Politecnico di Torino

POLITECNICO DI TORINO

Master's Degree in Engineering and Management

Master's Degree Thesis

Agile Product Development outside Software Development

Supervisors

Prof. Francesca MONTAGNA

Prof. Marco CANTAMESSA

Candidate

Stefano GALLO

April 2022

Abstract

During the recent years, hardware product development has changed. More and more complex products arrived on the market, often including sensors and software, collecting data and elaborating them. The customers have changed too, and the demand for a constant release of new and customizable products has increased. In this framework, the traditional product development methods may not be adequate. One of the possible solutions could be represented by the adaptation of agile software development methods to the hardware domain. The aim of this thesis is to investigate such a possibility. After a first chapter focused on the historical background of agile methods for software development, the second chapter goes one step further and discusses agile's state of the art outside their original domain. This literature review permits to identify some recurring trends in modern product development, such as the use of prototyping, the importance of customer requirements, and other practices that get hardware development closer to agile software development. These trends provide positive effects in reducing documentation, allowing for later changes in requirements, and increasing knowledge and transparency in multidisciplinary teams. Some limitations are identified as well, such as the impossibility of a direct application of agile methods to hardware development without a dedicated fine-tuning process. Finally, the third chapter presents an analysis performed on a series of case studies, aimed at validating the results obtained in the previous chapter. By examining real-life applications of agile methods on larger scale projects, it was possible to confirm their potentiality, even if some challenges are still present.

Table of Contents

List of Tables

List of Figures

Introduction

The goal of this paper is to understand how agile methods, that were originally designed for software development, can be adapted to other kind of development processes, and how they can be implemented in other industries.

During the recent years, in fact, hardware product development has changed: more and more complex products arrived on the market, often including sensors and software, collecting data and elaborating them. The customers have changed as well, and the demand for a constant release of new and customizable products has increased. In this framework, the traditional product development methods may not be adequate. One of the possible solutions could be represented by the adaptation of agile software development methods to the hardware domain.

In the following chapters, agile will be presented starting from its origins. First, an historical overview on how and why such a philosophy emerged is provided, starting from the agile's predecessors description, and continuing with the Agile Manifesto's analysis (Section 1.2). Following this line of thought, the main agile software development methods originating from it are then described and discussed, with a particular attention to the main two: Scrum (Section 1.3.1) and Extreme Programming (Section 1.3.2).

The following step goes instead to the hearth of the question: *can such methods be beneficial to the hardware product development as well?* Through a literature review, some recurring trends in modern product development are identified and compared to the possibilities of agile in these contexts. In this analysis, pros and cons are listed and discussed, in order to identify the boundary for agile's application. In a similar way, some areas where agile could be better applied are

determined starting from Section 2.1.2. As a partial conclusion, the numerous contributions will be subdivided according to their nature (descriptive, normative, or mixed), to provide a synthetic and organized overview on their observations. From page 65 onwards, in fact, a short recap in tabular form is provided to the reader.

Finally, a deeper analysis of six mayor case studies will be presented. By examining real-life applications of agile methods on larger scale projects, an attempt to validate the previously obtained results was made. As we will see later, some of the preliminary conclusions were confirmed by a number of case studies, while others did not reach the same result. Additionally, new contributions emerged from the direct observation and implementation of such theoretical concepts into industrial and experimental projects.

Chapter 1

Historical background of agile

In order to investigate agile methods' future and their possible usages in the hardware domain, a general introduction to such methodologies in their original form is needed.

In the following section, an historical view on agile's predecessors is provided, it can be useful to understand how and why agile became so popular. After this preamble, agile will be described by analyzing the original Agile Manifesto. Finally, a description of its practical applications will be provided, together with a more detailed view on a series of specific agile methods.

1.1 Before agile

Agile came out as a reaction to the traditional way of developing software or, using the very words of the Agile Manifesto creators, as "an alternative to documentation driven, heavyweight software development processes" (Beck et al., 2001). With traditional methods, the development process used to start with the research and documentation of a very detailed set of customer requirements,

followed by design, development, and inspection phases. Starting from the mid-1990s, these first steps were identified to be frustrating and sometimes impossible to follow (Cohen et al., 2004), and the reasons were many. First, documenting customer requirements in such a detailed way could take months or even years, delaying the real development process. Secondly, the industry was evolving at a different pace, so requirements were always changing, in a much faster way than what traditional methods could manage. Third, also customers – as a consequence – became more demanding, asking to modify requirements more often; they became unable to define requirements up front, in a definitive way, as these methods demanded. Agile solved many of these issues, but it was not a disrupting invention: most of its practices come from other basic principles and are held together by sharing a series of principles and values, that will be analyzed later on.

One of the first traditional software development methods was the *waterfall method*. It was built on a simple idea: dividing the software development process into two phases for small projects – analysis and coding – and into seven consecutive phases for larger projects (Royce, 1987). In the analysis phase, however, the waterfall method involved the traditional research and documentation of customer requirements that could last several months. The purpose was to document everything, then provide the collected information to the designers and the engineers, asking them to realize the customer needs. This kind of method worked really well in certain cases, but struggled with changes. In real life, requirements kept changing even after years of work, and also when the development phase was already started. Due to its approach, any change in the waterfall method meant that everyone involved in the development had to meet in a room, discuss the issue, and document it, causing a further increase in documentation and time consumed. The waterfall method tried to avoid changes in requirements by freezing them at the beginning of the process, but this idea collided with the real world necessities (Cohen et al., 2004). The main issue with the waterfall method is, in fact, its tendency to focus on a big, long-term goal – the final product and its major features – while requirements, especially in IT projects, are changing very rapidly (Cooper, 2016). B. Reagan (2012) describes this matter as follows: "it's hard to alter course when you're being swept down a large waterfall [...]. Too much up-front planning means too much change management downstream". In other words, focusing too

much on the final product gives origin to multiple unwanted outcomes like long feedback loops, replanning, and the need of reaching compromises. This results into longer and inefficient development cycles; products that do not completely satisfy the stakeholders; higher costs caused by the additional time and effort needed to implement changes and replanning.

For sake of completeness, it is worth mentioning how the waterfall method is not the only traditional method with this kind of drawbacks. Slight modifications to the waterfall model were brought by incremental and iterative techniques. The incremental ones suggested breaking the project in smaller increments and apply a traditional waterfall approach to every increment. This meant that the customer requirements' heavy research and documentation was kept in place, but the requirements were then analyzed separately as stand-alone functionalities, allowing different teams to work on them simultaneously. This allowed to reduce the development times, thanks to concurrent multitasking. Similarly, iterative techniques subdivided projects in "iterations of variable length, each producing a complete deliverable and building on the code and documentation produced before it" (Cohen et al., 2004). The big difference between the two techniques, however, is that with the iterative ones, change is not a problem. Every iteration has its own research phase and its own requirements to fulfil. It is not necessary to have a complete set of customer needs at the beginning of the project, every iteration can apply a smaller waterfall model to implement the desired features into the deliverable. The following iterations will then do the same, adding features to it, according to the new and updated customer needs. Starting from these ideas, several methods were developed as extensions of the classical waterfall method. Among them, the most popular ones were the *V-cycle model* and the *Spiral Model*, which however still found difficulties in supporting both the collaboration between different designers from different disciplines, and the integration between hardware and software (Mabrouk et al., 2018).

Nevertheless, this way of developing software represented a great move towards the agile methodology and away from the traditional waterfall method. Additional contributions came by analyzing a variety of existing methods, approaches, and techniques, both in their strengths and in their weaknesses. As an example, Ken

Schwaber (one of the seventeen agile manifesto's signatories and co-developer of Scrum, probably the most successful agile technique) focused his attention on the Capability Maturity Model (CMM). This model suggested companies a series of processes and goals useful to pass from the first level of maturity (*chaotic*) to the fifth one (*optimized*). He studied the model, and he understood that although CMM tried to turn software development into a series of repeatable, defined, and predictable processes, it was still made of a set of largely unpredictable and changing processes (Cohen et al., 2004). Schwaber realized how, to be truly agile, a process needed to accept change as quickly as it arose, and that in a dynamic environment "creativity, not voluminous written rules, is the only way to manage complex software development problems" (Highsmith & Cockburn, 2001).

Another source of inspiration was found in the engineering and manufacturing world. Mary Poppendieck and Bob Charette focused on *Lean Manufacturing*, an incredible invention of the second post-war period by Toyoda Sakichi, that did not gain popularity in the United States until the 1980s. Toyoda's idea was to keep the level of supplies and inventory as low as possible (the bare minimum to run the plant for one day), only producing enough products to fill existing orders. According to Poppendieck (2001), lean manufacturing is made of ten basic principles:

1. Eliminate waste
2. Minimize inventory
3. Maximize flow
4. Pull from demand
5. Empower workers
6. Meet customer requirements
7. Do it right the first time
8. Abolish local optimization
9. Partner with suppliers
10. Create a culture of continuous improvement

This whole framework was made of reciprocal influences between practitioners and engineers all over the world. As an example, Toyoda also integrated into his philosophy Dr. W. Edwards Deming's *Total Quality Management (TQM)* principles

and, in turn, Toyoda's work inspired Charette, that few years later presented the so-called *Lean Development* method.

Similarly, two other agile manifesto's signatories, Kent Beck and Ron Jeffries, rediscovered these principles in the late 1990s and gave origin to Extreme Programming (XP), another extremely successful agile method. These examples show how, during those years, many researchers focused their attention around new software development methods, often reaching similar conclusions and giving origin to comparable results. As Scrum and XP were developed, in fact, also others were doing the same, such as Alistair Cockburn developing the *Crystal Method*. It was just a matter of time before they realized how a new philosophy was born and how important this could have been for the software industry. With these premises, in 2001 "a Manifesto for Agile Software Development" (Beck et al., 2001), commonly known as agile manifesto, was formed and it can still be found in the original website http://agilemanifesto.org. The manifesto emphasizes small, co-located and self-organizing development teams working close to each other, taking advantage of frequent feedback gathered from close customer collaboration, and embraces change (Larman, 2004). In the next section, a deeper view of the manifesto is provided.

1.2 The Agile Manifesto

As the authors themselves described their 2001 meeting, "[a] bigger gathering of organizational anarchists would be hard to find." Continuing to cite the manifesto's history webpage,

> [...] what emerged was the Agile 'Software Development' Manifesto. Representatives from Extreme Programming (XP), SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming, and others sympathetic to the need for an alternative to documentation driven, heavyweight software development processes convened. [...] the Agile movement is not anti-methodology, in fact, many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and

> rarely used tomes. We plan, but recognize the limits of planning in a turbulent environment (Beck et al., 2001).

Apart from these high-sounding premises, the *manifesto* is presented in a very simple and concise way, as one can verify by accessing its original website. On the main web-page, in fact, it can be found a brief introduction followed by its four main values, as shown in Table 1.1.

We are uncovering better ways of developing software by doing it and helping others 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.

Table 1.1: Agile **values** presented by the Agile Manifesto (Beck et al., 2001)

In his article, Glass (2001) provided a comparison between agile and traditional methods, by analyzing each of these values in detail. It is interesting to look at how this new paradigm was welcomed, but also criticized, by the other practitioners at that time. About *"individuals and interaction over processes and tools"*, Glass believes that the Agile community is right, because traditional software engineering was gotten too caught up in its emphasis on process. However, he also states that "most practitioners already know that people matter more than process" (Glass, 2001), as if agile was not adding anything particularly new to the industry. Going on with the list, about *"working software over comprehensive documentation"*, Glass agrees with the agile community, saying: "It is important to remember that the ultimate result of building software is product. Documentation matters [...] but over the years, the traditionalists made a fetish of documentation. It became the prime goal of the document-driven lifecycle" (Glass, 2001). About *"customer collaboration over contract negotiation"*, Glass instead agrees with the agile manifesto on the importance of customer collaboration, but he also highlights how contracting should not be underestimated: "I deeply believe in customer collaboration, and [...] without it nothing is going to go well. I also believe in contracts, and I would not undertake any significant collaborative effort without it" (Glass, 2001). Finally, when commenting the last agile value, *"responding to change*

over following a plan", he tells about two contradictory lessons learned in the past years: "customers and users do not always know what they want at the outset of a software project, and we must be open to change during project execution [...] requirement change was one of the most common causes of software project failure" (Glass, 2001). In this way, again, he expresses how in his opinion the agile values are correct on the left side, but not necessarily on their right side.

Going on with the manifesto's description, it is important to know how, starting from the agile values, twelve "principles" were developed. In a second webpage, in fact, a longer and more detailed list contains the manifesto's twelve principles, here reported in Table 1.2.

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 primary 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 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.

Table 1.2: Agile **principles** presented by the Agile Manifesto (Beck et al., 2001)

This list is of course of extreme importance for the agile movement, but as agile evolved during the years some critical thoughts on it emerged as well. As an example, I found Meyer's comments very interesting and useful for people willing to understand and apply agile in real life projects. In his book "Agile", in fact,

Meyer points out how some of these principles are instead practices, platitudes, assertions, and in general how this list is incomplete. To cite some examples, he believes that principles number six and twelve can be defined as practice; number five and nine are too obvious and don't provide any extra help to the developers; principle number 10 is, in his mind, completely wrong since "maximizing work not done" and seeking simplicity are two important principles but are just not the same thing; and the list continues further (Meyer, 2014).

To solve this issue, Meyer created a "new" list, with a teaching purpose and not to replace the original one. This new list subdivides the principles in two categories - organizational and technical - for a total of eight new principles, three of which also have few sub-principles:

Organizational
1. Put the customer at the center.
2. Let the team self-organize.
3. Work at a sustainable pace.
4. Develop minimal software:
4.1 Produce minimal functionality.
4.2 Produce only the product requested.
4.3 Develop only code and tests.
5. Accept change.
Technical
6. Develop iteratively:
6.1 Produce frequent working iterations.
6.2 Freeze requirements during iterations.
7. Treat tests as a key resource:
7.1 Do not start any new development until all tests pass.
7.2 Test first. 8. Express requirements through scenarios.

Table 1.3: Agile principles according to Meyer (2014)

Although this is not the original set of principles published by the manifesto authors, it is probably more useful to people approaching agile now. The agile philosophy has changed a lot during the years, and it is more meaningful to deal with such pragmatic points than with the original and very vague ones. The outcome is the same, but the effort needed to understand and implement them is surely lower. Some of these principles are the reinterpretation of the original one, others are created by joining more than one principle, and others are brand new,

addressing issues that the agile philosophy values, but the original principles were not able to show.

In his book, Meyer analyses these eight principles in detail, over almost thirty pages. Of course, it would be excessive to do the same in this paper, however it is interesting to focus on some excerpts. Starting from the first principle, Meyer prefers to put the focus on how customers, in agile processes, should be at the center of the whole development process. It is not enough to satisfy their will and deliver frequently, customers must be welcomed at regular project meetings, they should be able to interact with developers and try the new versions, in order to give their feedback.

The second principle addresses the self-organizational nature of the agile teams, in which the manager's internal functions are usually split between the team members. Agile indeed places great trust in the team's ability to organize its own work. This however does not mean that managers are completely absent, they are usually needed because of the company structure, but have a sort of "subtle control consistent with the self-organizing character of project teams" (Schwaber & Sutherland, 2012). This usually means that the management decides what to build and who will work on their project, but let the team self-organize, free from influence and with the fewer possible constraints.

The third principle, according to Meyer's view, refers to the working pace of the agile teams. Programmers should be given working conditions that enable them to deliver their full potential, avoiding pressure and welcoming a calm and respectful working environment.

The fourth one is instead a bit more complex and is the first one being made of three sub-principles. The author summarized multiple agile principles into a very simple sentence made of three words: "develop minimal software". This means that developers should focus on minimal functionality elements, avoiding time losses due to elements that are not needed, or needed by only a few users. These non-essential features could cause delay in the releases, harm the team's focus, create a future maintenance burden, and constrain the future evolution of the software. The minimal functionality requirement could be easily misunderstood and confused with the need of producing "only the product requested", thus Meyer

separates the two in a clear way. According to this second point, developers should not aim at reusability and extendibility, as the traditional methods ask them to; they should simply build something that works now and here, for this specific customer and request. Although this tip looks a bit extreme, its real purpose is to discourage developers in "working too much" i.e., in putting too many checks and always try to handle the most generic case. Agile suggests, on the opposite, to focus on the given tasks and develop the best possible software for that purpose, but this does not mean that developers are encouraged to go against the software development good practices, because such an approach would be detrimental, no matter what. Finally, developers should only focus on coding and testing, according to principle 4.3. This simply means that they should stay away from everything else is rotating around the development process: feasibility studies, transcripts or videos of requirements interviews and workshops, requirements documents, PowerPoint presentations about the future system, emails, design documents, UML diagrams, and so on (Meyer, 2014).

The fifth principle is again very simple in its formulation, but extremely complex in reality, and it also represents the last *organizational principle*: "accept change". Meyer prefers to say that agile *accepts* change, rather than *welcomes* it: "it is one thing to state that change is a normal phenomenon in software development, and quite another to start hoping for more changes. After all, it always causes more work" (Meyer, 2014). Apart from this linguistic discussion, the text focuses on how, in reality, also traditional methods accept change, they simply address it in different ways. According to the author, agile-enthusiastic texts tend to "caricature" the traditional methods, but in real life they do not treat change in a completely different manner. In other words, change acceptance remains a key concept in the agile framework and every agile method addresses this issue in a different way, but this does not mean that only agile methods give programmers the tools to do so.

Continuing to refer to Meyer's work, it is now the turn of the first *technical principle*: "develop iteratively", that means to freeze requirements during the iterations, and produce frequent working iterations. According to this view, every iteration must yield a working system. That system may offer only a small subset of the full requirements, but it must be a functioning system that provides an

end-to-end user experience. This allows the customer to try it and provide feedback. All agile methods use iterations as the basis for their approach, and they all suggest short durations. It is important to point out also the nature of these iterations: their duration is fixed in advance. "If at the end of the allotted time some of the expected functionality is not completed, the functionality gets pushed to the next iteration, or dumped altogether, but the deadline does not change" (Meyer, 2014). This principle is called "time-boxing", it helps to get more realistic predictions, and can also act as a booster for developers: even if they could simply dump a given functionality, saying that it could not be fitted into the allocated time, they usually perceive the time constraint as a challenge, and find a way to deliver on time. Freezing requirements during the iterations, instead, is useful to manage changes during the process. As already discussed, change is accepted, but it can not be welcomed in every case, it should be properly handled. Usually this is done by preventing changes during an iteration, in order to allow developers to work with solid bases on the small iteration, and think about changes in the following one. If changes were to be allowed anywhere during the project life, developers would struggle in adapting too often to the new requests, and the whole development would result into a messy process.

The seventh principle suggests treating tests as a key resource, and agile methods indeed consider tests a central resource of any project. The suggested method involves, firstly, to not start any new development until all tests pass. It is more important to look at the integrity of what has been produced, than the addition of new elements. Although it might seem obvious that it is better to wait until *all* tests are passed, software developers often encounter small bugs that could question this idea. According to agile, any small defect should pause the project progress, allowing everyone to focus on it and solve the issue. On larger projects, however, this approach could be softened by defining different levels of importance for bugs, deciding that only over a certain importance they could cause the project to stop. The second point is instead apparently simpler: test first, i.e., never write code without first writing a test that exercises it. However, it is more than that:

> Some functionality is not present yet, and you want to add it. Instead of thinking about it in the classical style of defining requirements, write a test

> for it, and — this is the surprising part — run that test (after adding it to the regression suite). The test should fail, since the functionality is not yet supported. Then fix the code until the test passes (Meyer, 2014).

This idea could seem to provoke unnecessary, time-consuming, extra steps. According to its supporters, however, writing a test first forces the developer to imagine a usage scenario for the desired feature. If this scenario is not found, then the feature was not needed, and so is the code that was going to be developed. In this way, test first is actually saving time.

Finally, the eighth principle asks to express requirements through scenarios. Agile, as already stated, avoids big upfront requirements, but developers still need requirements. These are usually pictured by means of user stories and use cases, both describing typical interaction scenarios between the system and an imaginary user. The difference is that a specification is usually general, while the use case/user story needs to be very specific and tell the developer what should happen in a given case.

1.3 Agile methods

After having analyzed values and principles that agile enthusiasts share, it is worth taking a look at the most popular agile methods. According to Sommer et al. (2015), at least nine different methods have been developed: Scrum, Crystal, Extreme Programming, Adaptive Software Development, Agile Modeling, Dynamic Systems Development Method, Feature Driven Development, Internet Speed Development, and Pragmatic Programming. Other authors also include in this list different methods such as Lean Software, Lean Development, Dynamic Systems Development Methodology, and others. Over the last twenty years, these methods have had very different outcomes. While Scrum and XP are the most popular ones, many other agile methods have suffered a much lower implementation rate in the industry. Among the ones that at the beginning of the 2000s appeared to be revolutionary and granted of future success, many had to face the reality and leave room for the very few methods that are still in use nowadays.

Nevertheless, it is worth taking a look at a wide set of agile methods, to understand how the agile philosophy has been interpreted by different practitioners. All the agile methods have, in fact, their foundations on a set of recurring themes: incremental development, close collaboration, frequent deliveries, accepting change, and working in self-organized teams.

Working iteratively and in short cycles, agile teams are able to provide new software releases in a quicker way, compared to the traditional approaches, adding new features every time. In this way, it is always possible to introduce changes in the projects, since there is not a pre-planned route to follow *a priori*. This is one of the keys to agile's success, as it enables developers to follow the always-increasing demand expectations from customers, as well as the rapid evolution of technology and its rising complexity. The descriptions of agile methods are countless. In addition to the above ones, Cooper (2016) points out how agile exploits adaptive planning and a time-boxed iterative approach. Mabrouk et al. (2018) speak about its tendency to involve the various stakeholders, partners, and customers, reducing the interfaces' rigidity that characterizes traditional approaches. They also highlight other key aspects: the lead time between two deliverables is fixed and unchangeable; the high frequency of iterations allows a reduction in "time to market" while improving the quality of the product delivered; the waste minimization by iterative and incremental work allows the development teams to reach significant productivity levels in a faster way. Böhmer et al. (2017) summarize their literature review expressing how agile's goal is to have a functional product at any time, starting from a simple version of the product, which roughly describes the customer requirements and has minimal functionality, and advancing its development step by step adding new features. In this sense, the product increments need to be measurable and evaluated by the user, in order to improve it further. Going on with this process, as the product becomes more concrete, the project becomes more immobile, since the range of options available decreases at every step: "the target area of the planned solution usually changes due to gained knowledge; the obfuscation and fuzziness decreases as the project progresses" (Böhmer et al., 2017). This concept is visually represented in Figure 1.1.

Finally, Könnölä et al. (2016) summarize agile's main advantages in terms of

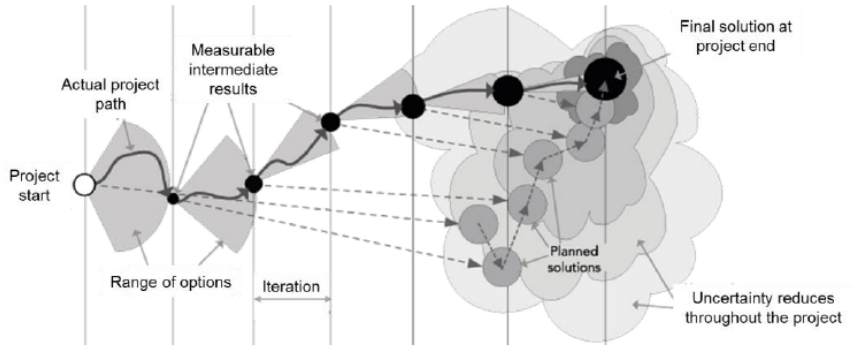

Figure 1.1: Agile project path (Oestereich & Weiss, 2008)

productivity and wellbeing as it follows:

1. Ability to manage changing priorities,
2. Increased team productivity,
3. Improved project visibility,
4. Improved team morale/motivation,
5. Improved team communication and coordination,
6. Enhanced ability to adapt to changes,
7. Increased productivity,
8. Capacity to deliver releases quicker.

As previously mentioned, in order to better understand agile methods, in the following sections a set of agile methods will be investigated with more detail.

1.3.1 Scrum

Scrum is with all probabilities the most well-known agile method at the moment. It has been developed between 1995 and 1997 by Schwaber (1997) and Jeff Sutherland, following Takeuchi and Nonaka (1986) study on six technology driven multinationals that adopted a new, holistic approach in their product development processes. So, it can be said that its origins are not in the software development world, and as it will be described later on, also other agile methods share this characteristic.

In order to understand Scrum, we could use one of the first definitions given by Schwaber (1996) himself: a process that "accepts that the development process is

unpredictable". In a certain way, he formalized the "do what it takes" mentality. Following, instead, the analyses of other practitioners, Scrum can be defined as a method with the purpose of "managing software development processes in volatile environments". In particular, the main concept behind Scrum is the frequent contact between the developer and all the stakeholders involved in the project, with features implemented in small sprints to foster communication (Cooke et al., 2012).

Going more into details, Scrum achieves this goal assigning only three roles in every project: Product Owner, Scrum Master, and Development Team members. In the following pages a description of the three roles will be provided, while a summary of roles and responsibilities is present for faster consultation in Table 1.4. The Product Owner is responsible for the product, she has to determine the features that need to be implemented and communicate them to the team, but also to maintain them as the project goes on. This does not mean that the Product Owner has to define individual tasks, she is responsible only for the product-level units of functionality. She can also change these properties, but not while a sprint is in progress. The team, instead, will then fulfill them by breaking them down in simpler tasks. In this sense, the Product Owner has to deal with the project at the start of every sprint, selecting user stories and explaining them, and at the end of each iteration, evaluating its results (Meyer, 2014). More generally, she has to facilitate decisions about that product, and has the final say over these decisions. The main benefit of having a Product Owner in Scrum is to separate the job of defining project objectives from the day-to-day management of the project, that is assigned to the Scrum Master (Meyer, 2014). The Scrum Master, in fact, has to monitor the team, ensuring that Scrum is understood and correctly applied by its members; she can also have an intermediary position between the Product Owner and the rest of the team, to ease communication. Originally, this role was created to substitute a multitude of overlapping roles, such as coaches, mentors, gurus, and method enforcers (Meyer, 2014). The Scrum Master should, as a coach, advise and not prescribe. She should be involved in the project directly and do some of the real work together with the developers, not simply act as a manager. Additionally, the Scrum Master should remove impediments identified by team members in daily meetings, both technical or organizational, that prevent the team from operating at full productivity. The Scrum Master is also responsible for protecting the team

from distractions and undue interference from the rest of the organization (Meyer, 2014). On the other hand, she does not have the same power of a Product Owner: *a Scrum Master may not be able to say "You're fired", but can say "I've decided we're going to try two-week sprints for the next month"* (Cohn, 2010).

Scrum Roles	Responsibilities
Product Owner	• Clearly express product backlog items • Order product backlog items to best achieve goals and missions • Ensure the value of the work the development team performs • Ensure that the product backlog is visible, transparent, and clear to all • Ensure the development team understands items in the product backlog
Scrum Master	• Clearly communicate vision, goals, and product backlog items • Teach participants to create clear and concise product backlog items • Facilitate Scrum events as requested or needed • Coach in self-organization and cross-functionality • Remove impediments to the Development Team's progress • Plan Scrum implementations within the organization • Help employees and stakeholders understand and enact Scrum
Development Team	• Self-organize - turning product backlog into product increments • Cross-functional collaboration • Share accountability in the Development Team as a whole • Avoid sub-teams dedicated to particular domains

Table 1.4: Scrum roles and responsibilities (Sommer et al., 2015)

Not only the people, but also the structural elements of the software development project are defined by the Scrum approach. In particular, it is worth mentioning the following:

- **Product Backlog**: the list of features that need to be implemented, as defined by the Product Owner, following customer requirements in order of priority. Also defined as the collection of user stories, meaning the set of artifacts that agile practices use to define customer needs.

- **Sprints**: short iterations, lasting from one to four weeks (or six, according to some authors), that the project will undergo, in each Sprint the team will add new features. Although Scrum embraces change, during a given sprint there is no room for changes in requirements, functionality can only be added in the sprint planning phase. Once the sprint has actually started, no one is permitted to add anything until the end of the sprint, managers included (Meyer, 2014).

- **Prototype**: a partial marketable product, achieved at the end of each sprint, it is then improved in the following ones changing requirements or adapting

to customer's needs, updating the product backlog;

- **Sprint Backlog**: list of tasks addressing a subset of the requirements that the development team must perform during a given sprint, it can also contain information about previous sprints, to provide project knowledge to all the team members. As the Product Backlog contains the user stories, the Sprint Backlog includes the smaller and more specific tasks related with every user story.

- **Daily Scrum**: a daily meeting in which the team members discuss the project progress. As Meyer points out, most agile methods advocate frequent face-to-face contact, but Scrum puts extra effort in this sense, demanding to organize these meetings every morning. Their duration is usually set around fifteen minutes, during which the team members answer three questions: "What did I do in the previous working day?", "What do I plan to do today?", and "What impediments am I facing?". Anything else must occur outside of the meeting (Meyer, 2014). In this way, a plan for the following twenty-four hours is created. Each sprint is then developed looking at the sprint backlog and monitored on a burn-down chart (Cooper, 2016).

- **Review** of the current Sprint. It usually starts with an assessment of the work done during the day, performed by the development team and presented to the Product Owner and outside stakeholders. At the end, new Sprints can be created, reorganized, optimized, and updated. The review meeting should only focus on results, not processes.

Different versions of Scrum also subdivide the sprints into a set of different subsequent phases, such as *Pre-Sprint planning, Sprint, Post-sprint meeting* (Cohen et al., 2004). This way of organizing the sprint may be useful but can cause ambiguity in people used to the previous one. The Post Sprint meeting, as an example, has the same function of the so-called Review. Similarly, the Product Backlog is also called "Release Backlog" by some practitioners. Thus, when applying Scrum into a project, it is worth taking the time to get used to the shared nomenclature inside a team. The same holds for any other popular method, that could have undergone similar modifications during the years.

Scrum also requires a set of physical facilities, such as a dedicated project room where the team members meet during the project development process. These

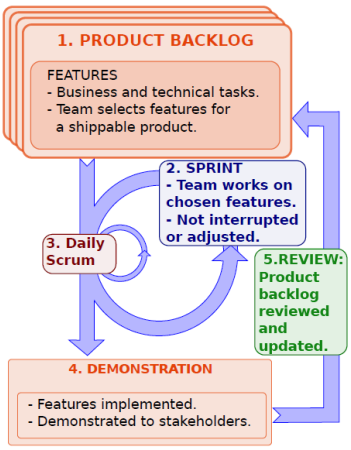

Figure 1.2: The Scrum method's (Schwaber, 2007)

rooms must be equipped with at least one large white board called the scrum board (or scrumboard). This tool is used for visually displaying the sprint process, meaning that all the elements of the sprint backlog must be listed here, sorted by priority. In addition, one could also report here a burndown chart, and the product backlog. All these elements together provide the team with a quick overview of the sprint status (Mabrouk et al., 2018; Mulder et al., 2014). The burndown chart, in particular, is a record of a project's velocity and depicts how fast the project "burns" the items in its task list. As shown in Figure 1.3, the velocity is the number of tasks discharged and the green line represents the constant-velocity line, also called the ideal burndown. If the chart (the dark blue line) is below the ideal plot, the project is progressing faster than expected. If it is above the line, it is progressing slowly than expected.

The above-mentioned roles and steps that Scrum users adopt daily were created for software development projects only. However, they include generic project management tools that could be re-applied in any industry and project, as discussed later on.

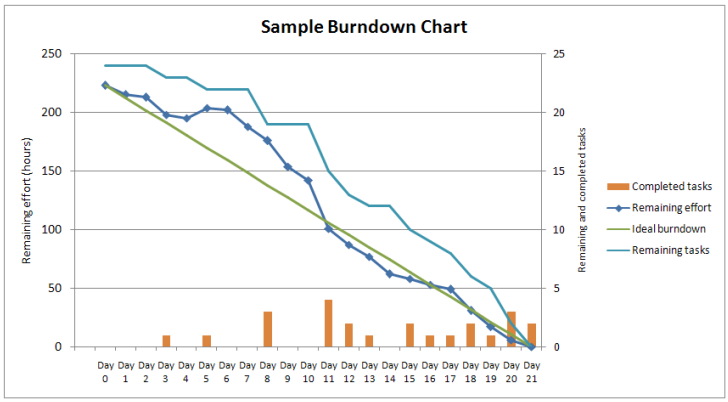

Figure 1.3: Burndown chart example from wikipedia.org

Figure 1.4: Scrumboard example from Meyer (2014)

1.3.2 Extreme Programming (XP)

Extreme Programming, often also named eXtreme Programming and usually abbreviated in XP, was presented in 1998 by Beck and Jeffries that, as partially mentioned before, re-discovered the principles at the basis of Lean Manufacturing and tried to apply them in the software industry. In particular, they were hired by Chrysler to unify three existing payroll systems, an apparently simple project that, however, was declared a failure before they arrived and was creating several issues to the company. Thanks to their approach, the task was quickly and successfully completed. In doing so, XP was in a certain way invented and applied for the first time. XP then gained popularity thanks to Beck's *Extreme Programming*

Explained: Embrace Change (Beck, 2000) in year 2000.

Compared to Scrum, XP is less structured and consists of a list of principles and best practices based on four main values: *communication, simplicity, feedback, and courage*, and *quality work* (Highsmith & Cockburn, 2001). In addition to the original XP's authors contributions, the following list includes extra comments from Cohen et al. (2004), Mulder et al. (2014), and Beck (2000):

1. **The planning game**: before each cycle, every task and feature is analyzed, estimated and organized in priority. Customers, managers, and developers meet to flesh out, estimate, and prioritize requirements for the next release. The requirements are called *"user stories"* understandable by all parties.

2. **Small releases**: at each release, make the minimum useful changes, adding a minimal amount of features. In this way, an initial version of the system is put into production after the first few iterations. After that, new working versions are put into production every few days or weeks, depending on the project.

3. **Metaphor**: to model the system, it is suggested to use metaphors explaining the roles and relations inside the team, both for the business people and for the technical people.

4. **Simple design**: developers are urged to keep design as simple as possible.A simple design should pass all the tests, have no duplicate logic, state its intention to the programmers and have the fewest possible classes and methods.

5. **Testing**: all codes should be unit tested to make sure they work properly. Developers should write acceptance tests for their code before they write the code itself. Additionally, customers write functional tests for each iteration and at the end of each iteration, all tests should run.

6. **Refractoring**: reorganizing the code/design after its implementation, to see if it can be done in a simpler way. In other words, the design should be evolved while developers are still working, in order to keep it as simple as possible.

7. **Pair programming**: two people work on the code simultaneously: one implements the code and the other one thinks strategically about what is the best way to implement the features. They should use the same machine.

8. **Collective ownership**: changing partners and roles often, the whole team understands different parts of the system, feeling more confident in changing and improving them, instead of knowing only a limited portion of it. In this way the code is *owned* by all developers.

9. **Continuous integration**: each piece of code is individually tested and then added to the complete system. New code integrations are performed as often as possible and all functional tests must still pass after the integration.

10. **40-hour work week**: people working at the project should not work more than that, overtime should be followed by free time to recover and the project requirements should be selected accordingly.

11. **On-site customer**: the customer should always be involved in the design process. He works with the development team to answer questions, perform acceptance tests, and ensure that development is progressing as expected.

12. **Coding standards**: standards should be used, to allow everyone to adapt and further develop the code. Other sources also insert.

13. **Open workspace**: developers work in a common workspace, with individual workstations around the periphery of the workspace and common development machines in the center.

Thanks to these practices, XP immediately became very popular, and was even considered to be better than Scrum, at least until the first half of the 2000s. Generally, practitioners of that time agreed in saying that XP's strength came from the simultaneous application of all the principles described before, meaning that each of them alone was not giving XP any extra usefulness. Developers had to commit and implement as many of them as possible in order to get some competitive advantage out of this method.

As any other method, also XP has some drawbacks and critics. The teams' size, as an example, can not exceed the number of people that can be fit into a room, so usually has to be lower than 10. However, thinking about this constraint in modern terms, it could also bring to the conclusion that XP can not be applied in case of remote working. Of course this is not the case, but it represents a very simple example of how, even within the software development boundaries – for

which these methods have been created – all of them have evolved throughout the past two decades. For this reason, and for the sake of brevity, the descriptions present in this section tend to focus more on their original form, giving only brief hints of how they evolved and transformed into different variants of the original methods.

Finally, comparing XP to the other agile methods presented here, it can be noted how it has the shortest recommended iteration length and, as many others, it was not created to be applicable to a single use case; according to Cohen et al. (2004), there is nothing in XP itself that should limit its applicability.

1.3.3 The Crystal Methods

The Crystal Methods were developed by Alistair Cockburn in the early 1990s. Highsmith worked in a close relation with Cockburn and described Crystal as a set of methods that "focuses on people, interaction, community, skills, talents, and communication as first order effects on performance. Process remains important, but secondary" (Highsmith & Cockburn, 2001). Cockburn himself, instead, gave a more drastic definition:

> To the extent that you can replace written documentation with face-to-face interactions, you can reduce the reliance on written work products and improve the likelihood of delivering the system. The more frequently you can deliver running, tested slices of the system, the more you can reduce the reliance on written 'promissory' notes and improve the likelihood of delivering the system" (Highsmith et al., 2000).

His goal was to remove what he identified as the main obstacle to product development: poor communication. As Meyer (2014) points out, however, Crystal does not enforce a "communicate-at-all-costs policy", on the contrary it forcefully imposes to respect the programmers, to accept that people are different, and to accommodate them.

Up to this point we did not put much attention of the origin of Scrum and XP's names, for Crystal however it is interesting to know how the name was accurately

chosen to represent a gemstone, where each facet stands for another version of the process, all arranged around an identical core (Highsmith & Cockburn, 2001). Moreover, the metaphor is also used to classify the different crystal methods: the most agile version is Crystal Clear, followed by Crystal Yellow, Crystal Orange, Crystal Red, and so on. The differences among all these variants depend on the number of people involved, as this directly affects the degree of emphasis on communication. As people are added to the project, the method becomes more opaque. This in turn means that the project criticality increases, more constraints are needed, and the method "hardens", becoming less agile (Cohen et al., 2004). Highsmith, however, stresses that even the more opaque ones are still agile, thanks to the common mindset that they share with the more agile ones (Highsmith & Cockburn, 2001).

Differently from the previously mentioned ones, the Crystal Methods do not have a specific team size constraint. Their iterations length is usually set around four months, much longer than Scrum and XP durations. Finally, Crystal supports the use of distributed teams, something that XP does not consider at all.

1.3.4 Feature Driven Development (FDD)

Feature Driven Development was developed in circumstances similar to those of XP. Its creators, Jeff DeLuca and Peter Coad, were hired to save a failing landing system. Their predecessors had created 3500 pages of documentation without succeeding and, more importantly, without implementing a single line of code. The FDD approach was created by joining DeLuca and Coad's previous experiences and was proven to be successful, saving the project from a disastrous ending.

FDD's core values are summarized as it follows by Highsmith and Cockburn (2001):

- A system for building systems is necessary in order to scale to larger projects.
- A simple, well-defined process works best.
- Process steps should be logical and their worth immediately obvious to each team member.
- "Process pride" can keep the real work from happening.

- Good processes move to the background so that the team members can focus on results.
- Short, iterative, feature-driven life cycles are best.

Apart from these general values that practitioners need to apply, the method also provides a series of practices to follow. The first step consists in "developing an overall model", i.e., creating a *walkthrough* version of the system thanks to the collaboration between team members and experts. Secondly, a "feature list" is needed, meaning that the team has to identify and collect a set of features that will represent the system. In this sense, FDD features are somewhat similar to XP's story cards, they need to describe items useful in the eyes of the client, in a language understandable by all parties. Another interesting remark about features is that the creation of such a list should not take more than ten days, otherwise features must be broken down into sub-features. The third step consists then in the prioritization of such features into subsections called "design packages". The design packages are then assigned to a chief programmer, who in turn assigns class ownership and responsibility to the other developers and team members, this is what the authors called "plan by feature". Fourth, the team will undergo the so-called "Design by feature & build by feature" phase, during which the iterative approach, classic of agile methods, is applied. The chief programmer chooses a subset of features that will take 1 to 2 weeks to implement. These features are then planned in more detail, built, tested, and integrated (Cohen et al., 2004).

With FDD, there are no specifications about an ideal team size. A greater focus is, instead, put on "premium people". The authors believe that their presence is of extreme importance inside any team. For what concerns the iterations' length, FDD is usually applied with two-weeks long sprints. Finally, FDD supports the use of distributed teams, similarly to Crystal.

1.3.5 Lean Development (LD)

Lean Development was created by Bob Charette who, as already mentioned, took great inspiration from the Lean Manufacturing methods, applied especially in the automotive industry in the 1980s. This method is quite simple, and it will not

be investigated much further, however it is interesting especially because it provides a very different approach with respect to the other methods already analyzed. Instead of looking at the development process, in fact, LD is a tool to reach what Charette believed to be the key to agile: "you need to change how companies work from the top down" (Cohen et al., 2004).

LD is based on twelve management strategies (Highsmith & Cockburn, 2001):

1. Satisfying the customer is the highest priority.
2. Always provide the best value for the money.
3. Success depends on active customer participation.
4. Every LD project is a team effort.
5. Everything is changeable.
6. Domain, not point, solutions.
7. Complete, do not construct.
8. An 80 percent solution today instead of 100 percent solution tomorrow.
9. Minimalism is essential.
10. Needs determine technology.
11. Product growth is feature growth, not size growth.
12. Never push LD beyond its limits.

1.3.6 Dynamic Systems Development Method (DSSM)

Dynamic Systems Development Method is not really an agile method, but more of a framework. It is made of six stages: Pre-project, Feasibility Study, Business Study, Functional Model Iteration, Design and Build Iteration, Implementation, and Post-project (Cohen et al., 2004).

During the six phases, the project is first analyzed to understand whether the conditions to start working on it are present or not. In fact, the purpose of the first three stages is to assess its feasibility under different perspectives. Funding is considered, but also the very DSDM usage is discussed, to understand if the project could benefit from it. The business study, additionally, makes use of workshops attended by knowledgeable staff who can quickly pool their knowledge and gain

consensus as to the priorities of the development" Cohen et al. (2004). At this point the project should get to the Business Area Definition, which identifies users, markets, and business processes affected by the system.

The subsequent phase is the Functional model iteration, where a set of prototypes are built, in order to satisfy the high-level requirements identified in the business study. These prototypes will then evolve towards the complete system during the Design and Build Iteration phase. Here, in fact, the prototypes are combined and tested, delivering a working system to the users. During the implementation phase, the system is finally transitioned into use. In this phase, it is important to notice whether the system already fulfills every requirement and whether it needs improvements. In the latter case, improvements are implemented.

1.3.7 Agile Modeling

The last method analyzed in this section is Agile Modeling, created by Scott Ambler (Ambler, 2002a) and based on values, principles, and practices that focus on two key aspects of software development: modeling and documentation. Ambler describes AM's goal as it follows (Ambler, 2002b):

1. To define and show how to put into practice a collection of values, principles, and practices that lead to effective and lightweight modeling.
2. To address the issue on how to apply modeling techniques on Agile software development processes.
3. To address how you can apply effective modeling techniques independently of the software process in use.

Agile Modeling, however, is not exactly a software development method. Its nature is, in fact, slightly different from the above-mentioned agile methods, AM is more of a documentation and modeling method, that can be coupled with any other agile technique. As an example, one could develop her project implementing XP, while making use of Agile Modeling to document it. The two methods share a lot of values and principles, and their mutual integration is extremely simple.

AM itself is quite simple and self-explanatory. It may be sufficient to read and

understand its lists of values and principles, in order to understand it and apply it. AM's values can be summarized as follows (Ambler, 2002a):

1. Assume simplicity
2. Content is more important than representation
3. Embrace change
4. Enabling your next effort is your secondary goal
5. Everyone can learn from everyone else
6. Incremental change
7. Know your models
8. Local adaptation
9. Maximize stakeholder investment
10. Model with a purpose
11. Multiple models
12. Open and honest communication
13. Quality work
14. Rapid feedback
15. Software is your primary goal
16. Travel light
17. Work with people's instincts

In the following list, instead, AM's principles can be found (Ambler, 2002a):

1. Active stakeholder participation
2. Apply modeling standards
3. Apply the right artifact(s)
4. Collective ownership
5. Consider testability
6. Create several models in parallel
7. Create simple content
8. Depict models simply
9. Discard temporary models
10. Display models publicly
11. Formalize contract models
12. Iterate to another artifact

13. Model in small increments
14. Model to communicate
15. Model to understand
16. Model with others
17. Prove it with code
18. Reuse existing resources
19. Update only when it hurts
20. Use the simplest tools

Of course, not every item on the two previous lists can be really understood by a simple definition. However, for sake of brevity and because this paper is not focused on agile software development, they appear here in their shortest possible definition. Any further analysis is left to the reader.

Chapter 2

Agile in the manufacturing industries

Up to this point, agile has been solely described as a software development tool. In the last few years, however, also other industries have approached these methods. Agile, in fact, could be applied to a wide range of manufacturing industries, potentially bringing them benefits similar to the ones provided to the software world.

Additionally, apart from the simple intuition that if something works well in a given environment, then it could work just as well in a different sector, there are other solid reasons to believe in the application of agile to the manufacturing area. In the last 30 years, in fact, the world changed: as noted by Cantamessa et al. (2020), digital technology has brought changes and disruption to many industries worldwide, and "digital" corporations have now climbed to the top of the league tables. As a consequence, design and product development changed too, and this affected manufacturing companies as well. The shift towards the so-called *Industry 3.0* and, more recently, *Industry 4.0*, are steps of a deeper and broader transformation, which is still ongoing. This process brought hardware companies to deal with an increasing level of digitalization, larger amount of data and, in general, to a new paradigm.

The aim of this chapter is indeed to describe this new paradigm and, later on, to assess how agile methods could fit in this framework. At the same time, it will be performed an attempt to understand how agile methods are already put in place outside software development, and how they contributed to the creation of hybrid methods and adapted version of agile in projects that involve hardware product development. Finally, some conclusions about the new possibilities for the agile in this sector will be discussed.

2.1 The new manufacturing

As we all know, in the First Industrial Revolution (mid-18th – mid-19th centuries) the main driver for change was the use of steam engines to mechanize manufacturing processes. As a result, factories sprang up, producing goods more quickly and cheaply than could be done by hand. Similarly, in the Second Industrial Revolution (early 20th century) the use of electricity sped up manufacturing even more, aided by the assembly line, pioneered by Henry Ford. This increased productivity and allowed automobiles and other complex items to be mass-produced for wide distribution. The following step occurred with the Third Industrial Revolution (late-20th century) and the invention of the Internet, which enabled goods and services to be produced, marketed, and consumed globally. Finally, the Fourth Industrial Revolution (early 21st century), now underway, connects people and things with digital technologies which provide computers of visual perception, speech recognition, decision making, and language translation abilities. But of course this trend has not yet ended and the next, preannounced, step will be the fifth revolution, in which humans and intelligent machines will work together (J. Reagan & Singh, 2021).

As mentioned, the product development process has undergone – and it is still undergoing – transformation processes that reflect the various paradigm shifts that the whole industry has experienced. In particular, due to the purpose of this paper, the focus will be on the last two revolutions, i.e., the third and fourth, also called Industry 3.0 and Industry 4.0, with an eye on the future.

At the beginning of this journey, the product development process benefitted

from IT and computer technology tools, such as computer-aided design (CAD), Computerized Numerical Control (CNC) machines, robots, Programmable Logic Controllers (PLC). As a result, the product design, traditionally relegated to 'passive' paper-based visualizations, migrated to digital models (Cantamessa et al., 2020). As an example, drawings passed from something that could only be looked at, to a tool that can automatically lead to calculations or simulations, but can also interoperate with another drawing of another part. Digital models now incorporate the structure and the history of the product and can support simulation under multiple perspectives (Cantamessa et al., 2020). With Industry 4.0, then, the digital side took even more space, thanks to sensors, Cloud Computing, Blockchain, Artificial Intelligence, Business process Automation, and many others. For instance, Cantamessa et al. (2020) observe that

> The Internet of Things (IoT) led to continual connection and flow of data between people and objects and between objects and other objects; Augmented and virtual reality allow richer representations of objects and environments, merging the real with the virtual, and vice-versa; powerful and cheap IT equipment enable the virtualization of physical servers, leading to cloud computing and the possibility of storing Big Data; Data mining can exploit the value hidden in massive amounts of heterogeneous data [...]; Machine learning allows the development of new forms of automation and decision-making, based on the re-elaboration and 'digestion' of large amounts of data.

These technological innovations, however, do not only provide positive effects, but also create new challenges to the businesses. They cause social shifts, modify the existing business models, and affect the products and services development. In turn, this has an effect on the consumers expectations, roles, and behavior, but also on the suppliers activities. In the following sections, some of these trends are analyzed with more detail.

2.1.1 The role of data

One of the first implications of some new technologies is surely the increase in the amount of data that any company has to manage and analyze nowadays.

Information come from different domains, such as design, manufacturing and after-sales services, and need to be, somehow, integrated. Not only internally, but even from different companies with different sources of information, and different methods or software programs that are rarely integrated, causing additional confusion. Apart from the sources of data, also the different systems are used and can cause troubles: coherence and de-fragmentation of the data regardless of the source and the format, and more importantly their usage may represent a significant operational problem for designers (Cantamessa et al., 2020). For instance, the presence of a large amount of data, which could be used to support decisions, is often useless for the simple reason that companies may not even know what data are already available in their database and what data they are producing (Altavilla et al., 2017). In other cases, designers do not know how to make the best use of these data, or tend to use modern technologies such as AI algorithms without having a clear overview on their potentialities. In this regard, product development could benefit from such algorithms thanks to design optimization and prototyping, but the uncertainties around them can rise additional questions and doubts about the automation of design: will they help the designers or substitute them? Agile could provide a solution to such issues, given its tendency to limit documentation to the bare minimum and to properly manage it in a useful and efficient way. Agile methods could also provide a standard that would enable different entities in the product value chain to cooperate consistently. Additionally, approaching such an amount of data in a more "agile" way could help in dealing with the continuous bundling and unbundling of data, required at each product lifecycle stage.

A slightly different organizational issue, instead, is represented by the trend of design information and knowledge management shifting from people to capital. Nowadays, in fact, CAD, simulations, modelling, and in general the digital tools employed allow designers to do what was previously done by intuition and experience. The transition towards design automation is reporting tacitly the knowledge that a designer can afford the luxury of not having anymore, thus attributing to systems an active role in design processes (Cantamessa et al., 2020). As a consequence, individuals progressively yield their knowledge to the company, to the capital: as the software learns from the users, the designer's know-how is transferred to the company, changing the organization equilibria (Cantamessa et al., 2020).

2.1.2 Cyber-physical systems

One of the results of the application of such new technologies is that, instead of simply automating tasks, companies nowadays are able to use Cyber Physical Systems (CPS) such as smart machines, storage systems, and production facilities that autonomously exchange information, triggering actions and controlling each other. This trend caused (again) an inevitable increase in the amount of data that any company needs to process, and, as a consequence, to the possibility that the agile methods, originally developed to deal with digital products, could now be adapted to fulfill the needs of the modern hardware development companies.

What is interesting to see, however, is that CPS can be exploited from companies as a massive source of data supporting their operations, but they also need to be developed and produced. And, due to their nature, agile could be applied starting from such hybrid objects, with a lot of software content. CPS, in fact, can collect information in the physical world through sensors, elaborate it in the cyber world, and finally change the physical world through actuators (Mulder et al., 2014). For this reason, hardware and software development are equally important in such complex systems, and agile methodologies could help in that, if properly applied. CPS development, in fact, does not work well if the *classical* subdivision between hardware and software development is applied. There is need of simultaneous actions. To go more into details, according to Horvath and Gerritsen (2012), in every CPS development process there should be five different *platforms* developed simultaneously: netware, hardware, software, firmware, and knowledgeware, not only HW and SW.

On this regard, Mulder et al. (2014) focused on adapting Scrum methods to the development of CPSs. They compared it to the four main methods usually employed in the CPS design: V-model, Model-based development, Component-based design, and Platform Based design. The idea was to exploit the selforganizing, multidisciplinary and non-hierarchical structured aspects of Scrum teams, in order to foster collaboration between different domains, essential for CPS's design and development. To do so, the team set up an experiment, trying to develop a sail simulator system, for competitive sailing. The CPS characteristics in such a product are found in its interactions with the user: the physical coordination of balance,

spatial awareness and haptic forces that make it useful for training purposes. It also includes a mechatronic system, a three-dimensional display, computational simulation, and game mechanics to measure performance.

Due to its complexity, initial knowledge was gathered in an agile way, using a number of prototypes, that in turn were possible only thanks to the short iterations that Scrum enforces. During the development phases, instead, all the Scrum elements were adopted by the team (one-week-long sprints, pre-sprint meetings, usage of a scrumboard and a burndown chart, prototypes evaluation at the end of each sprint, product backlog creation) and Mr. Mulder acted as the Scrum Master himself. The rest of the team was made by students with no Scrum experience, but they clearly indicated that the Scrum development process made the project easier. In particular, those who had previous experience with the waterfall method appreciated the fact that Scrum is a less documentation-heavy process and that the agile approach made predicting future problems and requirements easier. They also stated that it helped to get human-centered insights. The study suggests that a key point in applying Scrum to CPS projects is to carefully choose an appropriate team leader, a process defined "more important than expected: a good project leader should keep track of the progress made and take responsibility when changes in the planning had to be made". (Mulder et al., 2014)

Regarding the specific process adopted, Rapid Prototyping was a key enabler for the hardware components, that would have been impossible to build in a week-long sprint with other technologies. From an organizational point of view, instead, Scrum meetings were not found to be useful enough, mainly due to the poor ethics of some team members that did not show on time or did not share enough information. On the other hand, the review sessions were found extremely useful and also included athletes (in the role of customers) that helped the team better understand their requirements. The only tools employed were scrumboards, that were found not practical enough, and the team suggested switching to an online alternative.

Concluding, the authors listed the following recommendations and observations:

- Scrum philosophy and procedures have to be fully embraced by the team members, as the team cohesion is only achieved when all members follow

them.

- The lack of documentation can cause knowledge loss, thus the suggestion of using a more detailed scrumboard and implementing a checklist to better monitor which features are ready and which are not.
- After each review session a document which summarizes current and updated product requirements should be used, sharing the requirements with the reviewer, the product owner and the Scrum team.
- The length of the sprint should be selected based on the experience, available knowledge and motivation of the Scrum team.

2.1.3 Embedded systems

Similarly, embedded systems are another field of application for agile methods outside pure software development. They are defined as specialized computer systems designed for specific tasks, that typically consist of software and hardware. According to Könnölä et al. (2016), in developing embedded systems, usually only the software part makes use of agile development methods, while the hardware development is still exploiting traditional methods. The problem could lie in the absence of guidelines on how to organize development work on a weekly basis for such systems. The authors highlight how standards are present for hardware development, software development (as discussed in Sections 1.3.1 and 1.3.2 analyzing Scrum and XP), but no standard is set for embedded systems. Some principles could be *borrowed* from the SW methods, but most of them should be reinterpreted in order to be usefully applied. As an example, the Agile Manifesto states that working software should be the primary measure of progress; in embedded systems this parameter must be substituted with the whole system development and demonstrations of its working principle should become the measure of progress (Kaisti et al., 2014).

According to literature (Könnölä et al., 2016), the main opportunities for agile methods in embedded systems' development are :

- **System-wide understanding**: better communication would foster easier agreements on the interdependencies between different sections of the system and, as a consequence, better alignment during the development phases.

- **Acceptance of change**: when new requirements are needed the teams would be prepared for them and would have the necessary tools to face them effectively.
- **Fluent management of the interdependencies**: more transparency would reduce incomprehension in case of interdependencies, thus also the presence of critical paths between different sections.

However, when trying to implement agile in embedded systems' development, one would also face some challenges (Könnölä et al., 2016):

- **Changes may affect the whole system**: while agile welcomes change and tries to postpone decisions as much as possible, in embedded systems any change in the software could cause changes in the hardware specifications, and vice versa. This kind of constraints should then be addressed early enough, giving agile some space within these boundaries.
- **Difficulty in delivering new versions quickly**: unlike software development, hardware products can not be updated too frequently, due to the presence of systematic tasks that need to follow specific cycles (e.g., design, prototype manufacturing, testing, and verification). This aspect has to be taken into account when implementing agile.
- **Team members are highly specialized**: in agile development, every team member is welcomed to learn and know about every aspect of the code, instead of limiting himself on specific tasks. In embedded systems, however, usually every team member is highly specialized in one single discipline, due to their complexity. It is very unlikely to have people working both at the hardware components and at the software of the system. Nevertheless, agile could still be applied to foster transparency and help everyone understand what the others are dealing with, even if it will be impossible to reach complete cross-functionality inside the team.

Starting from these premises, Könnölä et al. (2016) performed three different case studies, applying agile methods to embedded systems design, getting feedback from the teams. The companies considered were Ericsson, a multinational company, leader in the field of communications technology; Nordic ID, a SME developing and manufacturing RFID and barcode readers; and Nextfour, another SME which develops embedded systems for medical, industrial, and safety-critical markets. As a result of the case studies, various aspects emerged. From the first two

companies, it was noted how the need for internal documentation diminished, while the amount of teamwork, visibility, and understanding about the work of other team members improved. On the negative side, team members faced some challenges in changing the process during its development phases, in particular maintenance tasks disturbed these changes. Also, the teams did not feel to be productive as before. Both companies, however, decided to continue the utilization of the new working methods, as the positive aspects overwhelmed the negative ones. The third case, instead, was considered separately, due to the different nature of the project. Here, practitioners found out that the team perceived the workload to be easier than before, thanks to the absence of circulating tasks. However, the schedules and deadlines were described as something that made the organization of work more difficult and less clear than before. Similarly, they did not experience a positive change in efficiency and productivity, but this might be due to the short length of the project. Viability, instead, improved as in the previous cases.

In general, the teams perceived the new practices useful, and their usage was considered potentially beneficial for future projects. They all appreciated the improved communication inside the teams. The use of backlogs helped to increase the visibility, but an effort had to be done to avoid that people focused too much on the individual tasks, forgetting about the big picture. The authors also point out how embedded systems projects have special characteristics, which need to be taken into account when applying agile methods. They also list some suggestions for this sort of "special tailoring":

- Take into account the different cycle lengths for developing hardware and software.
- Create team-driven agile practices inside the iterations.
- Accept the different knowledge between developers and build on it.
- Define the progress based on work, not schedules or documentation.
- Clarify the reasoning behind the utilized practices together in the team.
- Try more advanced agile techniques.
- Involve the whole organization.

2.2 Agile applications in manufacturing

As discussed for Scrum in Section 1.3.1, agile methods have their roots in manufacturing industries: some of them even originated from the observation of lean manufacturing – a set of techniques that Japanese companies (Toyota in particular) developed to improve the car industry efficiency in the 1950s – and its direct application to software development (Cooke et al., 2012). So, it was probably just a matter of time before agile was re-considered in the manufacturing world. On the other hand, as said in the previous Section 2.1, the world changed and the manufacturing changed accordingly. Becoming more and more digital and connected, it was necessary to find new management solutions, and agile was one of the available options. The fact that agile has a role in our society also outside the IT area was even confirmed by three of the seventeen manifesto's authors in an interview dated 2012 with Bowles Jackson (Jackson & Institute, 2012). In particular, Hunt stated that "agile has little to do with software, since it is all about recognizing and applying feedback". Van Bennekum added that agile is "applicable everywhere in business or life", while Highsmith underlined how it should be "used everywhere we have uncertainty". For these and other reasons, agile entered the manufacturing world more and more frequently, giving birth to the various trends that are analyzed and discussed in the following subsections. Before that, in Table 2.1 a brief comparison of Stage-Gate traditional method and agile is provided. Stage-Gate represents one of the most used methods in traditional industries, such as the manufacturing one, and this simple comparison already shows potential benefits that those industries could get by adapting agile into their working routines.

2.2.1 Agile Manufacturing

One of the first applications of agile in the manufacturing world was, indeed, the so-called *Agile Manufacturing* (AM). Its birth is traced in the 1990s, and it is conventionally set with the constitution of the Agility Forum at the Iacocca Institute at Lehigh University, USA (Vázquez-Bustelo & Avella, 2006). According to Vinodh et al. (2010), AM enables the modern organizations to quickly react

	Stage-Gate	**Agile**
Type	Macroplanning	Microplanning
Scope	Idea to launch	Development and testing, can be expanded to pre-development
Organization	Cross-functional team (R&D, marketing, sales, operations)	Technical team (software developers, engineers)
Decision model	Investment model-go/kill decisions involve a senior governance group	Tactical model—decisions about actions for next sprint made largely by self-managed team

Table 2.1: Characteristics of Stage-Gate vs. Agile (Cooper, 2016)

to the new dynamic demand of modern customers, without compromising quality, productivity, cost, and time. In a more general way, AM is defined as the capability to produce a variety of products within a short period of time also in a cost-effective manner. As AM was being developed, further applications have been studied, especially in the managerial and technological environments. In the managerial field, this resulted in the rise of several methods, such as Total Quality Management (TQM), Total Productive Maintenance (TPM), 5S, Kaizen, and so on. On the technological side, instead, very important tools that can support the agile philosophy are: Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Computer Integrated Manufacturing (CIM), Rapid Prototyping (RP), Rapid Tooling (RT), Reverse Engineering (RE), Computer Aided Process Planning (CAPP), Concurrent Engineering (CE), and Virtual Enterprise (VE) (Vinodh et al., 2010).

2.2.2 Multidisciplinary teams and multidisciplinary projects

The previously mentioned technologies not only influenced the design and manufacturing procedures, like agile manufacturing. They also had consequences on the interactions that design teams have with the other functions and departments of the firm (Cantamessa et al., 2020). As an example, a frequent trend is the integration of IT, marketing and product development functions, due to the evidence that marketers and designers need to cooperate closely, and that information technology

should be integrated at various levels. The creation of multidisciplinary teams allows them to better process the larger amount of data continuously gathered, and creates the possibilities for new professional roles as the one of data analysts. Enlarging the view to the whole company, functional integration among other operational departments, such as production, maintenance, logistics, etc. would allow companies to exploit equipped plants and operative processes with sensors and devices that, apart from increasing efficiency, could also help design processes. This new "setup", however, needs full functional integration, cross-domain knowledge and new competencies and skills (Cantamessa et al., 2020); it needs to be formalized and defined in order to work properly and, once again, the agile philosophy could play an important part in this. In absence of such a formalization, simple behavioral issues (e.g., the employees not sharing enough information) could cause useless money and time consumption to the process. Other more serious aspects should than be considered, such as the need for non-technical specialists to acquire industrial competencies and to develop digital skills, for industrial experts to learn how to deal with data analysts, and for IT specialists to learn engineering bases. This multitude of "hybrid" experts, in between digital and technical competencies, would also need to understand who is the person in charge, and agile's roles could come into play, defining responsibilities and ownership.

Agile could also be applied to the so-called multidisciplinary projects, i.e., projects in which hardware and software are both present and need to be developed simultaneously, involving several experts. These projects face a series of challenges due to the very different nature of the two sides. Firstly, in software development every modification to the product (the code) is almost free, in terms of material cost; while for physical products every new version may have very high costs. In addition, a software prototype (e.g., a beta version) can easily transform into the final product, while a physical prototype can not do that. Similar reasoning can then be followed for raw materials, components, logistic expenses and so on. Continuing, one could see many other obstacles to the application of agile methods in multidisciplinary projects. As an example, the agile approach aims at satisfying the customers' needs through iterations, starting from a very blurred initial idea. On the contrary, in many multidisciplinary projects the initial requirements are already very clear, sometimes also defined by regulations or by physical limits. The

same holds for deadlines and schedules. Instead of embracing change as a positive thing, in this case every discrepancy with the original project should be predicted and properly addressed (Cooke et al., 2012). These few lines already show how the transition of such methods from one industry to another is not easy and represents an interesting challenge. To properly implement it, agile must be adequately adapted to the new system and every potential issue has to be considered. In particular, for multidisciplinary projects, some new technologies can play a very important role: computer aided design (CAD) and rapid-prototyping techniques (RP) such as 3D printing made it possible to test physical products much easily. It is also easier to apply changes on such prototypes, and thus iterations can be exploited. This allows agile methods to be applied to a broader set of development areas.

2.2.3 Hybrid agile methods

In this context, agile can take many different forms and can be adapted to the hardware world in several ways. One of the most frequently seen is a hybrid version of agile methods (especially Scrum), that are coupled with other existing methods, traditionally belonging to the manufacturing area. The idea originated in the multidisciplinary projects described above, since the R&D departments of some manufacturing firms noticed that their colleagues from the IT section were using agile methods, and thought that they could be adapted to their purposes as well. In particular, the sprint approach has been enabled by the fact that in some fields (such as electronics and electromechanical systems), hardware development was becoming more like software development, with shorter, faster iterations in the development stage, thanks to the new technologies mentioned in the previous sections (Cooper, 2016). In other words, hardware development can look more like software development, with quick iterations and quick prototypes. Directly implementing agile practices, however, would lead to some challenges and management resistance. For instance, the lack of scalability and the proliferation of meetings, and in general the difference with respect to the traditional gating systems. For this reason, it is much simpler, at least in the first applications, to blend agile and Stage-Gate into a hybrid method, incorporating the positive features of both

(see Table 2.1). In particular, Scrum – Stage-Gate hybrids appeared as the best solution, since Scrum was the most popular Agile variant among the handful of firms employing Agile for physical product development (Sommer et al., 2015).

With Scrum – Stage-Gate hybrids, Scrum is usually employed in the development and testing phases, when the project has already been approved by a Stage-Gate process, but the development is not yet planned in advance, in opposition to what traditional methods suggest. The development is thus broken into small increments/iterations (similar to sprints), that are then time-boxed into short timeframes, spanning from one to four weeks. As in software development, each iteration is preceded by a meeting in which the following three questions are asked (Cooper, 2016):

- What does the customer value most (based on feedback from customers in the previous sprint)?
- What can be delivered in the upcoming sprint?
- What work is needed to achieve this deliverable?

Other similarities are found in the fact that the work plan for each sprint is controlled almost only by the team, with self-management, and in the post-sprint retrospective meetings focused on the review of the previous iteration, that often include feedback from the customer.

Among the differences, instead, the main one is the definition of a *done* sprint. In hardware development, in fact, it is almost impossible to get a working product every two to four weeks, as it happens in software development, thus the outcome of each iteration must be different. Other differences occur in the management of such projects. With hybrid agile approaches, in fact, the project team must be dedicated, i.e., its members should only focus on one project at a time, while in Stage-Gate this does not happen. This also implies that they should be placed in the same room although, as already mentioned for software development, this strict rule might vary with the advent of modern trends like smart working and virtual meetings. Similarly, they should be provided with a scrum board, a burndown chart with which measuring their progress, and more importantly a Scrum Master should be appointed.

Sommer et al. (2015) and Cooper (2016) also studied the implementation of Agile – Stage-Gate hybrid models in seven Danish, Swedish, and US companies, in a range of industries from consumer products to B2B heavy equipment, reporting the following results:

- Design flexibility (a faster response to change),
- Improved productivity, communication, and coordination among project team members,
- Improved focus on the project leading to better prioritization,
- Higher morale among team members,
- Discipline, provided by the staged structure, and the go/kill decision-points that cull out bad projects,
- Clear expectations (in the form of defined deliverables) for project teams,
- Built-in best practices.

Cooper (2016) goes further in describing five main advantages of these methods. They help to *get the product right*, by requiring the teams to develop something physical or visual, early and cheaply, and quickly get it in front of customers for feedback. This allows to get the customers comments even before the development stages begin, and is so adaptive that it permits modifications in case of changes in the customer requirements. Additionally, building something physical early and often means that solutions to technical issues can be worked through as early as concept prototypes emerge. Hybrids methods also *accommodate uncertainty*, since they do not require a heavy "homework phase" in which the team has to perform market, technical, and business assessments. Instead, they allow working on highly innovative projects, for which the voice-of-customer work or the technical assessments can not get all the answers. The problems are then dealt with only once they arise, by trial and error. Continuing, agile-traditional hybrids *accelerate development* with time-boxed sprints that bring a sense of urgency to the development project. Project teams commit to certain deliverables at the beginning of each sprint, and this forces them to focus on the essentials and deliver results. For similar reasons, hybrid methods help *to focus teams*, which means that project teams are dedicated, thus they need adequate resources to get the work done on the desired timeline. compressed sprint timeline. This makes them much

more productive, and their use is essential for Scrum and agile success. Finally, applying such methods *improve within-team communication* by means of – again – dedicated teams and daily face-to-face discussions. This leads to more effective, cross-functional teams with good internal cooperation and communication

Among the difficulties, instead, the following ones are listed by Cooper (2016), Sommer et al. (2015), and Stelzmann (2011):

- Difficulty in finding dedicated team members,
- Difficulties in linking project teams to the rest of the organization,
- Mismatches between the requirements of Scrum and the company's reward system,
- Need of a redefinition of sprints, to include something physical like a *protocept* (prototype/concept),
- A too bureaucratic system,
- Absence of a clear description on how to implement agile development that ensures coping with the industry regulations.

Stelzmann (2011) adds few contributions in this sense, explaining how hardware systems that have to be produced physically often are difficult to be developed in small cyclic steps. In addition, he states that "only if prototyping, testing, and implementing changes can be done quickly and cheaply, this principle is feasible" and that "agile development methods need to be adapted for the type of product or system which needs to be developed and cannot be implemented one to one". Additionally, he believes that companies should focus on fostering the usability and user satisfaction both in physical and cognitive aspects; focus on maintenance for lead time and sustainability aspects; make extensive use of prototypes and early versions; keep an eye on aspects such as crowdfunding, codevelopment, and social media to survive competition.

Cooper also points out how other companies are trying to apply hybrid agile systems to more than two steps, broadening its application to predevelopment stages, such as the concept identification and the feasibility assessment. He points out how "in these early phases, open knowledge gaps become analogous to desired software features on the burndown chart, and Scrum then works in the normal way, with each sprint aimed at resolving a particular gap or set of gaps" (Cooper, 2016).

2.2.4 Agile Project Management

In parallel with the agile implementation in the product development, other practitioners also focused on the exploitation of agile principles in the management field. They gave birth to Agile Project Management (APM), an approach that can be used irrespectively on the product or service that needs to be delivered and that, for this reason, is also part of the agile manufacturing world.

Applying APM, the project objectives are defined in less detail – at the beginning of the project – together with a rough project execution schedule. The project is then divided into small iterations. Every team focuses on the most important functions to be delivered in each iteration, leaving the least important ones at the end. In this way, the least important parts of the projects can be easily removed, depending on the customer requests, that can vary during the project life. Other aspects that can influence this decision are the conditions of the environment surrounding the project, and the direct proposal of the team members. Going on with this approach, every iteration can then be reviewed and scheduled with more detail. In this phase it is necessary to decide how the desired results will be achieved, considering every task, the hours of work needed, the personnel, and every other technical aspect. Again, every decision needs to take into account the customers' needs and changes, the team ideas, and the results obtained in the previous steps. Contrary to what happens in Scrum, the project team itself will be responsible for the creation of this kind of execution plan in every iteration. This function is not covered by a project manager. Empirical studies show that iterations usually last from one to four weeks. This short duration allows intermediate results' testing, that allows to detect issues earlier, as well as customers feedbacks (Stare, 2014).

A variation of APM is Extreme Project Management (EPM). The differences between these two are in the level of familiarity with the solution at the beginning of the project, the detail of planning, the role of risk management, and the collaboration with the client (Wysocki, 2011). With EPM, in fact, the approach is even more distant from the traditional one, with respect to APM. This means that not only changes are awaited, but even the project objectives are extremely unclear at the beginning. Everything is decided during the project, and for this

reason, EPM is suitable almost only for R&D projects.

In his work, Stare (2014) observed the application of the APM method in 21 product development projects in five Slovenian enterprises. In doing so, he was able to classify the main differences between this and the traditional approach in four groups: requirements & specifications (the level of detail at the beginning of the project), project scheduling (iterations and a rough schedule at the planning phase), team work (self-organized teams, daily meetings), and the client collaboration (the representative of the client is a regular team member). In general, his research highlighted how certain practices were already in place in the observed companies, and how the application of APM seems to have some potential in future applications. In Figure 2.1, a summary of the projects' success is provided, based on the team members' answers to Stare's questionnaire.

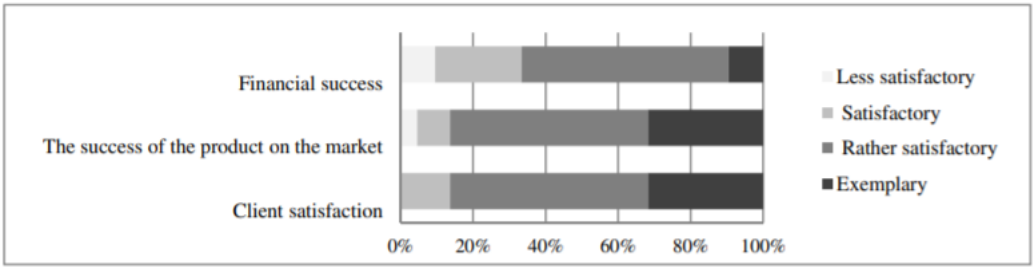

Figure 2.1: The projects' success indicators (Stare, 2014)

One last interesting thing highlighted by Stare, is that in the above-mentioned companies, client collaboration showed the lowest level of agile approach in projects. Only 5% of the projects had customers participate on a daily bases, on 50% of them the customer participated weekly, on 15% monthly, and on 30% even more rarely. Thus, customer participation could be one of the key agile aspects on which companies should put more effort, in order to implement it.

2.3 Agile trends and future applications

In the previous section, the description of the main applications of agile to the manufacturing sector has been provided; the aim of this section, instead, is to analyze the evolutions of such methods. In the following pages, the main papers

analyzed for this work are summarized, in order to describe as much as possible the future applications of agile in the manufacturing, as well as their current experimental uses and real life observations.

Some of the papers will follow a descriptive approach, observing the situation that surrounds them; others provide suggestions and best practices example, with a normative approach. The papers considered also represent a wide set of industrial applications. In fact, some of them report real life case studies from very different industries: pump manufacturing, consumer electronic, biomedical, kitchen tools, automotive, bathroom appliances, barcode readers, windows, power-lines, plastic toys, and so on. Other authors, instead, preferred to use in-house simulations, students' group works, and fictional teams competitions. Some of them then focus on SMEs, while others reported observations of bigger companies. Finally, most of the papers come from European authors, but also American and Asian ones are present, each describing (also) his/her local market situation.

2.3.1 Descriptive approach

The first descriptive contribution considered comes from Böhmer et al. (2017), who centered their work on the prototyping role in agile product development environments. In their work, a detailed literature review on this topic was performed, highlighting how prototyping can be seen as an insurance to minimize the risk towards the end of the development project; and a tool for the discovery, evaluation and development of new product ideas. It was also stated that prototypes have a different role in agile and in traditional approaches, since with the former methods their goal is to reduce uncertainty and produce knowledge iteratively, while in the latter case they are simply used as initial versions of the final product. Finally, an important distinction between horizontal and vertical prototyping was made: "horizontal prototypes represent a specific feature of the system, e.g., the human-computer interface, without fully implementing them. Vertical prototypes focus on implementing a small set of features in a nearly-complete fashion" (Böhmer et al., 2017).

With these premises, they tried to understand how prototypes could impact

the development of mechatronics projects adopting agile strategies. The research observed forty groups during a practical course at TUM (Technische Universität München), and each group was free to choose the preferred approach: most teams applied agile principles, but not all of them.

Part of the ones that chose the agile approach struggled in capturing the big picture while not getting lost within testing and incremental development, or simply applied agile principles wrongly. The absence of documentation provoked serious challenges for them. Thus, the authors suggest that a structured daily documentation is crucial for agile, as it eases to focus on the most important aspects and facilitates reviews. This does not mean that agile should make extensive use of documentation as traditional approaches do, but could still benefit from a light use of it, especially in prototypes creation, at the end of each testing phase. Agile, in fact, was found to be extremely useful in case of big knowledge gaps and uncertainty, but it had to be coupled with a clear object-driven goal, reached documenting each step after testing, rather than improvising at each step.

Apart from these differences, the teams split up in two groups also in the ways they used to approach prototypes. Some teams used a "black box" approach, building step by step upon the previous versions adding or removing features; the rest of the teams, instead, prioritized modularization, focusing on the minimum feature creating value, while putting non-critical features on the shelf. The two drastically different approaches also give very different results: in the first case, a high-fidelity prototype is created, with high level of structure and functionality, but low level of complexity; in the second case, instead, only critical components are manufactured in detail, while the rest of the product remains at a concept stage, in this way the team is able to get much faster to very complex products and functionalities, winning against its competitors. Prototypes were also found to make the project status clearer and to make the internal communication easier, as they were continuously representing the project status, improving the understanding of each component and of its constraints. This allowed for an object-driven path, not predefined and free to adapt to the changing requirements.

Speaking about the agile product development, the authors state that in such an agile environment, it is necessary to combine both a "technology push" and

"demand pull", and that transforming customer needs into product requirements asks for a joint understanding of the complete situation within the team. In the case study described by the paper, this was achieved by means of a prototype roadmap, that supported the visualization of the ideas: starting with an abstract vision, the prototypes became more concrete with each iteration. With each iteration, the team gained new insights, learned new lessons, and opened to new possible features, increasing the range of options for the future steps, rather than reducing it as it happens with traditional approaches. In other words, this allowed great flexibility and permitted to welcome change in the form of suggestions and feedback from the outside. The teams first "focused on building the right product [...] before building it right" (Böhmer et al., 2017). After these first steps, however, the need for traditional methods increased, and shifting to a well-documented process represented a new challenge. Similarly, during this shift, teams also shifted from horizontal prototypes to more vertical prototypes. But both approaches were still needed up until the end, in order to explore the complete solution space and the best results were achieved by the teams that frequently changed perspective, leading to well-defined projects. Agile, in fact, supports vertical prototyping, whereas traditional methods support the horizontal one, focusing on a single strategy the teams remained stuck at the preliminary phases and were unable to continue the project. Also, the teams that followed a smoother transition from agile to traditional methods got the best results, compared to the ones that abruptly switched from one to the other.

Other insights from this paper show that following a traditional approach, some teams got consistent results, with the only drawback of being unable to adapt to changes. The goal should then be to merge traditional and agile frameworks, in order to benefit of these two approaches in the best way.

Following with this list, in Cooke et al. (2012) paper, a bicycle stability test bench (i.e., appliance to make bikes more stable and safer) was created, in order to study the application of agile methods to a mechatronic, multidisciplinary project. Even though the dimension of the project is quite small, this case study was considered a useful opportunity, since it shares many of the issues related to communication and project management that bigger projects usually show. As an example, in

an ideal situation the computer model at the basis of the test bench should have been developed before the hardware components and, only after the test bench was built, properly tested; however, in this situation the computer model and the test bench had to be developed simultaneously, for a series of internal reasons, and customer deadlines. In other words, the project needed a good management strategy and development framework to ensure that the test bench was delivered on time. Moreover, the developers were continuously getting inputs from different sources at different times. To face these issues, the team created a system that not only pushes work into the system by a schedule, but also demands it to be pulled and makes sure the milestones are reached on time. Given this setup, the team applied the project management benefits of Scrum and the best practices of XP to determine if it can be used effectively in such a project.

Milestones for the project were created using systems engineering principles. They were then used as demonstrations to stakeholders (like prototypes) to make sure that they could see what was being developed. Usually CAD models, 3D printed models, or simple printed sketches were used as demonstrations. Milestones were then re-evaluated, applying changes according to the customer requirements. In this process, the team also used just-in-time and continuous planning approaches. In between the milestones, instead, Scrum was applied. Similarly to its original form, this version of Scrum introduced fortnightly Scrums (similar to daily Scrums), monthly demonstrations with important stakeholders, and milestone demonstrations, with the presence of all stakeholders, every two or three months. For what concerns XP, the following best practices were implemented during the Scrum cycle: the 'Planning game', 'Small Releases', 'Metaphor' and 'On-site customer'. The 'Simple design' was also directly applied to the project, supported by the usage of modularize components. Industrial or academic standards were then applied throughout the project, as well as the principle of not remaking the wheel, and re-using software or hardware from diverse sources. Test driven design provided strong, usable sub-systems, and unit tests were used within the software components. Continuing in this list, collective code/product/system ownership was encouraged, and the same holds for the '40-hour work week', and the coding/working style.

Regarding the conclusions brought by the authors, it has been shown that agile

methods can be applied to such multidisciplinary systems, as the differences between hardware and software is diminishing thanks to CAD, RP, and other technologies. Additionally, they provide a useful method to evaluate the effectiveness of such a Srum-Milestone approach by checking that applying it to a given project is: technically feasible, technical valuable, practically feasible, and practically valuable. This means to check whether it works or not, and if it works, is it better than other methods? But also, does it work for all stakeholders, and does it add any value to the project?

The third descriptive contribution comes from Enkler and Sporleder (2019). They tried to couple explorative and established Computer aided methods (denominated *CAx*, in general) to what they called "virtual product development", and in particular during its early stages. Their research brought them to the conclusion that successful projects tend to use more methods and apply them much earlier in the product development process, especially in its early stages. Here, CAx methods allow the designers to start from an abstract idea, develop it, and continuously increase its perceptibility through shorter product cycles, as discussed in the previous sections. An interesting thought added by the authors, however, is that those tools do not necessarily lead to innovations by themselves. Innovation requires creativity, that is often achieved with simple tools like a paper sketch, and without creativity in design, there is no potential for innovation. On the other hand, instead, CAx methods allow reducing costs and saving time, especially in the time span from the product idea to prototype construction. This set of effects, according to the authors, can be enhanced by coupling different methods, forming hybrid versions of them. As an example, instead of applying computer-aided design (CAD), computer-aided engineering (CAE), or computer-aided manufacturing (CAM) for specific purposes, coupling them would allow a larger spectrum of applications. Enkler and Sporleder (2019) went even further in that sense, coupling these techniques with other ones:

- 3D scan
- Reverse engineering
- Topology optimization
- Hybrid CAD modeling, especially using NURBS (Non-Uniform Rational B-Splines)

- Additive manufacturing

In their work, three use cases have been analyzed. The first one regarded the development of an exoskeleton, and also served as an example of mass customization through agile and multiple CAx methods. This kind of products, in fact, require extreme individualization levels, since they literally have to be designed to match someone's body for the sake of user-friendliness, usability, and ergonomics. Not only that, they are often required to be rapidly developed, as the customers/patients need them to perform essential activities. In this case, modern techniques were used according to the diagram of Figure 2.2. For this object, design was the only concern, while physical and mechanical properties were almost ignored.

For the second use case, instead, some students had to develop a bottle holder under consideration of boundary conditions in terms of fastening and load cases, and requirements for which the object had to be lightweight, robust and visually attractive. In this case a CAD system was used in parallel with topology optimization, and NURBS. This allowed students to simulate a lot of different conditions, obtaining a wide set of design ideas from which they chose the final one. And everything was done very quickly.

Finally, the third use case was an interior trim for a recreational vehicle manufacturer, that helped to investigate the possibilities of coupled CAx methods in case of blurry product requirements, lacking of data, and uncertainty. The aim was to generate an MVP (Minimum Viable Product) to gather initial information from potential customers. The tools exploited were 3D scanning, reverse engineering, hybrid modelling, and hybrid manufacturing. These methods helped the team to implement design even with lacking input data, for example by scanning the interior frame and getting precise measures.

According to the authors, in addition to the process-related incompatibilities, designers and engineers also face difficulties due to the differences in their ways of thinking and working. From the above-mentioned use cases, it has been noted how they can improve this aspect: CAx methods help to close the gap between the two categories of professionals. In particular, 3D scanning and reverse engineering help in complex situations in which manual activities of the designers are associated with a

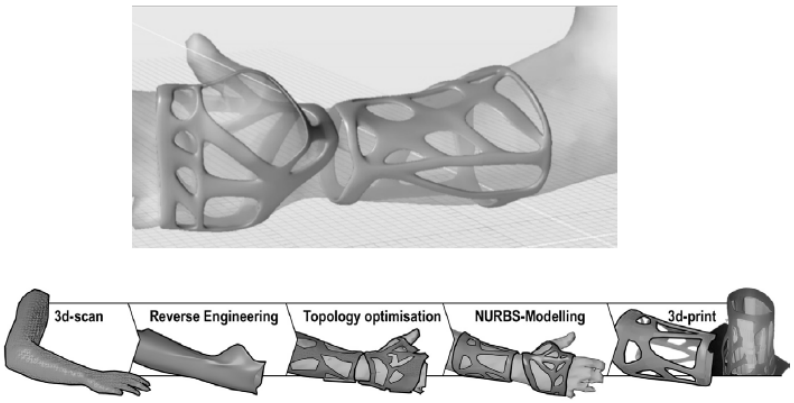

Figure 2.2: Exoskeleton development by means of modern techniques (Enkler & Sporleder, 2019)

digitization process. According to the authors, 3D scanning is also predisposed to an agile way of working, especially if coupled with 3D printing. They provide significant contributions to early product development stages and create added value. Similarly, the early use of simulation software creates the basis for joint developments and thus a basis for discussion, suggestions and new ideas. Hybrid CAD systems form an optimal framework for creative and collaborative development processes, and topology optimization supports professionals in form finding (Enkler & Sporleder, 2019). The authors are sure that in the future, this way of couplings methods will become very popular in interdisciplinary development projects, especially in case of lacking input data and, in general, during the early phases of development, when critical decisions are made. They also provide a solution for quickly reacting to changes in boundary conditions, and allow the creation of MVPs, an essential tool for future processes.

On the pain points side, Enkler & Sporleder identify the complex interface between individual isolated solutions, and the absence of software standards and data formats standards. Another suggestion provided is to always maintain simplicity, in order to remain flexible. This would also allow good usability and plausibility checks even by non-specialists.

The fourth descriptive paper analyzed comes from Ismail et al. (2007). The

researchers' team focused its work on Mass Customization (MC). This trend helps companies that operate in markets where offering customers more choice is the only remaining differentiator, but according to the author, small and medium enterprises (SMEs) struggle in achieving such results, and agile could help them substantially. The goal is to provide increased variety to customers, without increasing costs. MC should provide competitive advantage, not just customized products that put even more pressure on the company. The authors highlight how the customer presence can be exploited in the early stages, to let customers adapt the products by themselves; but even in a more embracing way customizing the product sale, design, fabrication, assembly, and delivery. The paper also refers to another work from Gilmore, Pine, et al. (1997), that in turn identify four customization levels mostly based on empirical observation: collaborative, where designers working closely with customers; adaptive, where standard products are changed by customers during use; cosmetic, where packaging of standard products is unique for each customer; and transparent, where products are modified to specific individual needs. In this way, the customers provide the product customization, that is then coupled with the process repetitiveness given by the modularity that every MC project should have. Customers are, in fact, just one of the actors involved, together with suppliers, distributors, retailers, manufacturers, retailers, and other value chain entities that need to cooperate in such a complex network.

According to the authors, four main business practices are relating to the MC concept: customer-driven design, agile manufacturing, supply chain management and lean manufacturing. As a matter of fact, the whole MC trend can be perfectly coupled with agile methods to respond effectively to market demands, as well as being proactive enough to create new markets and opportunities. Agility, however, depends on a wide range of capabilities, and many studies have approached it in different ways. Ismail et al. (2007), instead, created a framework that integrates all its main aspects under a single umbrella, and not treating them as individual entities: this is done through examining existing product structures and platforms, assessing how a degree of modularization and rationalization that could be introduced to simplify manufacturing complexity while maintaining product flexibility. The aim is to reduce the impact of the different customers' demands on the company operations (Ismail et al., 2007).

This framework is described with detail in the original paper, and a lot of attention is placed on the concept of MC, how to properly achieve it, and how to measure product similarity, a key concept for MC, but quite difficult to evaluate. A similarity matrix approach is then suggested, making use of different coefficients such as *Product Structure Similarity Coefficient, Product Cost Similarity Coefficient, Product Volume Similarity Coefficient, and Product Contribution Similarity Coefficient*, they all converge into the *Aggregate Product Similarity Coefficient*, according to a precise formula, invented by the authors. This approach was then applied to two case studies, the first one focused on a company that designs and manufactures shower enclosures and bath screens for the top end of the market; the second one considered a toy manufacture that designs and manufactures swings, slides and a variety of playground centers.

From these two studies, the authors described few effects that agile and MC have on such processes. In the first study, the design of new products was guided by the measure of similarity and flexibility of the existing products, coupled with an assembly approach. This permitted the company to reduce the number of unique extrusion profiles, since the same design could be used by more than one product. In parallel, this method also helped them to introduce an ERP system. This approach had consequences on the relation between the company and its suppliers: relying on a single design, many more products were affected by a delayed or missed shipment. For this reason, a closer relationship with the suppliers and a robust stock management system were found to be essential to the success of MC. Additionally, to properly apply MC, the company should integrate it as part of the product design procedures, and not only at the product review stage: the company has to move as a unitary entity, composed by manufacturing, design, marketing, and accounting staff. Again, this similarity between MC needs and agile best practices suggests a possible collaboration between the two. One last observation pointed out how the MC product platform appeared to cost more than the existing one, leading to the conclusion that some optimizations are needed in order to make this approach sustainable for enterprises.

In the second case study, instead, MC highlighted the issue of unnecessarily high number of raw material tubes used by the company observed. Ordering the

tubes in precut lengths contributed to the problem. With MC, it was possible to reduce the number of material grades, geometric sizes, thickness, lengths, and so on. Of course, this impacted the final products as well, and the design team had to work on these variations. In this case the team did not perform a cost analysis, but the proposal of reducing both the number of components of each product, and the components' variability were taken into consideration by the procurement manager, in order to overcome what was defined as a supplier constraint.

In conclusion, MC seemed to have possible positive effects on SMEs, even if they do not have the same resources of bigger companies. Agile could play an important role, since monitoring of the implementation process appeared to be critical, especially for SMEs that do not have the capacity to achieve everything on their own. Similarly, since MC is strongly affected and depends on the external influence from customers and suppliers, the diffusion of the agile philosophy in such companies would probably fit very well.

Concluding this list, the last descriptive work hereby considered comes from Vinodh et al. (2010), who based their research around CAD methods and Rapid Prototyping, applying them to an agile development process in which a pump was being designed. The chosen company for this study was 'Mayur Motor Industries' (abbreviated as Mayur), a SME located in Coimbatore, India. The goal was to demonstrate how these two tools could help traditional companies in their transition towards agile. As many other practitioners, Vinodh et al. also considered RP as a technology that could generate time compression, increased flexibility, and cost reduction. CAD, on the other hand, is fundamental to use RP. Customer requirements fulfillment also played an important part. Firstly, requirements were collected in the form of product specifications, and then converted into product specifications. Once finalized, the solid models of those products were developed using CAD, validated using computer-aided analysis (CAA), and finally the prototypes were produced by employing RP technology. During the first step, indeed, CAD models were created with Pro/E and their relative flow analyses were performed via specialized software packages like GAMBIT and FLUENT. During RP, instead, the chosen method was FDM (i.e., fused deposition modelling). This process utilizes fused thermoplastic filament as the base material, extruding it

through a nozzle and then depositing it in a proper geometry, under the control of a computer.

At the end of the development process, both the Managing Director and the Design Engineer of Mayur, as well as the design engineers of another pump-manufacturing company located in Coimbatore (Aquasub Engineering, abreviated as Aquasub) were asked to fill in a series of systematically designed questionnaire to measure their feelings about such an agile approach. Their answers indicated that "integration of CAD and RP technology interfacing for achieving agility is a practically feasible proposition" (Vinodh et al., 2010). A summary of their feedback is provided in Table 2.4. The study suggests that such an approach could help traditional industries (like the manufacturing one) to bring out different varieties of products within a short period of time, like other industries are able to do (e.g., the smartphone industry). CAD and RP technologies could act as agility enablers and bring the manufacturing closer to the modern market requirement, surviving the competition and anticipating changes in demand.

Alongside the pure research around CAx and RP implementation, the team also performed a cost analysis to estimate the financial impact that agile product development had on the company used for their test. First, the contribution of each agile activity was evaluated, as shown in Table 2.2. Furthermore, every agile activity was examined to understand how each of the following agile criteria was affected by it: Innovation, Quick Responsiveness, Mass Customization, Customer Relationship Management, Competitive Advantage. As an example, the activity "Understanding of customer requirements" has a total cost of 32,500 INR, that can be attributed in different portions to the five agile criteria: 6,500 INR for Innovation, 6,500 INR for Quick Responsiveness, 9,750 INR for Mass customization, and so on. These numbers were assumed by the authors thanks to calculations based on the number of hours dedicated to an activity, or the resources employed. The results can be seen in Table 2.3.

Agile Product Development Activities	Cost (INR)	Weight
Understanding of customer requirements	32,500	6.9%
Solid modeling	110,500	23.5%
Development of improved designs	74,500	15.9%
Design validation using CAA	100,500	21.4%
Prototype development	152,000	32.3%
Total	470,000	100%

Table 2.2: Apportionment of each agile activity's cost (Vinodh et al., 2010)

Agile Criteria	Cost (INR)	Weight
Innovation	73,650	15.7%
Quick Responsiveness	70,050	14.9%
Mass customization	124,800	26.6%
Customer Relationship Management	67,000	14.3%
Competitive advantage	134,500	28.5%
Total	470,000	100%

Table 2.3: Apportionment of each agile criteria's cost (Vinodh et al., 2010)

2.3.2 Normative approach

The first normative paper of this list comes from Mabrouk et al. (2018), who proposed to couple and integrate agility and Model-Based Systems Engineering methodology (MBSE), in particular for multidisciplinary systems design. The idea behind this, is that multidisciplinary collaboration – suggested by agile – could also be efficiently supported by the MBSE approach. However, "this approach does not avoid system defects during or after the development process" (Mabrouk et al., 2018). Their suggestion is to integrate agile – and in particular Scrum – into this approach, to identify defects before the system components are fully integrated, and to avoid expensive delays, both in terms of cost and time.

The "original" method is composed of two phases, namely the black box analysis, and the white box analysis. The black one consists of determining the main purpose and mission of the system, as well as its life cycle. Then, for each phase of this cycle, the corresponding requirements are collected. The white box analysis, that comes after the black one, has the purpose of building a physical architecture of the system. The architecture is gradually identified by starting from the system functions established before. Finally, alternative physical architectures can be defined and, based on some performance simulations, the final architecture is

#	Designation of the respondent	What is your overall opinion about CAD and RP integrated agility?
1	Managing Director — Mayur	Number of trials and overall development time can be reduced by implementing CAD/CAM driven agility
2	Senior Manager — Aquasub	By using CAD, CAM and CAA, I feel we can reduce the overall product development time by 50%
3	Senior Engineer — Aquasub	Improvement and research on material of prototype will really help the CAD/CAM driven agility
4	Testing In-Charge — Mayur	Time involved in product design and development can be reduced

Table 2.4: Overall opinions about the use of CAD and RP (Vinodh et al., 2010)

chosen. The validation process is made at the end.

In the authors' method, instead, agile is introduced between the two analyses. The set of requirements found at the end of the black box one, the product backlog, typical of Scrum, is created. From here, the first sprint is defined, and the whole white box analysis is performed as a series of Scrum iterations, followed by multidisciplinary teams. Agile, however, needs the creation of prototypes at the end of each cycle. For this reason, a new step is added: a partial system prototype is created at the end of each sprint, and a preliminary validation process is performed on them. This validation replaces what in Scrum is called "review" and, at the same time, substitutes the final massive validation process needed in the original method, allowing new architectures to be tested sooner, and the removal of inappropriate solutions.

According to the authors, the agile method introduction provides a great flexibility. The requirements can be modified according to some initially unforeseen constraints, or to the customers' changes. Apart from the money- and time-saving effects, this provides better quality to the product. The iterative approach, coupled with the frequent testing and validation steps, limits the integration issues between the partial architectures and ensures the development continuity of the final architecture (Mabrouk et al., 2018).

The second normative paper is signed by Riesener et al. (2019). It presents a methodology for the design of product development networks, to increase organizational agility. This method makes use of three parameters: agile attributes

(AA), that characterize the agile process; network features (NF), that describe the network design; and strategy-specific competitive bases (CB), that companies might develop when applying agile methods. AAs and CBs were derived from the literature, and are reported in Table 2.5, while NFs should be defined according to the project's specific characteristics. These parameters are linked by two different Houses of Quality (HOQ), the first one determines the importance of each AA according to the company's CBs; the second one relates these weighted AAs with the NFs. The methodology is represented in Figure 2.3.

Competitive bases	Agile attributes
Speed	Cross functional teams
Profitability	Decentralized decision making
Quality	Short development cycle times
Flexibility	Response to changing market requirements
Innovation	Adaptive and learning organization
	Making decisions frequently and act fast

Table 2.5: Competitive bases and agile attributes used by Riesener et al. (2019)

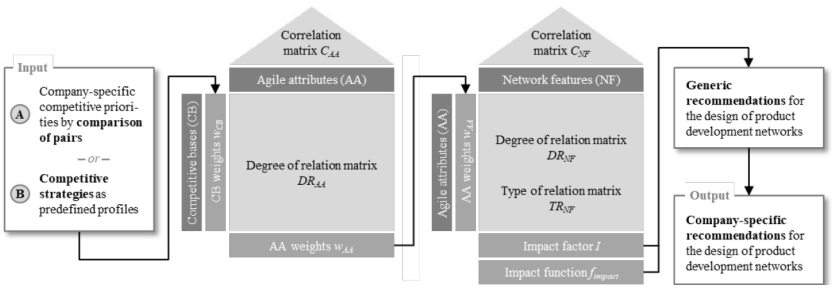

Figure 2.3: Framework of the proposed methodology (Riesener et al., 2019)

As a result, this method should help companies determine which NF could be optimized to get an efficient prioritization in network management, also suggesting the direction of optimization of such NF, according to the current network status. The results are generic recommendations that give an indication of how the design of the network should be adjusted to improve its agility.

2.3.3 Final observations

In light of the previous sections' content, some preliminary conclusions can be drawn. As the literature suggests, agile practices seem to have potential applications in quite different sectors. Projects that involve multidisciplinary teams, or that need very different experts to cooperate, can highly benefit from implementing agile. This also includes the design and development of any product that involves both hardware and software, or any other technological element capable of receiving, transmitting, and even elaborate data. However, these are not the only industrial fields that agile could help. Scrum and the other agile techniques can be adapted to project management purposes, system management, and in general any design process, even in absence of software and technological elements.

One thing, however, joins all of them: the need to adapt agile to its new purposes. Almost every author highlighted how the direct application of these methods to new areas can not work. Every industry and every project should take into account the differences between their needs and the original software developers' needs, before taking the agile way. This issue might be sorted out in the future, if a standardization process will take place. Nevertheless, agile methods are widely believed to be applicable in hardware products development by performing a certain tailoring procedure, and focusing on important elements such as the customer involvement, and the use of new technologies.

To begin with, the iteration lengths of hardware development should be taken into account, creating longer sprints when needed. In the case of CPS and multi-disciplinary products, the dual nature of HW and SW should also be considered: as one SW sprint could last two to four weeks, and end up with a prototype in a cheap manner, the HW part that needs to communicate with it could take much more time and not lead to a completely new prototype every time, due to cost reasons. In these cases, the two parallel design processes should be defined in such a way that they fit each other and help the overall project to go on. Additionally, for the HW part, a certain degree of pre-planning could be coupled with the agile way of doing, in order to predict severe and costly issues due to regulations or physical limits. Another solution could be to produce MVPs instead of complete

prototypes, in this way the product's main features could be tested without incurring in excessively high costs. In Figure 2.4 an example of simultaneous hardware and software development is provided.

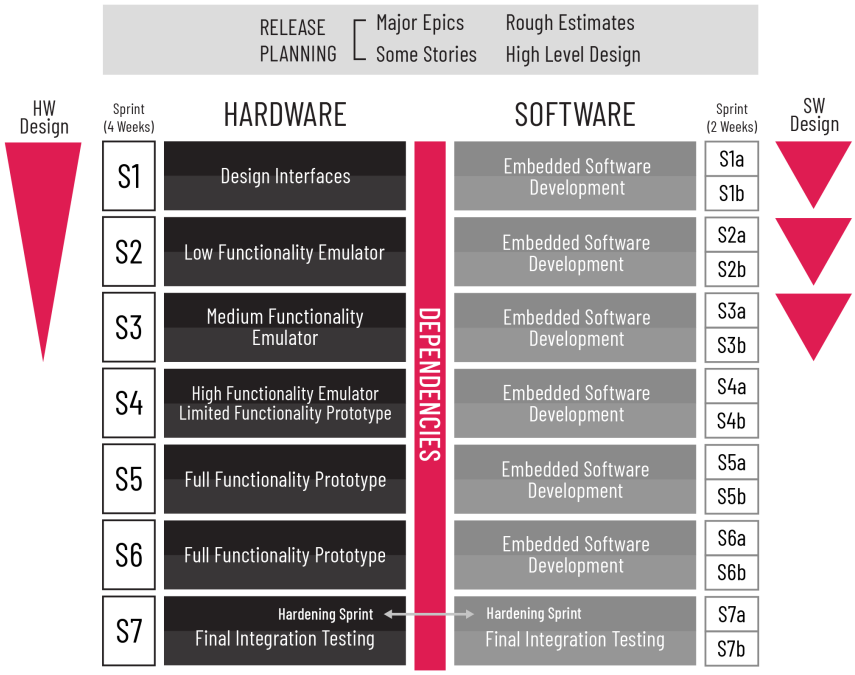

Figure 2.4: Example of the overall flow of development, for a hardware product that contains a software component (Thompson, 2021)

Another key difference lays in the mindset and culture that have to be seeded into the company to make agile profitable. Team members should have clear in mind the agile philosophy before its practices, they should understand how the managerial roles change in agile teams, and the whole company structure should be adapted to form multidisciplinary teams, with technical and non-technical people in them. The classical subdivision into departments needs to be taken down. For this reason, employees should be formed about new areas of the business, rather than being focused on a specific topic. The goal has then to shift towards the idea of a product that is designed as the process goes on, rather than something that is first thoroughly reasoned and then designed. The lightening of preliminary studies could indeed represent a big challenge. Understanding that the upfront planning can not be implemented anymore can take time, thus everyone should

also get specific training about the time-boxed iterations approach, the acceptance of change, the absence of documentation, and so on.

Customers also play an important role, since they need to be involved. Even if this already happens in traditional industries, it could be difficult to strongly apply this methodology with prototypes and partially working products. The customer itself should be aware of the whole process in order to provide adequate contributions.

Finally, shifting towards agile, manufacturing companies should also try to embrace a new way of doing business. Exploiting these methods could open the doors to Mass Customization and platform design. In this way, different products could be developed by the same multidisciplinary team – thanks to the platform approach, which would support a number of different products developed over the same basic design – and customized to fit a multitude of different customers, without impacting the economics of the design process.

In Table 2.6 it is reported a summary of the main differences between hardware and software agile development. All these differences have implications at different levels: operational, organizational, and strategical. Table 2.7 summarizes them.

Software	Hardware
Beta versions	Prototypes, Minimum Viable Products
Short, fixed length iterations	Iterations of variable length according to product architecture, complexity, and openness
Domain-oriented knowledge	Multidisciplinary teams
No documentation	Light documentation
Light negotiation with suppliers	Tighter relation with suppliers is needed
Low cost of design changes	High cost of design changes
Nearly absent sunk costs	Important presence of sunk costs

Table 2.6: Differences in applying agile to SW development and HW design

In the previous sections, a number of literature contributions has been analyzed. Their outcome is a wide set of normative suggestions and direct observations about the applicability of agile methods in manufacturing processes. As a consequence, a set of recurring trends were identified: the importance of customer requirements and the customers' involvement in the agile process; the use of prototyping and RP technologies; the use of agile methods. For this reason, in the following pages

Level	Effects
Operational	• Mandatory shift towards some technologies: CAx, Rapid Prototyping, Virtual Prototyping, Additive Manufacturing, Reverse Engineering, 3D Scanning • Diffused use of virtual prototypes, simulations, and digital versions of products • More frequent prototyping phase • Need to understand how to deal with software and data elements • Implement platform design to achieve economies of scale • The short term goal is the creation of a prototype at the end of each iteration • The long term goal is the final product, but has to be achieved through rolling wave product design
Organizational	• Stronger relationship with suppliers, due to product platform needs • Strong involvement of customers in the company organization to get feedbacks • Shift from functional departments/business units towards dedicated multidisciplinary teams • New roles have to be created (e.g. Scrum Master) and managerial practices have to change
Strategical	• "Focus on building the right product, before building it right" (Böhmer et al., 2017) • Looking at business opportunities, new products can be created thanks to MC and platform designs

Table 2.7: Effects of agile on HW development

a tabular framework of the previously mentioned literature researches has been provided. The subdivision in descriptive and normative papers is maintained, but a third, mixed, approach is added. In this last category, one could find those researches with a dual nature. In addition to the researches mentioned in Sections 2.3.1 and 2.3.2, the main sources of information previously used in this chapter are also included.

Paper	Customer requirements	Prototyping	Methods	Sector
Böhmer et al. (2017)	Transforming customer needs into product requirements asks for a joint understanding of the complete situation within the team. Prototypes are useful to adapt to changes in customers requirements.	Prototyping brings hardware development closer to agile software development. Iterating between different uses of prototyping leads to well-defined projects. Agile principles support the horizontal prototyping strategy.	Various methods, free choice was given to the teams.	Students' challenge (Mechanical Engineering, Informatics, Computer and Electrical Engineering).
Cooke et al. (2012)	If a project gets inputs from different sources it can be implemented a "pull" type system: work is pulled into the project and not pushed by a schedule. Milestones are then re-evaluated and adapted to changes, according to user feedback.	In the case study, milestone deliveries were usually done via CAD models, but also 3D-printed versions were used, to get physical prototypes that users could test.	XP best practices & Scrum project management beenfits.	Bicycle test bench – case study.
Cooper (2016)	Stage-Gate/Agile hybrid shows customers something tangible all the way through the project, allowing product requirements change when cost of change is lower and satisfying customers. The tangible items shown to customers may not be working products, but something physical that the customer can respond to.	3D printing and computer simulation make hardware development closer to software development, easing agile hybrid models. *"Protocepts"* 3D drawings, virtual prototypes, crude models, working models, or early prototypes.	Scrum; Stage-Gate/Scrum hybrid: modified agile approach with agile sprints and scrums for both physical and IT development within Stage-Gate phases. Agile is employed in particular in the development and testing stages of the Stage-Gate process.	US Consumer Electronic, Heavy Equipment sector.
Enkler and Sporleder (2019)	Creativity is needed to satisfy customers, it is not enough to reproduce existing knowledge. Coupling established and explorative methods can be helpful to reach customers' requests, even in highly individualized products.	Prototyping is essential in innovation phases, it can be coupled with CAx methods and hybrid modeling environments. 3D print is naturally predisposed for agile. MVPs can be used to gather additional information from customers.	3D scan, topology optimization, hybrid modeling, NURBS, reverse engineering, AM methods, 3D print.	Three case studies in biomedics, automotive, kitchen tools.
Ismail et al. (2007)	SMEs offer customers more choice without attention to the impact of this on their operations. Mass Customization allows them to provide customization while keeping modularity and repetitiveness. Agile manufacturing and agility marketing can be helpful.	Never mentioned in the paper.	Mass customization involves: customer-driven design, agile manufacturing, supply chain management, lean manufacturing.	Small medium enterprises in very specific sectors: bathroom appliances; outdoor toys.
Vinodh et al. (2010)	In this paper customer requirements are translated into product specifications at the beginning of the project, the customer is not involved after.	RP can be an agility enabler, but needs CAD methods to be used jointly.	Agile manufacturing only.	Pump manufacturing.

Table 2.8: Descriptive approach papers

Paper	Customer requirements	Prototyping	Methods	
Mabrouk et al. (2018)	Agile methods between the black box and white box analyses (from MBSE methodology). Improved requirements, adaptation to customers' changes, higher product quality.	Prototype delivery at the end of each sprint. New architectures and components are found, forming the new "induced requirements", added to the product backlog.	Model-Based Systems Engineering (MBSE); Scrum	System Engineering; Mechatronic Systems
Riesener et al. (2019)	The paper presents a methodology to design agile product development networks, exploiting QFD and crating relations between Agile Attributes (AA) and Network Functions (NF) in accordance with the company's competitive strategy. Both customer requirements and prototyping are present in the AAs list.			No specific industry is selected.
Sommer et al. (2014)	The reward systems and the KPI system applied in the case studies contrasted with the rigidity of the system and the level of change in PD projects, there should be more flexible and adaptable reward solutions.	Not mentioned in the paper	Scrum governance, Hybrid Scrum/Stage-Gate.	Pharmaceuticals, plastic toys, electronics, windows, cross-country power lines.
Sommer et al. (2015)	Hybrid process decreased customer complaints and late-stage change orders. It increased customer collaboration in early stages of product development and market success. The business manager acted as voice of the customer, providing detailed knowledge of customer needs and direct communication with customers involved.	In the proposed framework, prototyping is used in the feasability study, where pre-prototypes are created after developing the initial product backlog. If this leads to a positive decision, the development goes on with the real prototype.	Hybrid Scrum/Stage-Gate. A new Industrial Scrum Framework is proposed after having analysed multiple case study.	Wind turbines, valves and sensors, insulin, plastic toys, music amplifiers, windows, and cross-country power cables.

Table 2.9: Normative approach papers

Paper	Customer requirements	Prototyping	Methods	
Mulder et al. (2014)	Crowd funding, co-development and social media should be incorporated as much as possible to survive competition, integrating them with Scrum to clearly define requirements.	The attention of stakeholders is shifting towards prototypes and early versions: new products might be put on the market earlier, even if hardware updates are more difficult. Rapid prototyping allowed faster prototyping nevertheless most students preferred more time for the hardware elements.	Scrum: it helped reducing documentation, quickly generate user knowledge value, allowed non-experts to work on the project and let students apply emerging technologies at low cost from the early phases of the project.	Cyber Physical Systems, sail simulator case study.
Könnölä et al. (2016)	Changing requirements can create more problems for hardware development than software, so must be taken into account when setting the iterations length. In absence of the customer during reviews, implementation is difficult. Also the methods used depended on the customers.	Never mentioned, the projects studied were not delivering a physical product at every iteration: the primary measures for progress were demonstrations, rather then a new prototype version.	Mainly Scrum, with modifications from each team depending on the customers' needs, on the project, and on the team members.	Mainly Scrum, with modifications from each team depending on the customers' needs, on the project, and on the team members.
Stare (2014)	Higher client collaboration proved to increase the overall financial success of the projects.	Not mentioned in the paper.	Partial approaches that have been established by the individual teams/managers, not a proper systematic agile approach. Most of the teams used short iterations and teams focused on individual functions.	21 different Slovenian enterprises, industries are not specified.

Table 2.10: Mixed approach papers

Chapter 3

Case studies analysis

In this final chapter, six different case studies are analyzed. The purpose of this analysis is to investigate whether the preliminary conclusions drawn in Chapter 2 may find confirmation or not. Again, the main source of information for this chapter comes from scientific literature, but the focus is placed on researches that analyzed in detail the application of agile methods to the manufacturing world, considering both HW and CPS design and development.

The selection of case studies aims at providing information coming from a variety of situations, industries and timeframes. The following papers, in fact, range from 2012 to 2019, they describe different industrial application of agile, and as a consequence they reach different conclusions.

3.1 Introduction

The first two cases come from the automotive world, and follow two different product development processes held at Volvo Car Group, in Sweden (Eliasson et al., 2014). These first two contributions are perhaps the strongest ones, as they present studies performed in a multinational company, among the biggest in the automotive world, providing direct feedback from real life industrial applications. This means that the projects and processes involved are quite complex, and the

stakeholders involved are many. As for the products, both these case studies deal with CPS development, including HW and SW simultaneous development.

Continuing, the third case study hereby considered follows similar steps. The authors indeed focused on the feasibility of applying agile into mechatronics products development, formulating a set of goals and practices to support it in large-scale applications. Again, both SW and HW are involved and taken into consideration. Differently from the previous ones, this case does not focus on a single company, instead the authors looked up six different international enterprises, some of which already applying agile to some extent (Eklund & Berger, 2017). However, it will not be treated as six individual case studies, as the authors' approach – in which conclusions and observations are drawn from the collective observation of the projects – will be followed.

The next case study was instead included to provide a completely different view and add contrasting elements to this chapter. Mazzanti (2012), in fact, focused on a HW-only product, specifically luxury bathtubs, and applied agile product development techniques to its developing process. It is the oldest of the six papers, and describes the active role of its author, playing the role of the agile coach in an Italian SME for the whole duration of the project. Differently to the rest of this thesis, in this case study some lean manufacturing techniques are reported and applied. Despite the clear differences that the two philosophies present, this paper proved that they could be associated providing good results. Nevertheless, due to the aim of this thesis, the agile part will get most of the attention in the following chapter's analysis.

Finally, the fifth and sixth case studies here presented deal with the application of agile to medical devices development. Again, the products considered are purely HW ones, but due to their industry of origin, they present many differences with respect to the others. In these case studies, in fact, some important constraints were present in the form of regulations of the medical sector, hindering the possibility of having vague product requirements in the early project phases, and in general creating some challenges. For this reason, they provide interesting insights on how much HW product development, especially in specific sectors, can differ from SW development. The two studies are performed by two different teams of researchers,

that however share a good number of team members, reason why they are here presented together, while in the next pages they will have separated analyses. Moreover, one can be seen as the continuation of the other. Gerber et al. (2019) present a possible – generic – agile method for HW product development, with specific applications in the medical sector, and it does so while working on a microtiter plate development. Goevert et al. (2019), on the other hand, tried to apply the previously mentioned method to the same microtiter plate project, aiming at its validation and improvement.

3.2 Cases 1 & 2: Volvo Car Group

Agile Model-Driven Engineering in Mechatronic Systems

The automotive industry well represents the current transformation trend described in the previous chapters, as it has undergone a rapid transformation from a mainly mechanical industry into a computerized electromechanical one. Cars nowadays are composed of several mechatronic system and cyber-physical systems:

> A modern hybrid electric car has more than 100 electronic control units (ECU), collaborating in a complex in-vehicle network and executing several gigabytes of software [...] These systems include assistant systems like adaptive cruise control, safety-critical systems like autonomous emergency braking, but also mechatronic systems like electronically supported steering (Eliasson et al., 2014).

The traditional development processes used in the automotive industry have shown to be insufficient for handling such an exponential growth of software. For this reason, some manufacturers started to implement agile principles and best practices into their organizations.

Volvo Car Group (VCG) is one of them. Here, Model Driven Engineering (MDE) has been coupled with agile. For this reason, Eliasson et al. (2014) conducted a case study at Volvo's Department of Electric Development in Sweden, to better understand the challenges emerged from such an approach.

3.2.1 The background situation

Eliasson et al. (2014) performed their observations and interviews in an industrial environment that is already using (to some extent) agile techniques on a daily basis. The paper describes with detail the method adopted at VCG. Starting from the V-model, a widely used method in the automotive industry, it also includes concurrent development of hardware and software elements. In this process, also Mode Driven Engineering (MDE) principles are applied, making large use of models rather than documentation as a mean of knowledge transfer. In this case, models are referred to as "plant models". As indicated in Figure 3.1, the simultaneous development of different components convey into a unique stream by points E1 to E3, that represent the electronic integration points. Point P, instead, identifies the point at which software development should be production ready.

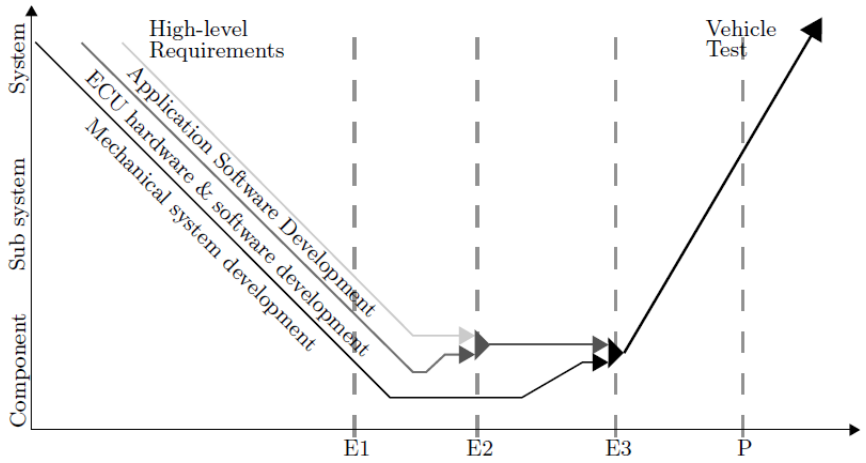

Figure 3.1: The V-model implemented at VCG for a car development project (Eliasson et al., 2014).

To make things more complex, it has to be kept in mind that VCG usually purchases all the mechanical systems, hardware, and software from external suppliers, except for the engines. However, during the recent years the company stated the desire of internalizing the software development process. This implies that the in-house software development needs to be started before suppliers deliver their components, in order to meet the project deadlines. As a consequence, developers

need to rely on their assumptions on the behavior of the components, since they are not available yet. Such a challenge is faced thanks to a custom-made tool, referred to as SysTool. SysTool allows the creation of models that include software components, the requirements to realize, their deployment on ECUs within the car, and the communication between them.

At this point, some agile aspects are integrated. The model is in fact frozen, and no changes are allowed until the next iteration. Differently from agile software methods, however, the iterations last 20 weeks. Moreover, each model is transformed into Simulink model skeletons. Each of them represents an ECU with skeletons of the deployed software components including ports and connections. These Simulink models can be tested in a virtual, model-in-the-loop (MIL), environment. This means that so-called plant models simulate the surroundings of the ECUs (including software and mechanical components) and enable to virtually and reliably test them, getting instant feedback.

Continuing with its description, the process also includes software code generated from the suppliers, starting from in-house models. The resulting software is tested in hardware-in-the-loop (HIL) test rigs, meaning that it uses real network buses but the rest of the environment surrounding the ECU is simulated. This point marks the first time that both, in-house and supplier developed software, can be integrated and validated together. After that, the last testing step involves the integration of software and hardware mechanical systems in a complete prototype vehicle.

This complex development/testing process, however, strongly relies on assumptions, on which the authors focused most of their effort, as described in the following Section.

3.2.2 Research performed

When software is developed in-house, mechatronic components produced by suppliers are usually not available yet. For this reason, the software development is based on "assumptions about the behavior and data to be expected from such hardware and mechatronic components" (Eliasson et al., 2014). Also, the project

timing and deadlines always make impossible the delay of software development to wait for the hardware components.

The goal of the study was to "investigate the challenges for MDE at an automotive original equipment manufacturer (OEM), when depending on assumptions during in-house software development" (Eliasson et al., 2014). To address this, the researchers based their work on three research questions:

RQ-1: What are the causes that lead to assumptions in distributed mechatronic development, and what are the consequences?
RQ-2: Does the combination of MDE and agile methods increase the knowledge in earlier phases of the project compared to a plan driven process?
RQ-3: What impact does faulty assumptions within the test environment have on the product and process?

The first one has been investigated through open-ended questions and interviews with developers, requirement engineers, testers, and architects, analyzing the challenges regarding software, hardware, and mechanical assumptions. The results of this preliminary study have been used to identify the main challenges and defined research questions RQ-2 and RQ-3. From here, two case studies have been designed to monitor these topics.

3.2.3 Results obtained

Starting from the open-ended questions part, in which the authors tackled the main issues related with the diffusion of assumptions and their consequences (RQ-1), some interesting conclusions could be extracted. First, the presence of assumptions in the early stages of the process was mainly attributed to the sequential nature of the product development process used at VCG. The fact that some stage-gate steps are still present, forces engineers to forge assumptions. In order to take decisions without having complete knowledge about the process, they need to fill this gap with them, as represented in Figure 3.2. As the authors point out, assumptions might be faulty and appear later in the process, when the cost of change is very high. The interviewees clearly stated that there is need for earlier and faster feedback to the developers and designers of systems. In other words, a need for

agile methods that, making use of increments, could allow faster decision-making, and building knowledge. This result shows how the implementation of agile methods in hardware and CPS development is not only suggested by the scientific literature, but also by engineers and practitioners of any kind, directly affected by the current methodologies' issues, implicitly asking for it.

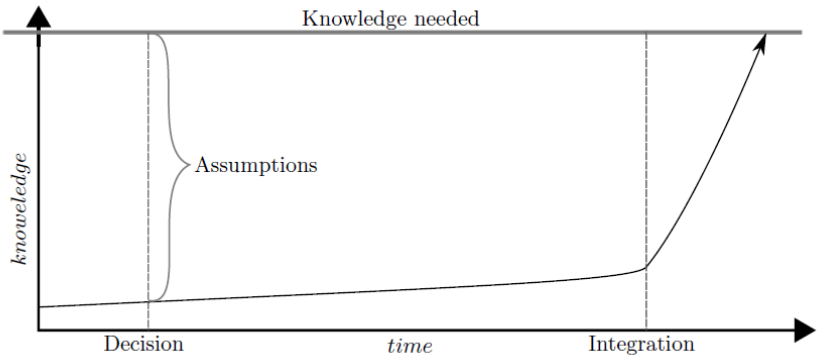

Figure 3.2: "The earlier an engineer has to make a decision, the higher is the risk of faulty assumptions that lead to unwanted side effects or defects, which need to be fixed later. Furthermore, these assumptions can only be verified at the integration points [...], any serious issues that are discovered at this time are obviously costly and time-consuming to fix" (Eliasson et al., 2014).

Following this preliminary interview, the researchers set up two case studies, involving respectively:

Case 1: The Electrical Propulsion Systems (EPS) unit, developing components for electrical and hybrid vehicles;

Case 2: The Central Electronic Module (CEM), responsible for an ECU of key importance within the vehicle.

In particular, the first case study followed the development of the clutch that is required in hybrid vehicles to safely engage the electric motor. The second case study, instead, analyzed the development of a software managing the active high beam headlight (AHBH), a modern tool that lets drivers use the high light beam of their car's headlight without blinding the other drivers by mechanically obstructing the light and put other vehicles in shade. Finally, it is worth to point out how the first case study made use of virtual testing environment, plant models, quick

iterations, CAx tools (especially CAE), and prototyping. All these aspects are related to the agile manufacturing concept. The second one, instead, started from a waterfall approach that was not giving the desired results, and was modified by an MDE expert. In this case the team made use of software modeling, virtual testing, and HIL testing on a real car.

From the two analyses, interesting observations concerning both RQ-2 and RQ-3 came out, as summarized in Table 3.1.

	RQ-2	RQ-3
Case study 1	Strong development of plant models was observed during the process (e.g. clutch going from simple model to a realistic one).	During the final coupling of SW and HW the clutch did not behave as expected (faulty assumptions).
	The workbench using plant models allowed for continuous software modeling during development.	The simulation environment allowed to go on without management approval, helped to start the project and communicate ideas.
	The virtual environment allowed for an otherwise-impossible flexible testing of early software.	
	During development, in case of faulty assumptions were discovered, both the code and the plant model could easily be corrected.	
Case study 2	Quick demonstration tools allowed for a lot of testing.	Due to assumptions, the early versions of the software were over-engineered: actuators and sensors were considered as ideal HW elements, detached from reality. This resulted in unnecessarily complex architecture.
	The HW fixing process related to the faulty assumption took one month.	Other faulty assumptions led to considering light beams as a perfectly calculated image, which turned out to be totally different once tested in a real car.
	The SW fixing process related to the faulty assumption was quick, and the overall design remained untouched.	Poor laboratory tests' settings allowed for the latter faulty assumption.

Table 3.1: Summarized results of the two case studies performed at VCG, in relation with the two research questions identified by the authors (Eliasson et al., 2014).

The two cases present many differences in the use of plant models, and in the implementation of more or less advanced simulation environments that led to more or less accurate assumptions. Also, they both highlighted some difficulties encountered by the teams. However, they both clarified how the use of virtual test environments significantly improved the team members' level of knowledge in early stages of the projects. In the first case study, virtual testing allowed to

reliably test the created prototypes, and get important assumptions about the externally-developed technology. In the second case, it was even defined essential, without virtual testing, the team had little or no progress at all. Virtual and early testing environments forced the developers to fill their knowledge gap, gaining correct – or almost correct – knowledge where it was missing.

The authors visually describe this finding by drawing the graph reported in Figure 3.3. Comparing it with Figure 3.1, where the traditional knowledge flow was depicted, it can be seen how at the integration point the knowledge gap reduced a lot. As mentioned, the results of the two studies were not always ideal, some mistakes were discovered in the later phases of the projects, leading to delays and money losses. In general, it has to be kept in mind that the risk of such occasions happening will always be there. However, the ability to conduct simulations this early significantly increases knowledge from the beginning of the development process, and allows to speed up the process.

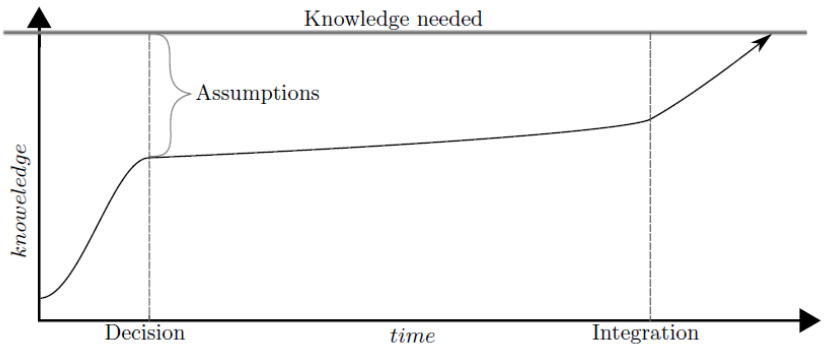

Figure 3.3: "By being able to integrate and execute tests more frequently using plant models and virtual environments, the developers can build and validate knowledge also early in the project" (Eliasson et al., 2014).

Among the negative outcomes of the two studies, the authors point out how plant models suffered from small inconsistencies, incompleteness, and needed of some simplifications due to the limited understanding of the mechanical principles involved. The lack of knowledge was also evident within teams, since the plant model's developers worked separately from the function's developers, lacking some contextual background. In a certain way this could be seen as a confirmation that

also during the development of HW products and CPS, the agile principle according to which team members should always work together in the same room still applies and could avoid some mistakes.

As a conclusion, agile integration with MDE seemed to work efficiently, thanks to the aid of a virtual testing environment. Without these implementations, it would have been impossible to relax the dependency on mechanical parts during the development. Short iterations were also made possible and, as discussed, knowledge in the initial stages of the process drastically improved. It was also noted how assumptions play a fundamental role and thus necessitate of great attention. Faulty assumptions, or assumptions exceeding the capabilities of the mechanical components could result in more complex and resource-demanding implementations.

About plant models, the authors identified two possible solutions that could decrease the amount of assumptions needed. First, suppliers could deliver plant models of their mechanical systems, giving the OEM more accurate models to base the development on. Alternatively, plant models could be built in-house, but verified by the suppliers. Both these solutions appear to have some limits as well. In the first case, the suppliers may not be willing to share their IP, thus the risk that they supply black boxes components, useless to the purpose. In the second case, the solution presents limits related to the time and money effort needed to implement and, more importantly, maintain the plant models, as well as to integrate them with the MIL environment.

Regarding the purpose of this thesis, Eliasson et al. (2014) proved that agile can be implemented in an automotive production plant. In this specific case, the software presence made the use of less traditional methods more obvious, but still very innovative and successful, as anticipated in Section 2.1 when discussing CPS and Multidisciplinary projects. Here, no agile SW development methodology was directly applied to the HW domain, agile principles were instead adapted and integrated in an MDE environment with success. For what concerns the tools used, the main two were Virtual Prototyping and Rapid Prototyping, confirming what has been found in the previous analyses and sustaining the hypothesis about their potentiality to fostering agile implementations. For the future of this specific

application, it would be ideal to reach further automation of parts of the testing, and reach continuous integration and deployment. Both in a model based virtual test environment, and in HIL testing.

3.3 Case 3: Scaling Agile Development in Mechatronic Organizations

The following case study focuses on the mechatronic sector. In particular, Eklund and Berger (2017) identified "a set of goals and practices to support large-scale agile development in companies that develop software-intense mechatronic systems". They collected empirical data from six companies in the Nordic region, over two years of researches.

According to the authors, the issue with mechatronic systems has its origins in the duality that they possess. On one hand, manufacturing and hardware development have long lead-times (1-4 years), and during their development the attention is given to predictability, and to the need of meeting the start of production with the required mechanical quality. Usually this is done by implementing waterfall/stage-gate processes. On the other hand, software development is all about increasing speed and being more nimble while keeping quality. In this case, lead-times are weeks- or at most months-long.

Therefore, some methods to overcome this intersection are needed, especially for the development of mechatronics components. In fact, usually teams are able to implement software features in 2-4 weeks cycles, while the overall R&D process is still depending on a stage-gate or V-model structure, that takes up more time, vanishing the agile SW development benefits.

3.3.1 Research performed

To tackle this issue, the authors compared experiences and practices from six international companies, some of which were already implementing agile in their SW development, but did not scale them to the HW domain yet. The

study included only companies falling within very specific requirements: large mechatronics organizations dealing with a large and diverse product portfolio with regular product upgrades, where timely manufacturing plays a large role, and with strong demands on high quality and safety.

The authors' main goals can be summarized by looking at the two research questions driving the study:

RQ-1: What are expected benefits and challenges when scaling agile principles beyond software development teams?

RQ-2: What key enabling practices are considered to scale agile in mechatronics companies outreaching pure software teams?

To get the desired answers to the previous questions, the two researchers made use of individual on-site workshops with each company; complementary online surveys; joint workshops with all the company representative together; final individual interviews.

3.3.2 Results obtained

As a result, Eliasson et al. (2014) extracted a total of 409 individual statements and 108 goals. This large amount of contributions has been processed and categorized, getting to more punctual results. As an example, it is possible to note how participants mentioned nearly three times as many benefits as challenges. The complete lists of expected benefits and foreseeable challenges are reported in Figures 3.4 and 3.5, answering RQ-1.

From the two infographics, it can be noted how the principal expected benefit was higher quality, while the main challenge was lack of flexibility in testing facilities. According to the authors, the benefits list highlights the expectation of quicker and better feedback to developers, compared to the previously used methods.

As anticipated, Eklund and Berger (2017) also identified 108 goals and practices from these investigating activities, trying to find valid answers for RQ-2. The majority of them were, however, overlapping with well-known best practices already established in agile and lean software development. On the contrary, 26 goals and

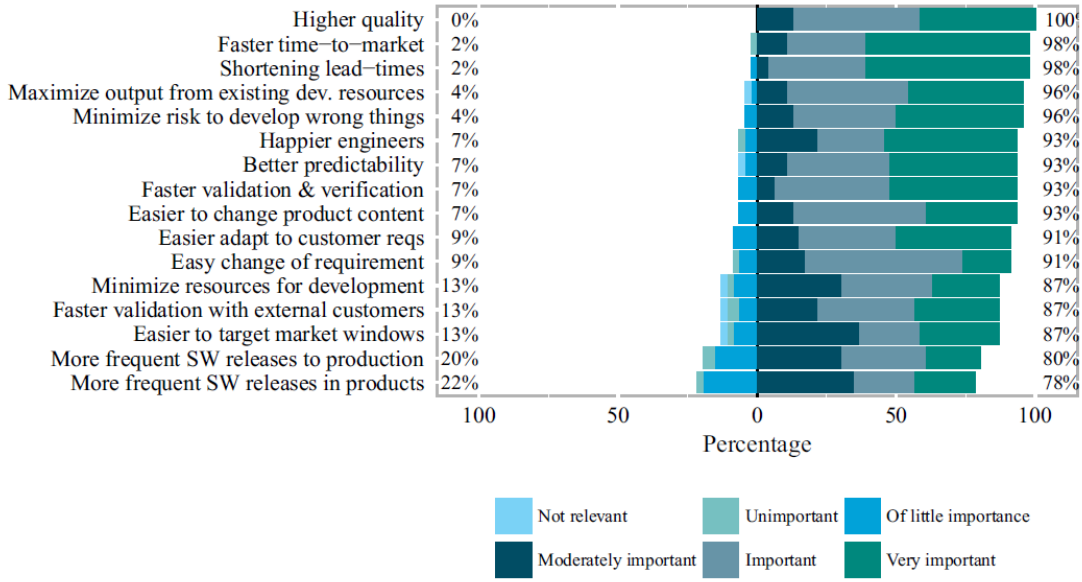

Figure 3.4: Expected benefits when scaling agile beyond software development teams (Eklund & Berger, 2017).

practices have been identified as unique to the mechatronics domain. From this list, a further refining permitted the authors to identify the final 16 practices that not only were recognized as uniquely correlated with the mechatronics domain, but were also verified as key agility enablers via a control set. The six companies, in fact, were subdivided in two groups: four were used as the main interview set, the remaining two as a control set. This means that the final 16 goals and practices, reported in Table 3.2 on page 82, were not only mentioned by at least one company among the first four, but also confirmed by at least one of the two companies from the control set.

In addition to the practices description, the table also reports the agile maturity level corresponding to each entry in the second column. The maturity level ranges from low to high: Collaborative, Evolutionary, Effective, Adaptive, and

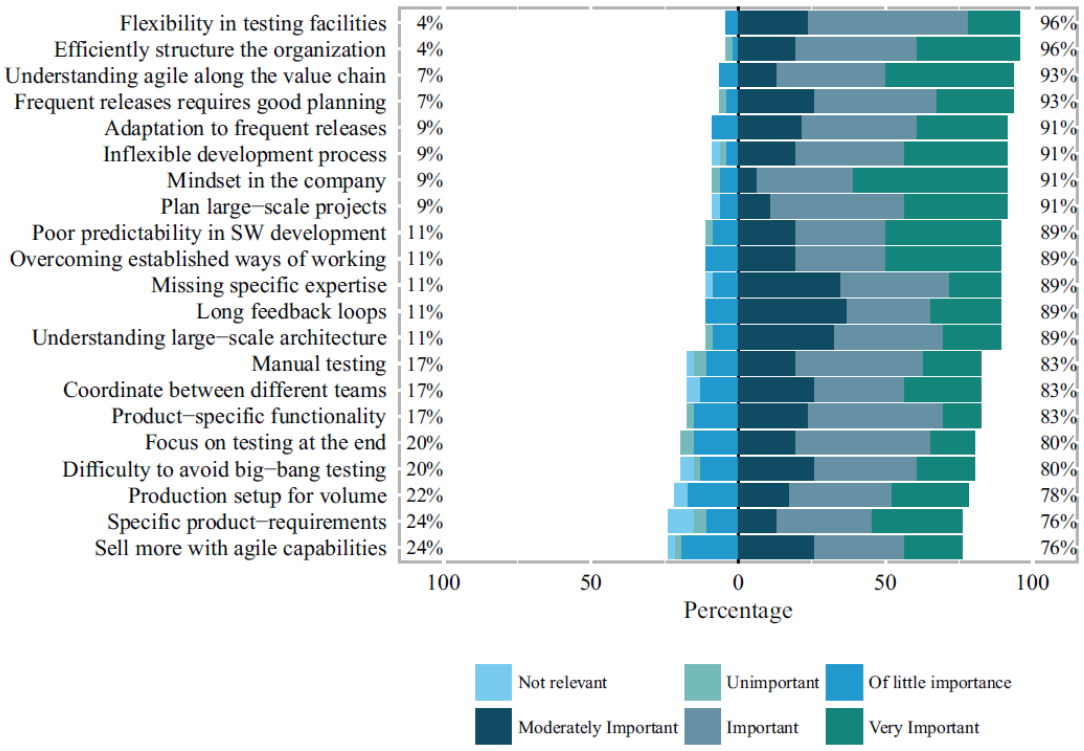

Figure 3.5: Foreseeable challenges when scaling agile beyond software development teams (Eklund & Berger, 2017).

Encompassing[1]. The idea behind such a scale is to provide information on the evolutionary path through the stages that can support an organization attempting to scale agile development. In other words, a company should focus on the items at the bottom of the table, and then aim at escalating towards the top, in order to successfully implement agile across multiple domains. These four levels also provide the rule for the subdivision of principles inside the table.

Finally, the third and last column sorts the 26 principles according to the Agile Manifesto's agile principle (Beck et al., 2001), already mentioned in Section 1.2.

[1]From Eklund and Berger (2017): "Collaboration is considered an essential agile value and is therefore the 1st level. The 2nd level is to develop software through an evolutionary approach. The 3rd level is to effectively and efficiently develop high quality software. The next level is using multiple levels of feedback to respond to change. The final 5th level is to achieve an all-encompassing environment to sustained agility".

Description	Maturity level	Agile principle
Minimize the number of point of contacts between SW, HW and mechanics	4 Adaptive	Technical Excellence
Reduce variant complexity (component level)	4 Adaptive	Technical Excellence
Allow for integrations of not the full product (e.g. Simulations)	4 Adaptive	Technical Excellence
Not using the same planning/project gates for HW and SW	4 Adaptive	Plan and Deliver Software Frequently
Reduce variant complexity (product level)	4 Adaptive	Plan and Deliver Software Frequently
Do not isolate disciplines	3 Effective	Human Centricity
Do not depend on manual deployment	3 Effective	Technical Excellence
Integration is a continuous activity (every 4 weeks)	3 Effective	Technical Excellence
Move towards platforms	3 Effective	Technical Excellence
Move complexity from mechanics to software/moves lead-time	3 Effective	Technical Excellence
Minimize supplier lead-times	2 Evolutionary	Human Centricity
Speedy deployment of test software to the (prototype) product	2 Evolutionary	Technical Excellence
Quick and dirty HW available to test SW functionality	1 Collaborative	Technical Excellence
SW available to use in tests of HW development	1 Collaborative	Technical Excellence
Multidisciplinary teams	1 Collaborative	Human Centricity
Having an agile process to adjust technical interfaces	1 Collaborative	Human Centricity

Table 3.2: Agile goals and practices particular to mechatronics development for scaling agile (Eklund & Berger, 2017).

Looking at the results gathered in Table 3.2, starting from the lowest agile maturity level, some observations can be made. At the *Collaborative* level (1), it is highlighted how the process must be as agile as the individual teams to achieve agility at scale. Teams should consist of people from multiple different disciplines. Not only they need to be cross-functional, meaning that they should come from different SW and HW domains, but they should also be able to cope with different tasks (e.g., both coding and testing). Moreover, it is important to have hardware ready as soon as it becomes available, even if not in its final shape, to test software functionality. Similarly, there must be relevant software available for the hardware testing.

Going on with the *Evolutionary* level (2), it is evident how, in case of outsourcing, it needs to be coupled with actions that minimize the suppliers' lead-times. Parallelly, attention should be put on providing organizational structures that do not isolate developers. Developers themselves should then focus on speedy development of test software for the prototypes.

At level (3), *Effective*, the table suggests to speed up the integration in prototype products (e.g., by means of automated software deployment at scale, instead of manual procedures). Full integration of software, hardware, and mechanics at least every four weeks should also be achieved. Unfortunately, this is one of the most complex goals for a mechatronics company. In fact, while more frequent integrations favor SW development, "it seems to have only marginal benefit for the other disciplines" (Eklund & Berger, 2017), that may need of longer iteration times, as suggested by the previously analyzed case studies in Section 3.2, where such a duration was set to 20 weeks. Continuing with the effective-level practices, one last suggestion is to move complexity from hardware to software if possible, by means of careful system design. In this sense, the use of platforms to develop multiple products would speed up the development and ease the addition of features.

The *Adaptive* level (4), presents the largest number of suggestions of the whole analysis. Among them, it is suggested to reduce variant complexity. In this sense Eklund and Berger (2017) suggest to pay attention to the number of variation points, if they become too many they risk introducing technical complexity that makes changes not agile at all. Another critical point is found during planning, and in particular when introducing product variants to be "carried over" to other projects: the risk is to reduce the flexibility in re-prioritization. Finally, variant complexity should be kept under control when deriving variants of existing products: if this process is not coupled with a parallel one, aiming at reducing the number of variants that R&D should maintain, the amount of work quickly grows.

As mentioned, however, there also exists a level (5), namely *Encompassing* maturity level. During the research, only one agile contribution emerged for this level, but it was not confirmed by the control set and, as a consequence, not included in 3.2.

Eklund and Berger (2017) also performed a comparison between the challenges that their work identified for large-scale agile transformation and a systematic literature review by Dikert et al. (2016). The results are reported in a schematic way in Table 3.3. It can be noted how challenges in agile transformations in different domains have more in common than what is different.

As a conclusion, Eklund and Berger (2017) work shown how a large portion of

Mechatronics challenges	Dikert et al. (2016)
Flexibility in testing facilities	*No equivalent*
Efficiently structure the organization	Internal silos kept
Understanding agile along the value chain	Misunderstanding agile concepts
Frequent releases requires good planning	Challenges in adjusting to incremental delivery pace
Adaptation to frequent releases	Challenges in adjusting to incremental delivery pace
Inflexible development process	Using old and new approaches side by side
Mindset in the company	General resistance to change
Plan large-scale projects	Challenges in adjusting product launch activities
Poor predictability in SW development	*No equivalent*
Overcoming established ways of working	Skepticism towards the new way of working
Missing specific expertise	Internal silos kept
Long feedback loops	Challenges in adjusting to incremental delivery pace
Understanding large-scale architecture	Achieving technical consistency

Table 3.3: Comparison of challenges for large-scale agile in mechatronics domain and pure software (Eklund & Berger, 2017).

the scientific research about scaling agile can be considered valid regardless of the application domain. On the one hand, this conclusion is encouraging for the agile transition of large scale mechatronics companies. On the other hand, however, it should also be kept in mind that "there is still no silver bullet in accomplishing this" (Eklund & Berger, 2017). This process needs the tuning of a large number of practices interacting one with the other, in order to declare the agile transition successful.

3.4 Case 4: Agile development of luxury bathtubs

One of the earliest case studies describing agile practices applied by real manufacturing companies in their routines comes from 2012 and analyzes Teuco-Guzzini, an Italian company producing luxury hydromassage bathtubs and showers (Mazzanti, 2012).

3.4.1 The background

In this case, the issues that the company management wanted to tackle were the too long time to market, and the too low quality level.

Before implementing agile principles, the author performed an assessment to understand the state of the art at his arrival. The tools used are simple interviews with both managers and engineers, Draw The Process[2] with engineers only, and Premortem Restrospective[3] with managers only. The interviews' results highlighted the following aspects (Mazzanti, 2012):

- General lack of transparency and effective communication.
- Focus on local efficiency, with each department having its own independent goals.
- Constantly changing requirements and priorities.
- Too long concept phase for new products, creating incomplete or vague output.
- Overburdened and somewhat demotivated engineers.
- Corporate process and procedures, considered too complex and cumbersome, were often ignored or cheated.
- Engineers had no specific skills in managing projects and external partners.
- No interaction between engineers and customers.

The other two activities provided few new considerations and some confirmations on what already emerged. Draw The Process indicated the most critical steps to be the feasibility phase; the late product modifications, due to continuous specification changes; and the product concept phase, perceived as too long and not providing clear guidance. Premortem Retrospective, instead, identified some key information such as the fact that overburden, multitasking, and continuous changes in priorities were the norm; specifications changed too often and too late; concepts approval

[2]Draw The Process is "an activity in which groups of 6-8 people are asked to draw the process that brings a product from concept to shipment and then identify the most critical and problematic steps. [...] The goal of this activity is to check if there is a shared understanding of the process and to get a first feeling of where the major issues are" (Mazzanti, 2012).

[3]The idea behind Premortem Retrospective is to "fast-forward ahead in time, typically 6-12 months, and presume that a project has miserably failed. Key events that 'happened' during the project are positioned along a timeline. Events are then grouped and discussed, and finally dot-voted to identify the most important ones" (Mazzanti, 2012).

took too long; optimistic planning lead people to stay late due to schedule slips; and production costs were often higher than expected (Mazzanti, 2012).

Putting together the evidence just discussed, the author was able to identify the major issues, as well as those issues that were perceived in a completely different manner by managers and employees. As an example, overburden was one of the firstly mentioned items from employees, but it was not even noted by managers.

3.4.2 Agile practices introduction

In this case study, the author also acted as an external agile coach. Differently to the other papers analyzed in this chapter, however, the author implemented both agile and lean manufacturing processes/practices. Even tough the focus of this thesis is about agile, the early nature of this research paper made it interesting to analyze a mixed approach like this as well.

Mazzanti (2012)'s first step was indeed a long learning activity, necessary to teach both lean and agile to the engineers working at Teuco. Initially this caused misunderstanding and annoyance among the employees, but the author considered such an activity as essential to the following steps. Among other topics, great attention was put on the importance of interactions, communication, and self-organization. Agile methods were used a lot to handle interactions and conflicts. During this step, it was difficult to drop some old habits, like the tendency to multitasking in order to counteract the overload of work.

The following step was then to map the old development process, in order to prove to the team why it was so important to learn new practices and revolutionize their habits. Mazzanti (2012) mapped Teuco's process by identifying three major phases: concept, development/prototyping, and production. Each phase had three sub-phases. In such a process, the product features, costs, and market shipment date were set during the concept phase, where uncertainties were too high. This exposed the company to expensive late changes in requirements. Moreover, the average delay in completion time was measured to range from 2 months for small projects to 6-9 months for larger projects, corresponding to a relative 33%-50% delay in relation to the projects' length. Other issues concerned the role of project

managers who had to manage too many projects at once, ignored deadlines, and difficult to predict demand.

Despite the evidence, it was difficult for the author to make employees and managers realize the real situation. For this purpose, a physical representation over a board was created. Here, each column represented the process stages from the late concept phase onwards, and each card represented a project. As visible in Figure 3.6, most of the projects were stuck at the beginning of the process because of delays, overloading, bureaucracy, and other reasons. The simple fact of exposing the problems triggered the management to immediately act. This fact shows the power of such a simple tool.

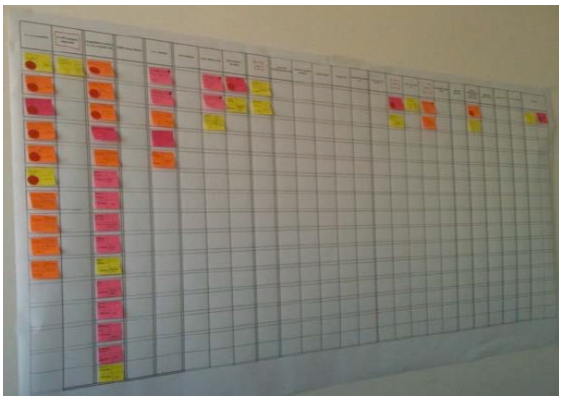

Figure 3.6: The initial process mapping from Mazzanti (2012), showing overcrowding in the first columns

3.4.3 Results obtained

At this point, some typical agile practices were introduced: the author described them as daily stand-ups in front of the board, and weekly retrospectives, resembling very much the idea of daily Scrums and Scrum reviews. Again, engineers found these activities useless and boring, so the approach changed, in order to focus only on highlighted problems rather than providing a full overview to the management about ongoing projects. After some adjustments, meetings became shorter and more useful, identifying hidden and forgotten issues. They also provided practical suggestions to remove time-consuming corporate procedures that were taking several

days for projects approval, and increased teams cohesion. The author, in facts, described engineers as more individualistic people with respect to software engineers, according to his experience. With the aid of Kanban, Portfolio Management, and lean procedures, other issues were tackled jointly: visibility for overloading situations increased, a new priority structure was suggested, and project ownership changes were better exposed during the project life.

Concluding, after one year of agile implementation over the product development process, the author noticed a significantly reduced overburden of engineers. These results have been achieved thanks to demand management and capacity leveling. In the words of the author, "assigning priorities based on due date and cost of delay" and "adjusting activities and work to available capacity" (Mazzanti, 2012). A consequential result was better predictability, both in terms of schedule and quality. Additionally, cycle times have dropped consistently and according to the projections made during the paper writing, time to market was reduced by 30% on average, corresponding to 2-4 months earlier entrance. Finally, engineers seemed more engaged and motivated.

Focusing on agile applications to product development, the author highlighted how stand-ups reviews and retrospectives induced "engagement, commitment, transparency, self organization, trust, and empowerment in the teams" (Mazzanti, 2012). Being an agile coach, the author stated that, at the beginning, the absence of common SW practices caused him some troubles, and he felt a bit lost. His suggestion in this sense is to "focus on the flow of activities and on how these activities are producing value for the company's customers". He also identified similarities between design reviews in SW development and in HW design. For what concerns the people involved, he found engineers to be a group of individuals rather than a team. However, engineers seemed to have a better understanding of the whole product cycle, compared to SW developers. Other differences were that engineers tended to wait for their coach's solution, and did not show much interest in the technical solutions possibly available and implementable. Nevertheless, it has to be kept in mind that these considerations are strongly affected by the author's background, and the company's status when he intervened. Additionally, the small dimensions of the company, its geographical location, and the fact that this research

has been performed 10 years ago could also alter these conclusions, especially the ones related to the human resources differences between HW and SW development.

Finally, it is interesting to note how, after noticing the positive impact of this approach on product development processes, other departments at Teuco demanded to try similar approaches as well. Despite the short timeframe, that did not give enough time to the author to state them as successful, the adaptation of such an approach over sales and marketing was providing surprisingly good results.

3.5 Case 5: Product Development of Medical Devices

Gerber et al. (2019) focused their research on a particular sector of the manufacturing industry, namely the medical devices product development. In their study it is firstly pointed out how this discipline could benefit from the introduction of additive manufacturing processes, data analysis, and virtual reality, but would still need to face important limitations and strict regulations on procedures and documentation. This means that, as an example, fast prototyping would be allowed, but a no-document process will never be possible in the medical devices area.

3.5.1 Research performed and results obtained

With these premises, the team tracked the development of a microtiter plate, trying to adapt Scrum to the conditions of the product development of such a device. The goal was to reach an agile, more specific, and better suitable method for physical products. In this case study, as in the previous ones, two research questions were elaborated:

RQ-1: How do agile methodologies influence the team climate in student research projects?

RQ-2: Which adaptions and changes to agile methods like Scrum support effectiveness and a comfortable working environment?

For this project, a team made of seven members, responsible for design, simulation, requirements management, and agile procedures, worked six moths at the product development. Thanks to the combination of classic and agile requirements' management, and thanks to the adaptation of the manufacturing processes to additive manufacturing, they were able to give birth to a final product, while taking medical guidelines into account. As a result, a 3D-printed prototype was created, as reported in Figure 3.7. During the process, a new geometry to enable more cell-friendly microfluidics – required by the product's specifications – was implemented as well, after various simulation runs.

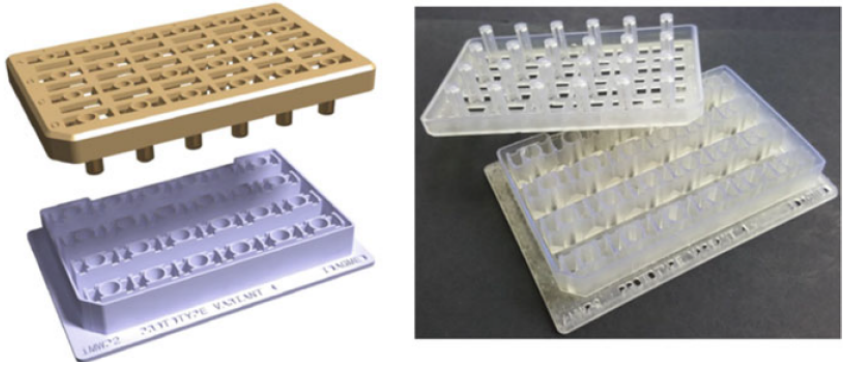

Figure 3.7: CAD model, on the left, and 3D printed microtiter plate, on the right (Gerber et al., 2019).

Speaking of the process itself, Scrum was kept as the method at its basis, but some modifications made are worth of mention. Daily scrums were initially implemented via Slack™, an online communication tool, rather than in person. Soon after the project started, however, they moved into video chats or on-site meetings, that proved much more efficient. Retrospectives were held with a similar structure to the original Scrum methodology ones, and provided suggestions for future improvement potentials. Among the tools used, the authors listed: user story mapping (see Figure 3.8a), Starfish retrospective (see Figure 3.8b), timeboxing, feedback pitches, and joint visualization of the share emotional perception (see Figure 3.8c).

The results of the project were evaluated by making use of the team climate inventory (TCI) (Anderson & West, 1998) – a structured self-report measure to

Figure 3.8: Sprint impressions (Gerber et al., 2019).

assess the climate for innovation within groups – and a feedback log. Firstly, it was noted how the distribution of tasks at the beginning of the project was not clear. Explicit rules turned out to be needed, in order to control the Scrum process. Similarly, also the formulation and maintaining of user stories was found to be difficult. In this case, the solution adopted was to adapt the user stories to physical product development. Continuing, the team was skeptical about the possibility to produce complex medical products within one sprint. This feeling changed throughout the project, thanks to the periodic retrospectives that boosted the team's expectations for its own results. This trend was accompanied by an increasing need of physical presence during working hours. This lead to the a first answer for RQ-1, and provides an interesting observation on how agile product development, also for HW products, benefits from a physical presence and suffers the use of remote working. In the previous chapters, it was questioned how the modern trends and technologies, as well as a the different dynamics involved in HW development, could influence the original agile mantra according to which teams should work together in a single room. This survey seems to provide a preliminary answer to this kind of doubts. Going back to RQ-1, the authors got to the following conclusion:

> Agile methods accelerate the team development process at the beginning, but the additional effort due to the new process rules is most profitable after about 3 sprints. Afterwards the well-rehearsed team can benefit from the known scrum standards and developed practices (Gerber et al., 2019).

3.5.2 Practical suggestions

From these observations, Gerber et al. (2019) elaborated a list of suggestions for future works, providing an answer to RQ-2. These thoughts have particular importance in the agile development of medical technology:

- Shorter sprints and a division into pre-phase, iteration-phase, and final phase are recommended.
- Research, interviews, and joint workshops on the topic would be urgently needed before the actual development. This would allow consensus to be built on the goals and vision of the project, technical knowledge, boundary conditions of the product, a common picture of the end-user of the product, and primarily an understanding of agile methods.
- The approaches of Design Thinking could be used to align the project goals and vision.
- The iteration phase should have the structure of a sprint/Scrum. Innovative solutions for partial problems can then be systematically developed.
- To select the documentation and technique level during a sprint, the sprint missions should be used.
- In order to keep an eye on both technical requirements and user needs, personas should be used. Their use allows evaluating the work results from the preliminary phase.
- It would make sense to coordinate the final phase independently of the regulations and sprints. This would make possible to adapt the documentation to the requirements of a previously untreated risk and quality management system, and thus meet the most important requirements of medical technology product development.
- The length of the final phase should not exceed one to two sprint lengths.

The above suggestions refer to the process identified by the authors as a possible solution for agile product development of physical products, reported in Figure 3.9. The process is not analyzed in detail in this section, as its further evolution will be discussed in Section 3.6.

The project results were overall satisfying. Prototypes were printed with additive methods and simulation models. Sprints lasted four weeks, and enabled a sort of product development process that very much resembles the one encountered in the

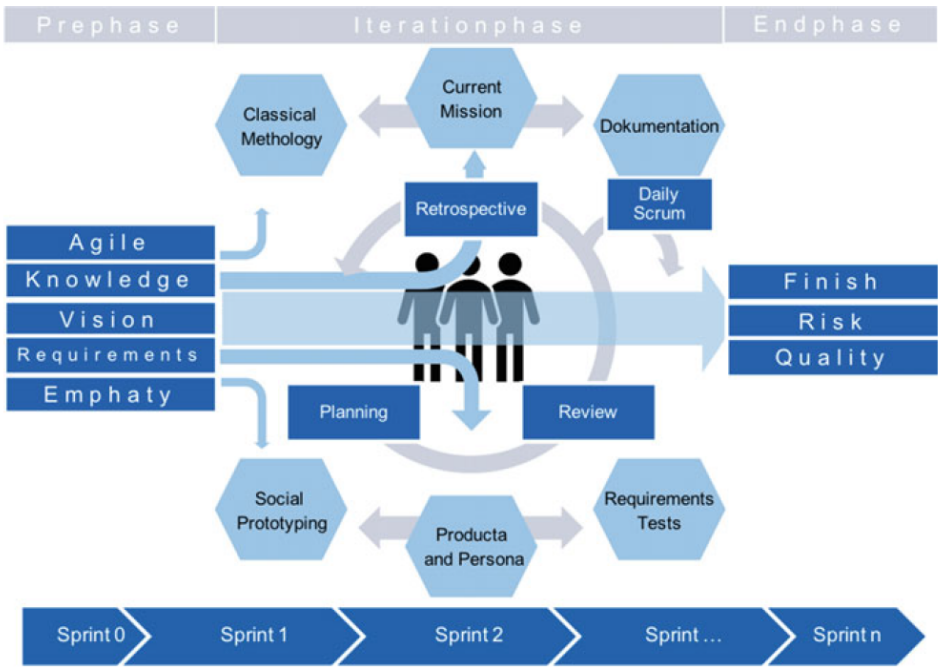

Figure 3.9: Recommendations of an agile process for physical product development (Gerber et al., 2019).

SW domain, with frequent adaptations due to changing customer requirements. The following section will analyze a second case study reporting further development of this methodology.

3.6 Case 6: Agile Development of a microtiter plate in an interdisciplinary project team

After formalizing the previously illustrated methodology, part of the research team that contributed to the previous case study attempted to evaluate and improve such a method, by means of a second real life application study. Kristin Goevert, Sebastian Schweigert-Recksiek, and Udo Lindemann were in fact signatories of the previous article. With the aid of Bidal Tariq and Lukas Krischer they were able to perform the study analyzed in this section (Goevert et al., 2019).

3.6.1 Research performed

During the project, the five researchers cooperated for six months. On the product side, the goal was to create a biocompatible additively manufactured microtiter plate. On the process side, as a second goal parallelly pursued, the researchers looked for a new agile medical product development process. During the whole project, the collaboration of the team and other stakeholders was analyzed as well, in order to suggest improvements to the efficiency of the cooperation itself.

The team made use of the same tools previously cited, such as scrum sprints, for which a duration of three weeks was found ideal, user stories, daily meetings, retrospectives, and weekly meetings. The external stakeholders, such as the project partner that hereby plays the role of the customer, were also involved. Self-organization and transparency were fostered in order to, respectively, motivate the team members and speed up the decision process. With respect to Gerber et al. (2019), some changes were applied. For example, the task board was digitalized, and a chat for team members was implemented. Other product development methods were used, such as benchmarking, risk matrix, kano-model, and weak-point-analysis. The team also implemented a matrix to show dependencies between user stories and requirements, as well as to show the status of fulfillment of the different user stories. With respect to agile SW development, this method was adapted to permit a better supplier's management: "as certain user stories were highly dependent on suppliers, risk management had to be integrated into the agile framework (Goevert et al., 2019)". As a consequence, the team focused and reassessed risk on a regular basis, taking precautionary actions accordingly.

The first research question drafted by the authors is described as a consequence of Gerber et al. (2019) as well. The second one, instead, comes from the first challenge that the team had to face: changing the product development process from injection molding to an additive manufacturing process.

RQ-1 : How does the agile process have to be built so that the tasks, goals, and requirements of medical device product development can be integrated?

RQ-2 : How can a biocompatible microtiter plate with different geometries be built with an additive manufacturing approach

3.6.2 Results obtained

Collaboration within the team

For the collaboration assessment, the team found out that many barriers were present between the team members and the other parties, leading to a number of unfinished tasks and user stories. Specific actions were taken for each user story, in order to better monitor them. As a result, the progress of the project could be planned more accurately during the scrum meetings, and the final product could be delivered on time.

Process-related results

Speaking of the practical structure of the process, four phases were set up, as illustrated in Figure 3.10. Black symbols represent the products, gray arrows the events, blue arrows the processes, and blue figures the roles. As schematized, the

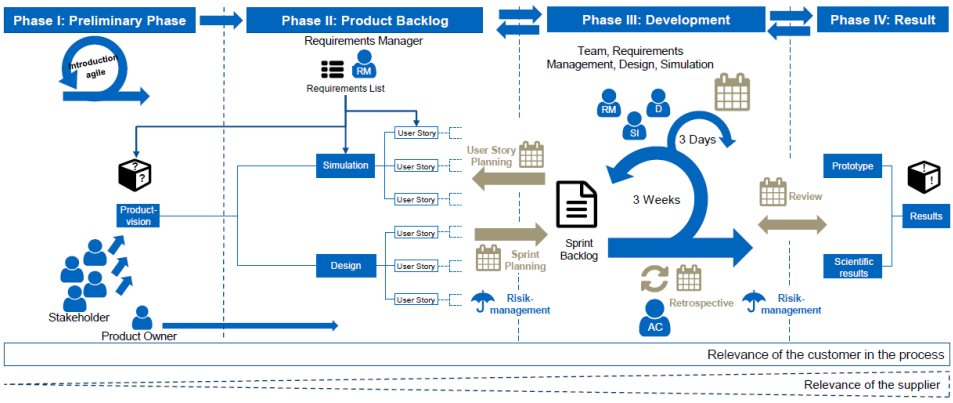

Figure 3.10: Agile model for physical and medical product development (Goevert et al., 2019).

process is made of four phases. During the preliminary one, the vision of the product and its project are defined with the stakeholders. This information is then transferred by the product owner onto the second phase.

Here, the product owner is responsible for the product backlog and its refinement, including the pre-definition of user stories, with the support of the other team

members. In this case, the team opted for a subdivision of user stories in two fields: simulation, and design. In this way, experts of each field could collaborate. Finally, during the product backlog phase, a continuous product requirement integration was present at all levels. In this specific case, product requirements played an important role due to limitations and guidelines that rule the medical products development, but in general any physical product development could benefit from this addition. To successfully meet product requirements, it was necessary to have constant feedback from a requirements' manager, as well as getting proper combination of user stories and requirements, by means of the previously mentioned matrix.

Going on, the third phase begins with the sprint planning event. During the development phase, the elements of the sprint backlog are created within the three weeks sprint. A sort of daily scrum is held at least every three days, if not more often, and the requirements' manager supervises the whole process. At the end of each sprint, the results are discussed with the relevant stakeholders, and everything is brought back to the product backlog, with the creation of new user stories. During the final stages of this phase, the risk manager establishes the transition towards the final phase, protecting the team by external influences, and avoiding that such influences interrupt the development of some user stories.

Finally, in the last phase prototypes are created, providing added value for the customer, so that he/she can provide early feedback. Additionally, the presence of prototypes allows for early testing.

Concluding, for what concerns the development process just described, this case study proved to be successful and provided interesting insights. The authors however noted how during the first weeks of the project, the team was fulfilling more user stories with respect to the end, where planning was milder. In this sense, teams should keep in mind that the plan is only a rough estimation, and the higher importance should be given to the product: "a product is more important than following a plan" (Goevert et al., 2019). In other words, also during physical product development, teams should switch towards a more agile mindset.

Product-related results

For what concerns the project in analysis, the team was able to transform the injection molding design into a design for 3D printing via additive manufacturing processes. This affected the product geometry in particular, giving birth to a new patented shape with rectangular chambers instead of the classical round ones. As a consequence, new tests and examinations were needed, to verify the fulfillment of the product requirements. Here, CFD simulation models were used, allowing iterative modifications to the various prototypes, until the final one.

Among the authors conclusions, it is worth to mention that "the produced prototypes in combination with the simulation present a value for the product owners and project partner", and "the integration of product development methods supports the team, as well as in structuring the agile process" (Goevert et al., 2019).

Going back to the two research questions, they were both answered by the study. The first answer comes from the proposed agile methodology described above, and the second one by the production of the final prototype.

Conclusions and discussion

This research started with a literature review on both agile software development methods, and their possible application in the hardware world. After having discussed the origins of agile, the purpose for which the agile manifesto has been published, and the first practical applications of agile methods, the analyses moved towards the possibilities that such methods could have in the development of Cyber-Physical Systems, in managing multidisciplinary teams, or in revolutionizing the HW development processes.

In a certain way, all these fields of applications have been touched by the case studies analyzed. Case studies number 1 and 2 (Eliasson et al., 2014) have been useful to understand how agile product development already is a mature technology, being applied in an important corporation like Volvo Car Group. Moreover, the fact that VCG implements agile methods on new and experimental projects confirms their capability of managing uncertainty and highly innovative projects. This paper also attested how the automotive industry can be one of the ideal field of application for agile methods, given the always increasing amount of electronics involved in car manufacturing. Finally, modern technologies proved once again to be great agility enablers. In this case the researchers made use of Virtual Prototyping, Rapid Prototyping, virtual testing, and plant models, applying Model Based Engineering principles in parallel with the agile ones.

Case study number 3 provided, instead, a different kind of contributions to this document. While the previous two studies described with detail how an important corporation applies such innovative methods inside its organization, Eklund and Berger (2017) elaborated a set of suggestions that should help the

application of agile methods in any mechatronic project. The results can be hardly summarized, and should be considered in their entirety, as reported in Table 3.2. This table suggests a total of 16 best practices to follow, in a hierarchical order over 5 different maturity levels. Ideally, any company operating in mechatronics product development could benefit from the application of such practices starting from the basis and reaching the top. As an example, one could find useful the implementation of multidisciplinary teams (level 1) at the beginning of its transition, while more mature companies could examine with more detail the implications that a high product variety has on their operations.

Case 4 brought again a completely different point of view, being the first to analyze an HW-only product being developed by means of agile methods. The main contributions of this document regard the methods applied. Apart from the presence of some lean manufacturing techniques, the paper demonstrated how Scrum might be the best solution for physical products development, among the existing agile methods described in Section 1.3. As already suggested in Chapter 2, Scrum also proved to need a lot of tailoring and its direct application on HW projects was found impossible once again.

Finally, cases 5 and 6 acted as a conclusion, providing a mix of different results. They represent another example of how agile can be useful also in case of HW-only products, as for case number 4. At the same time, they made larger use of modern technological tools such as additive manufacturing, CFD simulations and their consequential testing. While case number 3 focused more on the organizational side and the implications of Scrum, Gerber et al. (2019) and Goevert et al. (2019) also experimented the efficiency of such technologies as agility enablers. Continuing, these two case studies suggested how agile methods are ideal for innovative solutions in highly restricted field of operations, such as the medical devices one. In addition to these observations, the two case studies also suggested a complex and complete agile product development method, tailored to the medical devices development needs.

To provide a more complete overview of the previous chapter results, in the following Tables 3.4 and 3.5 a set of advantages and limitations of agile in the

development of non software products is present. Both pro's and con's are subdivided by paper and by field: technical/operational, organizational, and project management.

Case #	Product	Technical/Operational	Organizational	Project Management
1	Safety clutch for electric vehicles' engines	Virtual/Rapid Prototyping and testing allow to simulate the behaviour that HW and SW elements will have at the integration point, reducing the knowledge gap present in traditional approaches by diminishing the number of assumptions needed at product specifications level. This decreases the risk of faulty assumptions, potentially causing time and money losses.	Dependency on mechanical parts during the development can be relaxed thanks to the agile implementation and the use of testing.	Simulations allow going on without needing managerial approval at every step, and ease communication within the teams.
		Fewer assumptions decrease the risk of faulty ones, potentially causing time and money losses.		
2	Active high beam headlight	Quick demonstration tools allow for a larger number of testing.	Agile can be coupled with a MDE approach successfully: plant models can be used as prototypes.	
		Virtual testing environment allows for flexible tests and rapid adaptation of both the HW plant models and SW code in case of faulty assumptions.		
3	Mechatronics systems	The case study focused on providing practical suggestions rather then exposing benefits and limitations of agile.		
4	Luxury bathtubs	Scrum allows better quality and schedule predictability also in HW project.	Scrum can reduce engineers overburden by managing demand better and leveling capacity.	Stand-up reviews and retrospectives can induce engagement, commitment, transparency, self organization, trust, and empowerment in the teams.
		Scrum can lower cycle times in HW products development.		
5	Microtiter plate	Scrum can be applied in HW product development projects with sprints of four weeks, allowing for frequent adaptations in case of changing customer requirements	Research, interviews, and joint workshops held before the development build consensus on the goals and vision of the project, technical knowledge, boundary conditions, a common picture of the end-use, and an understanding of agile methods.	Periodic retrospectives can boost the team's expectation, making members more confident.
		Prototyping and simulations allow for faster development and product modifications along the way.		Agile methods accelerate the team development process at the beginning, but the profitability peak is reached after about 3 sprints.
		Additive methods ease prototyping and, as a consequence, agile implementation.		Applying agile methods, Personas allow to track both technical requirements and user needs.
6	Microtiter plate	Additive manufacturing, CFD simulation models, and prototyping allow for faster development, and can also lead to completely new design solutions, as a consequence of the adaptation of the product to be compatible with it.	The constant presence of all stakeholders permits early feedback from the customer, crucial to meet the product requirements.	The presence of a requirements' manager troughout the project is useful to ensure the fulfillment of product requirements, and get proper combination of user stories and requirements.
		Faster developing allows for faster testing, useful to get to the following prototype quickly.		

Table 3.4: Advantages of Agile in the case studies analyzed

Case #	**Product**	**Technical/Operational**	Organizational	Project Management
1	Safety clutch for electric vehicles' engines	Despite the use of virtual environments for testing, some faulty assumptions can always arise. It is impossible to foresee every possible outcome with precision.	Virtual testing allows HW and SW developers to work separately, but this can cause issues related to the knowledge gap between them: having little knowledge of what the others do leads to simplifications and inconsistencies.	Poor laboratory tests' settings augment the risk of faulty assumptions.
2	Active high beam headlight	In case of faulty assumptions, HW problems may need much more time to fix with respect to SW ones, and can also affect the product design.	Virtual testing and assumptions on the components' behavior can detach researchers from reality, creating unnecessarily complex solutions.	
3	Mechatronics systems	The case study focused on providing practical suggestions rather then exposing benefits and limitations of agile.		
4	Luxury bathtubs		Engineers may tend to wait for their coaches to solve the problem when applying agile: Scrum masters should enforce a cooperation mindset, making the team follow a shared goal.	For agile coaches, the absence of SW in some projects can be a challenge. In these cases it is useful to focus on the flow of activities and on how these activities are producing value for the company's customers.
5 & 6	Microtiter plate			User stories may be difficult to maintain for HW product development: they need to be tailored on the development process of a physical object.

Table 3.5: Limitations of Agile in the case studies analyzed

Comparing these results with what has been achieved in Chapter 2 – and summarized in Tables 2.8, 2.9, and 2.10 – some points have been confirmed, while others did not find much evidence. In general, prototyping (as well as testing and simulations) was proved to be a key element in the agile transition. In both chapters, the need for faster and more frequent product improvements has been often mentioned. As a consequence, also the use of simulation softwares, CAx tools, Rapid prototyping, and Virtual prototyping technologies has been confirmed as part of the agile product development methodology.

Similarly, in the case studies analyzed in Chapter 3, the centrality of the customers requirements was confirmed. However, some differences in this sense are present as well. For physical products, in fact, it is essential to *get the product right*, according to specifications and demands, but often the customer is harder to engage, and sometimes completely absent. In this sense, the concept of "customer" could be enlarged to include any stakeholder that can affect the product requirements. For medical devices, as an example, regulations and certifications play a very similar role. Concerning its engagement, instead, this very much depends on the possibility to include him/her in the company processes. For big corporations, this might still be a challenge. Parallelly, a second obstacle is represented by the ability of developers to release prototypes and new versions often. In the analyzed papers this varied a lot, depending on the approach chosen by the team, but also on the product. In this sense, it should be kept in mind how physical products entail bigger differences among them, and a standard agile practice for all of them is hardly possible to implement.

For what concerns methods, the literature analysis performed in Chapter 2 was open to a variety of possibilities, including, as an example, Agile – Stage/Gate hybrids. Such possibilities were not verified by any case study, since almost everyone made use of Scrum or versions of it. Scrum in fact appears to be the best possible solution in case of HW products, provided that a fine-tuning and tailoring process is applied to it, using the product characteristics and customer expectations as the drivers for its adaptation.

Concluding, agile product development outside software development was proved to be successful by several applications, both at academic and industrial level.

Modern technologies are essential to support the diffusion of such an approach, making HW development closer and closer to SW development. Differently from the software domain, however, it is difficult to set up standard procedures and methods. Since physical products differ a lot, standardization is possible only within very tight borders: as an example, it is possible to suggest an agile method for the production of medical devices, but this will not necessarily be applicable in the automotive industry.

Bibliography

Altavilla, S., Montagna, F., Newnes, L., et al. (2017). Interdisciplinary life cycle data analysis within a knowledge-based system for product cost estimation. *DS 87-5 Proceedings of the 21st International Conference on Engineering Design (ICED 17) Vol 5: Design for X, Design to X, Vancouver, Canada, 21-25.08. 2017*, 375–384.

Ambler, S. (2002a). *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons.

Ambler, S. (2002b). Introduction to agile modeling (AM). *2002b*.

Anderson, N. R., & West, M. A. (1998). Measuring climate for work group innovation: development and validation of the team climate inventory. *Journal of Organizational Behavior: The International Journal of Industrial, Occupational and Organizational Psychology and Behavior*, *19*(3), 235–258.

Beck, K. (2000). *extreme programming eXplained: embrace change*. Addison-Wesley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for Agile Software Development*. http://www.agilemanifesto.org/

Böhmer, A. I., Hostettler, R., Richter, C., Lindemann, U., Conradt, J., Knoll, A., et al. (2017). Towards agile product development-the role of prototyping. *Proceedings of the 21st International Conference on Engineering Design (ICED 17) Vol 4*.

Cantamessa, M., Montagna, F., Altavilla, S., & Casagrande-Seretti, A. (2020). Data-driven design: the new challenges of digitalization on product design and development. *Design Science*, *6*, e27.

Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. *Advances in Computer*, *62*(03), 1–66.

Cohn, M. (2010). Succeeding with agile: software development using scrum.

Cooke, A., Bonnema, G., & Poelman, W. (2012). Agile development for a multi-disciplinary bicycle stability test bench. In R. Scheidl & B. Jakoby (Eds.), *Proceedings 13th Mechatronics Forum International Conference, MECHATRONICS 2012* (pp. 812–819). Trauner Verlag.

Cooper, R. G. (2016). Agile–Stage-Gate Hybrids. *Research-Technology Management*, *59*(1), 21–29.

Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, *119*, 87–108.

Eklund, U., & Berger, C. (2017). Scaling agile development in mechatronic organizations-a comparative case study. *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 173–182.

Eliasson, U., Heldal, R., Lantz, J., & Berger, C. (2014). Agile model-driven engineering in mechatronic systems-an industrial case study. *International Conference on Model Driven Engineering Languages and Systems*, 433–449.

Enkler, H.-G., & Sporleder, L. (2019). Agile Product Development—coupling explorative and established CAx methods in Early Stages of Virtual Product Development [29th CIRP Design Conference 2019, 08-10 May 2019, Póvoa de Varzim, Portgal]. *Procedia CIRP*, *84*, 848–853.

Gerber, C., Goevert, K., Schweigert-Recksiek, S., & Lindemann, U. (2019). Agile development of physical products—A case study of medical device product development. *Research into Design for a Connected World* (pp. 823–834). Springer.

Gilmore, J. H., Pine, B. J. et al. (1997). The four faces of mass customization. *Harvard business review*, *75*(1), 91–102.

Glass, R. L. (2001). Agile versus traditional: Make love, not war! *Cutter IT Journal*, *14*(12), 12–18.

Goevert, K., Schweigert-Recksiek, S., Tariq, B., Krischer, L., & Lindemann, U. (2019). Agile Development of a Microtiter Plate in an Interdisciplinary Project Team. *Proceedings of the Design Society: International Conference on Engineering Design*, *1*(1), 2139–2148.

Highsmith, J., & Cockburn, A. (2001). Agile Software Development: The Business of Innovation. *Computer*, *34*, 120–122.

Highsmith, J., Orr, K., & Cockburn, A. (2000). Extreme Programming. e-Business Application Delivery.

Horvath, I., & Gerritsen, B. (2012). Cyber-Physical Systems: concepts, technologies and implementation principles.

Ismail, H., Reid, I., Mooney, J., Poolton, J., & Arokiam, I. (2007). How Small and Medium Enterprises Effectively Participate in the Mass Customization Game. *IEEE Transactions on Engineering Management*, *54*(1), 86–97.

Jackson, M. B., & Institute, P. M. (2012). Agile : A Decade In. *26*(4), 58–62.

Kaisti, M., Mujunen, T., Mäkilä, T., Rantala, V., & Lehtonen, T. (2014). Agile principles in the embedded system development. *International Conference on Agile Software Development*, 16–31.

Könnölä, K., Suomi, S., Mäkilä, T., Jokela, T., Rantala, V., & Lehtonen, T. (2016). Agile methods in embedded system development: Multiple-case study of three industrial cases. *Journal of Systems and Software*, *118*, 134–150.

Larman, C. (2004). *Agile and iterative development: a manager's guide*. Addison-Wesley Professional.

Mabrouk, A., Penas, O., Plateaux, R., Barkallah, M., Choley, J.-Y., & Akrout, A. (2018). Integration of agility in a MBSE methodology for multidisciplinary systems design. *2018 IEEE International Systems Engineering Symposium (ISSE)*, 1–5.

Mazzanti, G. (2012). Agile in the bathtub: Developing and producing bathtubs the agile way. *2012 Agile Conference*, 197–203.

Meyer, B. (2014). Agile. *The good, the hype and the ugly*, *1*.

Mulder, F., Verlinden, J., & Maruyama, T. (2014). Adapting scrum development method for the development of cyber-physical systems.

Oestereich, B., & Weiss, C. (2008). Agiles Projektmanagement: erfolgreiches timeboxing für IT-Projekte. *Aufl., Dpunkt-Verl*.

Poppendieck, M. (2001). Lean Programming. http://www.leanessays.com/2010/11/lean-programming.html

Reagan, B. (2012). Going Agile with Ca Clarity PPM & Agile Vision. *Going Agile with Ca Clarity PPM & Agile Vision*. https://www.slideshare.net/DCsteve/going-agile-with-ca-clarity-ppm-agile-vision

Reagan, J., & Singh, M. (2021). *Management 4.0: Cases and Methods for the 4th Industrial Revolution*. Springer Singapore. https://books.google.it/books?id=szyQzgEACAAJ

Riesener, M., Rebentisch, E., Doelle, C., Kuhn, M., & Brockmann, S. (2019). Methodology for the Design of Agile Product Development Networks [29th CIRP Design Conference 2019, 08-10 May 2019, Póvoa de Varzim, Portgal]. *Procedia CIRP*, *84*, 1029–1034.

Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. *Proceedings of the 9th international conference on Software Engineering*, 328–338.

Schwaber, K. (1996). Controlled chaos: Living on the edge. *American Programmer*, *9*, 10–16.

Schwaber, K. (2007). *The enterprise and Scrum*. Microsoft press.

Schwaber, K. (1997). SCRUM Development Process. In J. Sutherland, C. Casanave, J. Miller, P. Patel, & G. Hollowell (Eds.), *Business Object Design and Implementation* (pp. 117–134).

Schwaber, K., & Sutherland, J. (2012). *Software in 30 days: how agile managers beat the odds, delight their customers, and leave competitors in the dust*. John Wiley & Sons.

Sommer, A. F., Dukovska-Popovska, I., & Steger-Jensen, K. (2014). Agile product development governance–on governing the emerging scrum/stage-gate hybrids. *IFIP International Conference on Advances in Production Management Systems*, 184–191.

Sommer, A. F., Hedegaard, C., Dukovska-Popovska, I., & Steger-Jensen, K. (2015). Improved Product Development Performance through Agile/Stage-Gate Hybrids: The Next-Generation Stage-Gate Process? *Research-Technology Management*, *58*(1), 34–45.

Stare, A. (2014). Agile Project Management in Product Development Projects. *Procedia - Social and Behavioral Sciences*, *119*, 295–304.

Stelzmann, E. (2011). Contextualizing Agile Systems Engineering. *Aerospace and Electronic Systems Magazine, IEEE*, *27*.

Takeuchi, H., & Nonaka, I. (1986). The new new product development game. *Journal of Product Innovation Management*, *3*(3), 205–206.

Thompson, K. (2021). What is agile for hardware development? https://www.cprime.com/resources/blog/what-is-agile-hardware-development/

Vázquez-Bustelo, D., & Avella, L. (2006). Agile manufacturing: Industrial case studies in Spain. *Technovation*, *26*(10), 1147–1161.

Vinodh, S., Devadasan, S. R., Maheshkumar, S., Aravindakshan, M., Arumugam, M., & Balakrishnan, K. (2010). Agile product development through CAD and rapid prototyping technologies: an examination in a traditional pump-manufacturing company. *The International Journal of Advanced Manufacturing Technology*, *46*, 663–679.

Wysocki, R. K. (2011). *Effective project management: traditional, agile, extreme*. John Wiley & Sons.