus primarily on cost rather than value. It is common to attempt building some defined thing by some defined deadline. Agile ideas turn both assumptions on their heads. Projects should meet deadlines by controlling scope and should control scope by managing which chunks of value should be included and which should not. Agile is not about building a plan and working to it. It is not even about building a plan and modifying it as we go. Agile is not about planning at all; it is about choice. Many companies are learning this. Some project managers are learning it as well.

Andrew Hunt: Agile probably has not impacted project management as much as it should have. Primarily, the agile movement has impacted development and developers more than management and managers. I hope, at least, it has helped point out what project management should be doing - that is, removing those things that are blocking the developers.

Arie van Bennekum: It is not the manifesto but the agile movement that has impacted project management. The big difference is the way you report to clients on progress, do acceptance procedures and create ownership. As a project manager, you do not control a team tofacilitate it. And this team includes end-users.

- ***How has agile evolved during the 10-ptus years after the manifesto?***

Mr. Highsmith: It has evolved in several different ways: First, early agile projects were smaller and collocated. Now organizations have scaled agile to very large projects that most often are distributed. Second, organizations now focus on agile technical practice in software development and are learning to implement continuous delivery in which new features can be deployed at any time (daily, weekly, etc.), Third, agile software development has evolved into agile project management that covers a wide range of products.

Mr. Hunt: I don't think agile has evolved a whole lot, which proves everyone missed the point. Agile is supposed to be ever -changing and ever-adapting to the context and project at hand - and it hasn't been. Instead, we've seen wide-scale adoption of some better software development practices, but we haven't seen any significant adoption of true agile behavior. Remember, you can't do agile; you have to be agile. That distinction has been lost over the last decade.

Ron Jeffries: The fundamental ideas we expressed in the agile manifesto have held up amazingly well, at least with the authors and, we believe, with everyone who understands what we were and are talking about. Naturally, we have refined our understanding of exactly how to go about effectively putting in place those ideas. But so far, none of us feels the need to drop any of those values and principles, nor plug in any new ones. What's been most surprising regarding agüe and its effect on project management?

Mr. Highsmith: I've been surprised that it has taken the project management community so long to seriously engage with the agile community. That said, I think the project management community can have a significant impact on the wider use of agile methods in organizations.

Ron Jeffries: Frankly, I have been most surprised at how tenacious the old ideas have been. Adoption of the full spectrum of what we were talking about remains rare. There are too many entrenched stakeholders who cannot see how their personal situations will be enhanced by going in the agile direction.

Mr. van Bennekum; It has been surprising to me that the same issues keep coming back. During the '90s, the agile project delivery framework, dynamic systems development method (DSDM)1 was growing in Europe, and we ran into all kinds of problems, such as how to prioritize, test and manage a steering committee or management team to adapt to an agile decision-making process. From the DSDM perspective, those problems were solved, but at the moment, they are coming back again on most of the agile projects and agile implementations I see.

- ***What is agile's place outside of software development?***

Ron Jeffries: The key ideas apply everywhere. These include, but are not limited to:

- Putting people with needs in direct contact with people who can fulfill those needs
- Populating projects with all the needed people and capabilities to get the job done
- Building work incrementally and checking results as you go
- Preparing for and influencing the future but not predicting it
- Making tasks concrete and quickly finishing them
- Giving people work to do and the knowledge to do it, not pushing them around like pawns on a chessboard
- Focusing on providing value frequently and rapidly, not directly on cost

Agile ideas are based on how people and organizations work best. There are specialized details we need to know regarding software, just as there are in any other domain. The principles, however, are broadly applicable.

Mr. Highsmith: In any project that faces uncertainty, complexity, volatility and risk, there is a place for agile practices and principles.

Mr. van Bennekum: Agile is holistic and applicable everywhere in business or life. I use it as a concept wherever I am and for whatever I do, from defining online strategies to the total refurbishment of my house.

Mr. Hunt: The agile mindset is critical to successful business in the 21st century. At its heart, an agile approach has little to do with software; it's all about recognizing and

applying feedback. The definition of "agile" that Venkat Subramanium and I proposed in Practices of an Agile Developer states, "Agile development uses feedback to make constant adjustments in a highly collaborative environment." Notice there's nothing in there about software. The Pragmatic Bookshelf publishing company, in fact, has been called an "agile publishing company" by our many fans. We seek to embody the principles of agility throughout our business practices. Charles Darwin famously said, "It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change." We have become keenly aware of this through our work with software. But rapid adaptation is the single most important idea of this century for any business, in any market.

- ***Where would you like agile to go in the future?***

Mr. Jeffries: I would like to see more people working toward a true understanding of agile - which only can come about by actually doing it and experiencing it It is a way of working that leads to a way of thinking. Entering into it theoretically is tempting, but people's theoretical imaginings do not amount to real understanding.

Mr. Hunt: I'd like to see agile go where we thought it would go 10 years ago; with a flourishing ecosystem of new ideas, new development practices, new languages, new methodologies and new ways to satisfy the customer with working code that's easy to modify and evolve to suit the rapidly changing needs of modern business. We've seen some modest individual successes in each of these areas but not exactly a flourishing.

Mr. van Bennekum: Agile will become more important because of its ability to respond to change. Product life cycles are much shorter and businesses are continuously changing, which requires agile. Agile especially will grow in portfolio management and business management because we cannot do without it. The most healthy organizations will be agile organizations.

- ***How is the recent PMI Agile Certified Practitioner (PMI-ACP)SM credential helping the progress of agile?***

Mr. van Bennekum: People use the word "agile" sometimes without any sense of what it really is. For people hiring consultants, certification will become a criterion.