LOUVAIN
School of Management

UNIVERSITE CATHOLIQUE DE LOUVAIN

LOUVAIN SCHOOL OF MANAGEMENT

Visual Thinking in an agile environment

NON CONFIDENTIAL

Promotor : Samedi HENG
Co-promotor: Prof. Manuel KOLP

Research Master's Thesis
Submitted by
Aurélien DUMONT DE CHASSART

With a view of getting a Master's degree in
Business Engineering

ACADEMIC YEAR 2015-2016

I would like to warmly thank all those who supported me to achieve this thesis.

My first thoughts go naturally to my promotor, Samedi Heng, for his precious availability and useful advice. He has guided me with his expertise and always ensured high quality follow-up. I am sincerely grateful for everything he has done for me.

I also wish to thank my co-promotor, professor Manuel Kolp, for his valuable time spent reviewing my thesis.

Another special thank should be addressed to my internship supervisor, Arik Azoulay, who made me discover, through numerous discussions, the potential of Visual Thinking in an agile environment.

This thesis could finally not have been realised without the continuous support of my parents and close friends who encouraged and supported me through all of this.

Table of contents

Background and Research question

Since a few decades, the innovation and technology in business has become increasingly more dynamic. As a result, software development methodologies turned out to be crucial in the area of information systems. The use of traditional methodologies no longer met the strong need of flexibility, adaptability and rapidity (Gupta, 2009). In that context, the agile methods were developed to better deal with the fast changing needs and requirements.

From the writing of the *Agile Manifesto* in 2001 onwards, the agile community has developed enormously. But at the same time, little by little, the agile concept has encountered numerous challenges. As a reaction, agile practitioners get inspired from visual project management (Williams, 2015) to solve some of these challenges by using more and more visualisation tools to manage successfully their software development projects in an agile environment. Visual Thinking started thus playing an important role in agile project management with tools such as whiteboards and sticky notes, but still remains relatively limited in requirements engineering. Possible reasons for that could be that the Agile Manifesto (Beck et al., 2001) never mentioned the importance of visualising requirements and that agile practitioners are contented with simple text based User Stories artefacts. On the other hand, we observe that there is a clear need to build a shared understanding of the changing requirements and visualise the big picture of the project. This need is precisely met by employing visualisation tools throughout the whole agile project. A shift towards Visual Thinking is thus going on within the agile environment and this latter seems logical as a lot of agile values and principles can be applied by appealing to our visual senses.

The research question that will be analysed in this thesis is thus:

To what extent does Visual Thinking address the existing challenges within the agile environment?

In order to answer in detail this research question, an extensive preliminary study will need to be done with 6 research goals:

- Defining the initial *key values and principles* needed in an agile environment, based mainly on the Manifesto;
- Identifying the underlying core *challenges* related to these fundamental agile values;
- Overviewing the *two most used methodologies* in an agile environment (XP and Scrum);
- Defining the link between *Visual Thinking* and the agile environment in general and discuss how combining them could benefit a project;
- Overviewing the set of tools encouraging Visual Thinking throughout the agile Software Development Life Cycle and discuss whether they address the challenges related to the fundamental agile values;
- Discussing a case study of Visual Thinking in an agile environment.

The goal of this thesis is to contribute to the agile literature by linking together two widely used concepts which are 'agile software development' and 'Visual Thinking'. Various Visual Thinking tools are already used by agile practitioners throughout their agile project, but no research has been conducted yet on the benefits of using these tools in an agile environment. This thesis will thus firstly identify challenges which are harmful to the core values depicted in the Agile Manifesto and then analyse whether or not the existing Visual Thinking tools in the literature address those challenges.

Research methodology

The first 3 research goals of this thesis will be discussed using a qualitative approach. This step can be considered as the preliminary study before answering the research question and will be conducted using an explorative and descriptive approach. In this study, a whole range of sources from the literature will be discussed in order to clarify and analyse some key concept and ideas. Formal sources like books, articles in peer-reviewed magazines or research papers will be preferred to informal sources like websites, personal blogs of agile practitioners or youtube videos. The *Literature review* methodology will be used in this first step as this methodology is useful when a large amount of information in the literature needs to be gathered about a specific subject (Fink, 2010). However, a limitation to this method could be the lack of information about the criteria of selection of the specific sources in the review.

The two next research goals are also discussed using a qualitative approach. The research methodology of selecting the most relevant tools is explained in the related chapter. Here again, we will firstly look for formal sources such as books and articles, usually written by the founders of the tools. As these sources are limited in the literature and for the sake of gathering multiple perspectives on the subject, we will complete our research with informal sources, using Google as search tool. During this research personal blogs of respected agile practitioners will be chosen over forums which will be completely avoided. Throughout the research, the experience of the author in an agile consultancy firm could sometimes by used to illustrate more concretely a complex concept, this addition will be, whenever possible, confirmed with formal or informal sources.

The first limitation of this methodology is the lack of in depth interviews which could have been interesting in this research. Indeed, according to one of the basic agile principles which is collaboration with the customer, developing a new methodology should be done with close customer (i.e. an agile practitioner) collaboration. However, such a collaboration is

very time consuming for a busy agile practitioners and was impossible to organise. One could also argue that no quantitative research has been conducted in this thesis. Conducting such a research among students was indeed a possibility but would not have been very reliable as a large majority of them do not know anything about the agile environment. Conducting a quantitative research among agile practitioners would have led to a very limited size of the sample.

Chapter 1: The agile environment and its challenges

1.1 Introduction

Every study or thesis about the agile environment should start by discussing the Agile Manifesto. How else is it possible to understand this environment without explaining the basics of it. Seventeen independent thinkers have laid the solid foundations in 2001 of a philosophy that gave birth to dozens of various methods that, in turn, developed hundreds of working software. Here is where it all started.

1.2 Context of the Manifesto

To clearly understand the history and context in which the Manifesto has been written, we have to go back in the 1990's. At that time, the software development in businesses began to proliferate, but in the meantime, the majority of the software projects were chaotically managed. That time was widely referred to as "The application development crisis" (Aitken & Llango, 2013). As a result, the time lapse between the identification of the business need and the delivery of the final software was way too long. Within that time lapse, the business reality and requirements were very likely to change and thus even if the initial objectives were met, many of the IT projects did not meet the current business needs.

In that time of chaos, the business leaders were looking for something that was more flexible and responsive to changes. Little by little, experts were trying to build new simpler ways of developing software without all that documentation overhead of the previous *waterfall* techniques. So during the 1990's, various lightweight development methods evolved as a reaction to these heavily regulated and over processed methods (Agile Alliance, 2013). A bunch of people started experimenting with XP, rapid application development and

Adaptive Software Development. Although these methods emerged before the Manifesto, they are now referred to as agile methods (Larman, 2004).

A group of seventeen unhappy people about the current chaos of heavyweight software development processes, representatives from the different existing methodologies, meet together to share their thoughts and find a common way to focus on writing working software. At this meeting, they all agree that the only thing which was important are the values in writing the software. So what emerged from this meeting was a symbolic *Manifesto for Agile software Development*, signed by all 17 participants (Highsmith & Fowler, 2001).

Although the initial concerns of many participants like those of Alistair Cockburns "I personally didn't expect that this particular group of agilites to ever agree on anything substantive" (Highsmith & Fowler, 2001), the feelings afterwards were much better, everybody would look back to the result with some pride.

> *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.*

Source: Beck et al. (2001)

Taken these 4 values, 12 other principles for agile software development were behind this Manifesto (Beck et al., 2001):

- *Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.*
- *Welcome **changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.*

- ***Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
- *Business people and developers must **work together** daily throughout the project.*
- *Build projects **around motivated individuals**. Give them the environment and support they need, and trust them to get the job done.*
- *The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.*
- ***Working software** is the primary measure of progress.*
- *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a **constant pace** indefinitely.*
- *Continuous attention to **technical excellence** and good design enhances agility.*
- ***Simplicity**--the art of maximizing the amount of work not done--is essential.*
- *The best architectures, requirements, and designs emerge from **self-organizing teams**.*
- *At regular intervals, the team **reflects on how to become more effective**, then tunes and adjusts its behavior accordingly.*

It is important to note that the authors agree upon common values and principles in the Manifesto, which is only a guide for agile methods. This does not mean that the Manifesto is an agile methodology on itself. It can be applied in many different ways by other development methodologies (Beck et al., 2001; Hazzan & Dubinsky, 2008).

1.3 The four values of the Manifesto and its related challenges

1.3.1 Individuals and interactions

When people work in an agile environment, they are probably using many of common agile-related words such as user stories, product backlogs, task boards and the list goes on and on. When you ask someone if he is working in an agile environment, he will often give an answer such as "We are doing scrum" (Holub, 2014). This answer as well as the set of tools just mentioned mean that the process is more important than the people, while the very first value of the Manifesto is just saying that we should avoid it and prefer *individuals and interactions over processes and tools* (Beck et al., 2001).

One reason for the lack of attention to this first value of the Manifesto is because the majority of the people who are actually developing software, comes from computer "science" or software "engineering" studies. These people are used to solve problems with black or white solutions. As a result, the most complex and difficult **challenges** software companies are facing are more **related to people** and human behaviour (Rogers & Howard, 2011). The same can be observed in the literature. One of the first popular agile books from Alistair Cockburn, *Agile Software Development*, explains the agile world through human interactions in the development of software. But since that book was published, almost no study has been done on this important part of the agile environment (Rogers & Howard, 2011).

This distinction is also related to the statement of Taylor in 1911 evoking his **scientific management** theory: "In the past, the man has been first, in the future the system must be first" (Taylor, 1911). In the past, people who had to do the work could decide how they want to do it. But according to Taylor, this had to change. A separate group of people, the system designers should from now on decide the process that other people have to follow. This approach still influences the traditional approach of how we develop software. System designers first work on the software development process and then they try to find people who fit into the different steps of the process (Fowler & Ford, 2013).

An agile practitioner should however put the people in the first place. Individuals are more important than anything and the whole company should be organised to support these individuals and maintain constant pace (Beck et al., 2001). According to Alistair Cockburn, people are highly **variable and non-linear**. Putting them in a rigid process will only lead to failure as one process does not fit all teams neither all projects (Layton, 2012). System experts wrongly compare managing individuals to building software (Cockburn, 2002), individuals are not as predictable as software are. Cockburn believes that people as primary actors in a project are a key agile notion. We have to reverse everything, rather than coming up with a good process and fitting people into it, we need to come up with a group of people that can work efficiently together and they will choose on their own the best process to

follow. Doing this, communication will go faster and more efficiently, better teamwork will be created and team members are more willing to innovate as they have more responsibilities and are more self-organised (Layton, 2012).

Although, this does not mean that you cannot use any tools or processes in an agile environment, because tools and processes are inevitable in a project. But when you use some, these should directly support value creation for the software. Along with this idea, the more a process or tool is complex and robust, the more money and time is spent on it and the more money and time is lost to create value for the software (Holub, 2014).

Summary of the underlying challenges related to the first value
• Supporting interactions between individuals to avoid communication breakdown • Setting up flexible processes that support highly variable and non-linear people

1.3.2 Working software

The second value of the Manifesto is easier to understand and more widely spread. The large majority of agile authors mention this value in their works and *working software* is a common found in the 5 core agile methodologies (DSDM, XP, Scrum, ASD, Crystal) analysed by Strode while minimising documentation is also a common goal to the 5 methodologies (Strode, 2005). Nevertheless, this value may be the most difficult one to apply as it concerns the highest level goal, the working software is after all the reason of any IT project (Holub, 2014).

So an agile development team should focus on producing a working software. The only way to measure if a specific feature is working, is to check if it **fulfils the associated requirements** (Layton, 2012). In software development, this means the definition of done: developed, reviewed, tested, deployed and documented (Waters, 2007). But this does not mean that we

cannot create the **documentation** we need, it just has to be **as simple as possible** when it makes sense to write it down (Leffingwell, 2011). Indeed, agile has always been misinterpreted as being completely against documentation, but the Manifesto only says that we should prefer a *working software over comprehensive documentation* (Beck et al., 2001). The real point here is to find the right balance between documentation and discussion. The authors of the Manifesto only wanted to highlight the fact that in the past, developers where way to much focused on documentation (Cohn, 2010).

The **IT measurement inversion** illustrates well this value. It shows that the cost of developing a software has no impact on the investment decision, while the most significant information value is whether the project will be cancelled, followed by the utilisation of the system (Hubbard, 2007). As a result, a good agile practitioner should analyse any documentation that distracts him from actual development to see whether it supports or undermines the value creation for the software (Layton, 2012).

Scrum is one of the agile methodologies that favour a working software over comprehensive documentation. As we will see in the next section, Scrum works with short time boxes. By the end of each one of these, the development team has a **whole-team commitment to deliver a working software**. This way of working will encourage the team members to find ways to eliminate unnecessary documentation (Cohn, 2010). What would happen if your customer told you that they run out of many and that the project needs to stop right now when 60% of the project is done? In a traditional project, no working software could be delivered because 60% of the project is still in progress and nothing is done yet. However, in an agile project, 60% of the highest priority features would be done and could be delivered immediately.

Summary of the underlying challenges related to the second value
• Writing only simple and useful documentation that will not undermine the value creation for the software • Having a whole-team commitment to deliver a working software after each iteration

1.3.3 Customer collaboration

Given the fact that we are highly social creatures, communication is one of the most important aspect in the area of software development. Indeed, an overall majority of the IT projects fail because of a **communication** issue instead of a technological problem (Fowler & Ford, 2013). We could take all the outsourcing efforts as illustration. These efforts are very challenging and seldom successful for the simple reason that developers are isolated from the product owners and customers, that communication is going less smoothly and thus that the vital feedback loop is broken. The authors of the Manifesto realised the importance of communication during a software development project and wrote it down in the first and the third agile value: *Customer collaboration over contract negotiation* (Beck et al., 2001).

Indeed, an agile project success does not only rely on intra team communication mentioned in the first value, but can be extended to the importance of a collaborative relationship with the customer. As a result, most of the agile methodologies try to make the **customer part of the project** on an ongoing basis. This will also increase customer satisfaction and will ensure that the changes can be incorporated smoothly in the project (Layton, 2012). In order to fully benefit from these advantages, Holub (2014) even argues that all the members of an agile project should work together in the same room on a daily basis throughout the project. Scrum practices encourage customers to see the software at the end of each sprint to ensure maximum value creation from the first sprint of the project onwards, when still little money of the customer has been spent on it (Strode, 2005).

Furthermore, a customer does not really know what is hard or easy to develop in a project. When he comes up with a little feature that takes two weeks to develop, another feature that looks similar to him could take a lot of months to develop due to a higher technical complexity. So if developers and customers start collaborating, they will find alternative solutions working well for both of them (Holub, 2014). The **Planning Game** practice from XP in which the customer collaborates with the development team to make predictions about the weights of future user stories could solve this kind of situation (Cohn, 2004). While the historical focus on negotiation will only create an adversary relationship between the actors which will lead to misunderstandings or disruptions when any changes in the requirements occur.

Summary of the underlying challenges related to the third value
• Making the customer part of the project on a daily basis • Showing the software to the customer after each iteration • Including the customer for the ongoing planning

1.3.4 Responding to change

In a world where traditional *waterfall methods* are valid, all the results are predictable. You first plan your work and then work your plan. This predictive approach would be effective if everybody accomplishes his work according to his plan. However, this statement depends on the fact that we can get a stable set of requirements, although this is almost never true in our fast changing business world (Fowler & Ford, 2013). As a result, **predictive plan driven approaches** will try to stabilize the requirements to work according to the plan and ignore all the unforeseen changes that will arise in the project (Bennet-Lovsey, 2015). They will attempt to wrestle the changes by setting up rigorous procedures and budgets which cannot incorporate new requirements. In an agile world, however, we will embrace these changes

in requirements and go even further by considering these changes as a competitive advantage for the software (Poppendieck & Poppendieck, 2003).

This is why the last agile value of the Manifesto addresses the response to changes. Agile teams that can **respond quickly and effectively to changes** from the customers, will be able to create even more value in the software development. What is the point to develop a software that would have been useful a few years ago when the requirements where written? Any new event becomes thus an opportunity to provide additional value to the software instead of trying to blindly follow a plan (Layton, 2012). In fact, this capability to accommodate changes systematically provides stability to the project and increases its change for success.

Moreover, you need to be able to respond to changes at all levels in the agile organisation, thus, this value also counts for the **changes in the processes itself**. If a process is not working well, agile practitioners should have the freedom to just change it. In theory, the perfect agile team should not even do retrospectives during the project. If a bad process appears, the team should gather to find a solution and move on immediately (Holub, 2014). As we will see in the additional key concepts, if you are not constantly working to improve your processes, they will gradually degrade over time (Humble, 2015).

Summary of the underlying challenges related to the fourth value
• Responding quickly and effectively to changes in requirements • Being able to change your own processes

1.4 Additional key agile concepts

The initial core values are well reflected in the Agile Manifesto. However, through numerous readings, two additional key concepts stand out and deserve our attention. The *empirical process* control has been discussed and analysed by a whole range of agile experts and

pioneers (Schwaber & Beedle, 2002; Larman & Vodde, 2008; Fowler & Ford, 2013; Holub, 2014) as well as the importance to make the whole organisation agile (Larman & Vodde, 2008; Larman, 2012).

1.4.1 Empirical process

Defined process

Let us firstly have a look at the description of Ken Schwaber to better understand what is a defined process in Agile Software Development:

*"The defined process control model requires that every piece of work be completely understood. Given a well-defined set of inputs, the **same outputs are generated every time**. A defined process can be started and allowed to run until completion, with the same results every time."* (Schwaber & Beedle, 2002).

Ken Schwaber is one of the first agile experts who made a difference between an empirical process and the defined process in an agile environment. With the latter one, the team is able to set everything in advance and will always obtain the same results if he runs the process several times. It is like following a recipe to bake a fruitcake, with the same ingredients, you will probably obtain always the same fruitcake if you follow correctly the different steps. If this would be the same for software development, the project process should be predictable, with well known requirements from the beginning and no changes would appear during the execution phase. Of course, this is not the case in the real world.

Empirical process in an agile environment

A simple example to illustrate an empirical process is when you take a **shower in an unknown place**. The whole process of finding the right temperature is an empirical process. The two hot and cold water valves are the inputs, but when it is the first time you use them, you have no idea what temperature will be your shower (= the output). You have no idea

which inputs you need to get the desired output. As a result, you will try some inputs, inspect the first outputs and then adjust the input again based on your conclusions from the output. This is a typical empirical process.

This process happens exactly in a similar manner during the software development. A whole range of unpredictable inputs like constantly changing requirements and complex people force us to adopt an empirical approach and to get through a feedback loop (Fowler & Ford, 2013). The information is collected by **observing and experimenting**. Indeed, software development is too complex to elaborate a defined process. The empirical process is a core concept in the Scrum methodology (Strode, 2015). Scrum emphasizes it through three concepts: transparency, inspection and adaptation. Instead of fixing the scope of the software and the processes to develop it before the project, short developing cycles are preferred to observe and inspect the output of every cycle and adapt it for the next cycles (Schwaber & Beedle, 2002). Thanks to principles like transparency and self-organising teams, subtle control is kept on the empirical process.

Developers have tried too many times to use highly defined processes on complex software development, but it almost never works. However, we can still find countless people who will pretend having found the correct detailed process and try to push it into organisation through useless "Agile trainings" (Rossberg, 2014). If an empirical process is used well, every agile team will create his own process on every new project, based on early observation and experimentation, instead of receiving a complex or prescriptive methodology pushed into the team. Researchers should thus more focus on providing people knowledge on developing their **own thinking tools to create their own processes** and frameworks adapted to each different project (Larman & Vodde, 2008).

But one important question remains. How do you know what to do if you cannot take processes and practices from others? Indeed, an agile team seems to be on his own. The answer is very simple according to Dave Thomas. An agilist will do exactly the same as he will do in any other unknown situation, he will **take a small step forward, examine the situation, revise his plan and retake a small step forward** to repeat the process (Thomas, 2015).

Larman and Vodde also address the issue. They admit that simple, adoptable practices and framework help new agile team get started. But this should only be the starting point to later on adopt deeper agile principles through their own processes. Indeed, when a new agile teams starts, concrete guidance on developing thinking skills to improve your processes does not offer any help, so they reflected on how to communicate effectively this guidance to new teams (Larman & Vodde, 2014).

Feedback loops

The empirical process brings us naturally to the crucial feedback loops in an agile project. Given all the uncertainties inherent in an IT project and the fact that people do not know what they need until they have it in their own hands, make the feedback loops a central principle in an agile world (Holub, 2014). Indeed, the sooner users see the software, the faster they will change their minds about their initial ideas (Cohn, 2014). **Constantly questioning** whether we are working effectively on the project is vital for the project. In early stage we want short and very fast testing loops to receive feedback on what we are doing. Later on, little by little, more detailed and consequent tests are performed on the whole project (Fowler & Ford, 2013). If feedback loops are delayed on a project, this will compromise early productivity and the creation of value in the software (Leffingwell, 2011).

Kniberg and Skarin discussed the various feedback loops that exist within Scrum and XP. Pair programming, unit testing and continuous integration are for instance three XP practices that provide feedback on a very short period of time, while short sprints, code review and daily scrum meetings are practices within Scrum that provide slightly longer feedback loops (Kniberg & Skarin, 2010). Leffingwell also explained the advantage of writing the smallest possible user stories, although these deliver lower global value on the project, these user stories are developed in a shorter period of time and will go faster through the iteration for faster feedback (Leffingwell, 2011). If the feedback loops is **as short as possible**, the agile team will be able to adapt the process or the software quickly. In that same spirit, having all

the agile team members and the customer work closely with each other, will support these crucial loops.

The improvement kata model

The empirical process explains us that the process and practices we have are never the best ones, but the important thing is to constantly question yourself how to improve those processes. If an agile team is not constantly working on improving its processes and practices, these will gradually degrade over time (Humble, 2015). An agile project is like doing sports or music, you need to constantly train and improve yourself over and over again. The improvement work should become a habitual process. The improvement kata understands this and explains in a simple four-step model how to get **from an actual situation to a new better situation** (Rother, 2009). First, you need to have a vision or direction in which you want to go, then you grasp the current condition, when your target situation is defined, you can then move towards it iteratively. The point here is that this last step is not to create a specific plan to get to the desired situation, but to experiment with various ideas through an empirical process to achieve what you want. This improvement kata model is illustrated in Figure 1.

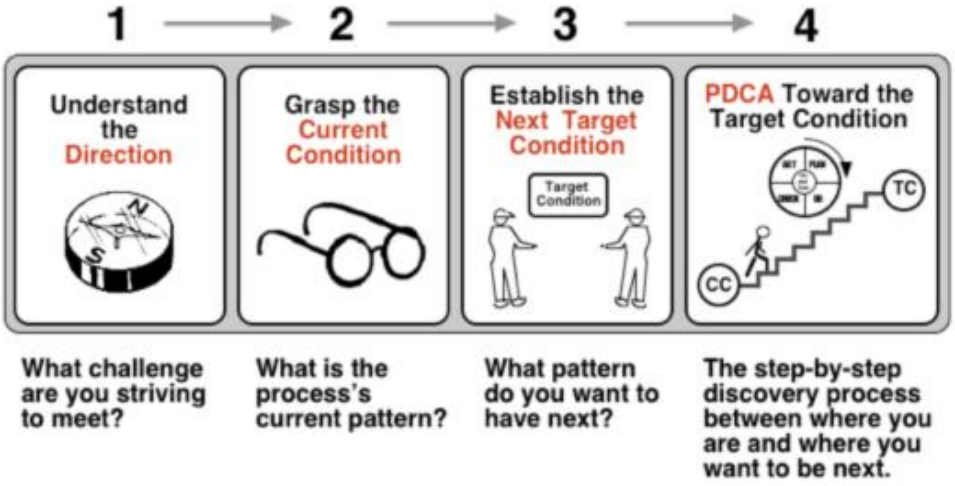

Figure 1: The improvement kata model
Source: *Rother (2009*)

Let us conclude this first additional principle with two thoughts from two agile pioneers. Firstly, Larman explained that agile is all about simplifying the organisational system instead of adding some agile tools on the top of any existing complex organisation (Larman & Vodde, 2015). Dave Thomas could add to this that it is time to reclaim agility by giving the simplest definition of it: **"**Three steps plus a loop**"**. Similar to the improvement kata model, you firstly need to understand where you are, take a small step toward where you want to be and then evaluate that step and repeat. If you are facing various options, you need to choose the one which is the easiest to change in the future (Thomas, 2015).

Summary of the underlying challenges related to the "empirical process" concept
• Promoting observation and experimentation • Supporting self-organised teams that create their own tools and processes • Creating short feedback loops

1.4.2 The whole organisation

The second idea that deserves our attention is the importance of making the whole organisation agile. We have to understand that agile is an organisational "culture expressed through a set of practices" (Küpper, 2016; Holub, 2014; Martin, 2014). A team needs to be agile before it tries to use any agile method, otherwise it will fail for sure. This idea has been applied to the whole organisation by Kahkonen and Abrahamsson (2003). Indeed, we have already seen that the process is not important as no specific detailed process is better than another. As a result, an **organisational culture** has to be set up to help its people to continuously rethink their own processes (Holub, 2014). This has to be a **top down approach**, if the CEO does not understand and support this agile culture, it will not work.

Doing so, the whole organisation will be structured into small cycles with results every couple of weeks on which feedback loops are created (Fowler & Ford, 2013).

When we ask any organisation if they are agile, every single manager will answer yes. However, when we then actually check in that organisation, we realise that the agile culture **stops at the boundaries of the software development teams** (Gothelf, 2014). The rest of the organisation is still focused on supporting the existing and fixed processes. But if processes change by being continuously improved, the whole organisational structure that supports these need to change as well (Holub, 2014). Bringing in trainers to teach how to adopt some agile methods is of course not enough.

But *why is it so difficult to make the entire organisation agile*? Holub gives an answer to this question by explaining the **reluctance of the quality insurance department** as an illustration. In every agile process, testing is already integrated in the development process. When that process is done, no work is left for the QA department and this latter would be useless (Holub, 2014). Larman goes even further. According to him, the management layers will interfere with an agile model. Indeed, in a lot of organisation, there is a large amount of management overhead. Most of them mistakenly believe that if they can adopt agile processes and still keep all this **management overhead**, but most of it is already integrated in the agile team (Larman, 2012). Indeed, agile is about trusting people to do their job (Fowler & Ford, 2013), nobody is ordering someone else as it is a collaborative kind of process (Holub, 2014). These are the reasons why making the whole organisation agile is crucial. In an agile organisation, these people should not be afraid of losing their jobs because **job safety** will be preferred **over role safety** (Larman, 2012). Furthermore, this management overhead will also extend and slow down the decision-making processes, while these processes need to be as close as possible to the customer to be able to quickly analyse and take decisions (Gothelf, 2014).

The **metaphor of rugby** illustrates these principles very well. In rugby, the whole team is working closely together to bring the ball forward. The ball represents metaphorically the delivering of valuable features. Moreover, rugby implies that the players have some

specialities but when something else needs to be done, you do not have to stick to your role. For instance, if the ball is on the ground and I am a kicker, I will pick up the ball to take it forward instead of waiting for someone else. This is the fundamental thinking mistake called **local optimization instead of system optimization** (Brechner, 2015, p. 135). If we look at one specific job in an organisation that seems to perform outstandingly well, it could be especially because of him that the whole organisation is delivering slower value. An agile organisation need people who are willing to say "My job is a kicker, but what my team needs right now is moving the ball further" (Larman, 2012). People naturally think that everybody should be busy at his own task any time, which is the common mistake of working the job title.

Summary of the underlying challenges related to the whole organisation concept[1]
• Having an organisational culture that supports agility • Ensuring job safety instead of role safety • Encouraging system optimization rather than local optimization

1.5 An overview of two agile methodologies

1.5.1 Introduction

Highsmith stated in 2001 that "the Agile movement is not anti-methodology, in fact, many of us want to restore credibility to the word methodology". Since the Agile Manifesto (Beck, et al., 2001), the popularity of the agile methods has increased drastically. In 2005, there were about 13 published agile methods (Strode, 2005) but that number has not stopped to grow since then.

[1] A summary of all the challenges is shown in Appendix 1

We have chosen to overview two of the earliest published methods: *eXtreme Programming* (XP) (Beck, 2000) and *Scrum* (Schwaber & Beelde, 2001). XP is one of the most well-known agile methodologies and furthermore, according to Strode (2005), XP is deemed to be a methodology in line with most (not all) of the agile values of the Manifesto and thus being one of the most 'agile' methodologies. Scrum, for its part, is the other popular method used in a whole range of organisations. Scrum is also known for meeting most of the values and principles in the Manifesto (Schwaber & Beelde, 2001). In the following, we will find a short overview of these two methodologies, a summary of the techniques used and a description of the possible issues related to each method.

1.5.2 eXtreme Programming (XP)

The earliest publication about eXtreme Programming dates back from 2000 with the book *Extreme programming explained* by Kent Beck. Beck first described his methodology as an answer to the constantly changing requirements issues and is defined as a consistent set of values and practices to work well together in practices (van Valkenhoef et al., 2011). Beck defines XP as a "lightweight methodology" which is based on five core values: *simplicity*, *communication*, *feedback*, *courage* and *respect* (Lindstrom & Jeffries, 2004). We can notice the clear similarities between these latter and the values of the Agile Manifesto. Beck's fundamental idea is to start with the **simplest** solution that works. This will create quick **feedback** from the customer, but other feedback mechanisms from the system and from the team are also supported within XP. **Communication** is the next value which is closely related to the two first ones. Documentation is abandoned in favour of face to face communication in open workspaces in order to give all the team a shared view of the system. The **respect** value is also fundamental to create trust and a positive atmosphere in the team. **Courage** is the last value that forces you to try and to not be afraid of failure.

Next to these values, Beck also describes a whole set of 12 interdependent practices that are intended to be more concrete for actual guidance in ad-hoc situations.

- Pair programming
- Planning Game
- Test-driven development
- Whole team
- Continuous integration
- Refactoring
- Small releases
- Coding standards
- Collective code ownership
- Simple design
- System metaphor
- Sustainable pace

Beck is the first who comes with the idea that code should be written by **pair programmers**. While one developer is coding, the other is more focused on the big picture of the code and will continuously review the code of the other. The **planning game** is a vital communication point once every iteration, where the **whole team**, developers and customers, gather to plan the next user stories. XP uses also a **test-driven approach** because Beck believes that we need to test the code to know if it is the right one, as a result XP teams perform a whole set of test such as unit test, run tests and fail tests. The methodology also supports the presence of a **simple design** with **small releases** to create shorter cycles and effective feedback loops. Next, XP encourage **continuous integration** and to **refactor** courageously your code when a simpler solution is possible. Everyone is responsible for the code, this **collective code ownership** will help the team to tell a whole story together (**system metaphor**) by using the same **coding standards**. Finally, if these practices and values are respected, **sustainable peace** in the whole team will lead to effective communication and collaboration. In an XP

team, nobody is exclusively specialised in something, every member writes, codes and tests in all stages of the development process (Brown & Topi, 2003).

As we can see, the XP methodology provides techniques and practices for the majority of the values in the Agile Manifesto. The customer is supposed to be central in the team and works on a daily basis with the XP team (Lindstrom & Jeffries, 2004, p.43). XP is the only one methodology of agile methods that offers concrete guidance over the whole project lifecycle (Abrahamsson et al., 2002). This may also be a reason why XP is regularly combined with other methods. However, XP lacks clearly practices for project management and for supporting the customer role which is still too much under pressure during an IT project (Strode, 2005; van Valkenhoef et al., 2011). Further **critics** have been made by various authors. Among those, we find Rosenberg & Stephens (2003) who believe that XP insufficiently supports software design or documentation which could lead to issues in contract negotiation or a deterioration of the customer relationship when the product is not defined precisely enough.

1.5.3 Scrum

Sutherland and Schwaber have introduced another well-known agile methodology called Scrum in 1995. In 2001, Schwaber and Beelde popularised the methodology in their book "Agile Software Development with Scrum". Scrum is created to manage a software development in fast changing environment by using iterative and incremental techniques (Scrum Alliance, 2014). Scrum adopts an empirical approach that enables self-organising teams to deliver quickly which enables short feedback loops (van Waardenburg & van Vliet, 2013). The lightweight software development methodology is now used in small as well as in big Fortune 500 companies around the world (Scrum Alliance, 2014).

Like XP, Scrum also consists of a set of core values (Schwaber & Beelde, 2001; Agile Alliance, 2014) on which all its processes and interaction find their foundations:

- Commitment
- Focus
- Courage
- Openness
- Respect

When the whole team is **focused** and only a few user stories are assigned in short sprints, they will deliver valuable products sooner. The second value is **openness**, the scrum team needs to communicate clearly and unambiguously to everyone on what we are working to better collaborate with the other members of the team. Moreover, **courage** is encouraged to undertake greater challenges by accepting help and positive critics towards improvement from the other members of the team. Thanks to the self-organising teams, greater control allows us to be fully **committed** to the project in order to achieve the desired goal. Finally, the whole team faces success and failures together, thus it is important to **respect** each member of the Scrum team (Cohn, 2010; Schwaber & Beelde, 2001).

Based on these values, different roles will perform a software development project using among other processes, the scrum workflow. Before explaining the scrum roles and workflow, it is important to mention that scrum is not a process or technique for developing a software, but a framework in which various different processes and techniques can be used depending on every scrum team and project (Lacey, 2012). There are three different roles in a scrum project: the product owner, the development team and the scrum master (Scrum Alliance, 2014). The **product owner** represents the customer. He is responsible for managing the product backlog, this means adding or removing features and prioritise them. Moreover, he is the role who maximizes the value creation of the features developed by the development team. The **scrum master** is the one who helps the development team to achieve their goals within the committed boundaries and make sure the whole team is working according to the scrum values. Lastly, the **development team** is doing the actual development work. Its goal is to deliver a working product at the end of each sprint. It is important to note that scrum teams are self-organised and take their own decisions. (Scrum

Alliance, 2014). No management overhead interferes with their decision making processes, this ensures higher flexibility and productivity, next to the short cycles and the collaborative nature of the scrum teams (Lacey, 2012).

The workflow starts with the elaboration of a **product backlog** that contains a list of required features that could go into the product. The product owner prioritizes this list according to the value creation of each feature (Cohn, 2010). Although Schaber & Beedle (2001) never mention user stories, these are a current tool to represent these features. During the **sprint planning**, the product owner, the scrum master and the whole team discuss the top priorities in the product backlog to determine what will go into the next sprint. The result of this meeting is the **sprint backlog**. This is the list of features that are assigned to the next sprint. Every member of the team knows at this point exactly what contains every feature and customer as well as development team should share the same understanding of these features (Sutherland, 2008). Then comes the **sprint**, which is a 1 to 3 weeks time box during which the features in the sprint backlog are developed. During the sprint, short daily meetings are organised in order to keep track of global project progress, share useful information and discuss who is working on what feature. The outcome of each sprint is a potentially shippable product increment (Scrum Alliance, 2014). In order to check if the sprint goal is achieved, a sprint review is organised at the end of each sprint. During this meeting, the development team presents the developed features during the sprint to the customer. Finally, a **retrospective meeting** is also organised to discuss the possible issues that occurs during the last sprint. This workflow is then repeated for each sprint. You can find an illustration of the workflow on Figure 2.

Figure 2: Scrum workflow
Source: Scrum Alliance (2014)

According to Dave Thomas (2015), scrum is a very powerful methodology to get the whole team onboard, but the implementation of it is usually very poor. Thomas **critics** fiercely the short scrum master courses and all the books on scrum explaining how to adopt the processes without understanding the underlying values of it. Indeed, when explained wrongly, scrum has **no respect for the existing processes** in the organisation (Holub, 2014), while we already discussed that when we change the processes, we need to change the organisation that supports those processes. New scrum teams will thus read those scrum books, follow blindly the processes and will get stuck these without understanding clearly what and why they do it. Critics were also made by Holub (2014) against the **rigid time box**, which goes straight against the idea of agility. Finally, some authors also believe that scrum lacks some practices for the actual **design, code and test** of the development process (Lacey, 2012).

As a conclusion, we can say that these two methods were founded to meet most of the agile values from the Manifesto. However, certain practices provided by XP are lacking in scrum and vice versa. This gave the idea to **combine the two methods** in order to benefit from

their complementarities. Indeed, XP lacks an "agile guardian" role who provides a proper agile environment (Strode, 2005), this role is ensured by the scrum master in the scrum methodology. Furthermore, scrum does not address techniques for designing, coding and testing (Lacey, 2012), which is provided by XP. Combining these two methods could create a third improved method that could **meet** maybe all the values of the Agile Manifesto (Strode, 2005).

28.

Chapter 2: Visual Thinking

2.1 Introduction

In today's lean business culture where time is money, busy managers do not have the time to write and read long project reports and documentation. Given the fact that the speed of business is always increasing and that the projects are extremely data rich business processes, a shift towards less paper-based and more agile-based methods has take place (Williams, 2015). Indeed, in an IT project, data is constantly captured, transformed and communicated to a lot of different individuals with all different backgrounds and expertise. Managing all these projects in a more visual way could be a possible solution to this challenge to improve effective communication and shared understanding throughout the project. This led to the development of a whole set of data visualisation tools that helps managers take right decisions within increasingly shorter time frames, but the power of visualisation goes way beyond that. We will discuss in this chapter how Visual Thinking could help for software development in an agile environment.

2.2 The concept of Visual Thinking

Let us firstly discuss the Visual Thinking concept before integrating it in an agile environment. Visual Thinking is the common idea of thinking and processing words using pictures and images. This way of thinking is a natural process for approximatively 60% to 65% of the general population (Deza & Deza, 2009, p.659). Indeed, vision is by far the **most dominant sense** for the human being. Scientific study conducted by Dr. Medina has proven that our brain dedicates more neurons to vision than to the other senses combined (Medina, 2008). This shows that we are much better at remembering images and pictures rather than written or spoken words. Various examples exist in our everyday life to illustrate this. Traffic

signs for instance make use of our Visual Thinking. Every driver will still recognize the stop sign if we take away the actual 'STOP' letters on the board, by the octagonal, red shape. The same counts when we are reading a text. Our brain sees words as patterns that still can be understood when the interior letters are scrambled, as long as the first and the last letters are in place. This fairly ubiquitous meme on the internet illustrates the phenomenon:

"Aoccdrnig to rscheearch at an Elingsh uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht frist and lsat ltteer is at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit a porbelm. Tihs is bcuseae we do not raed ervey lteter by it slef but the wrod as a wlohe."

A lot of **research** has been done to show that pictures beat text. Reading takes much more time for our brain because it needs to see words as tiny pictures and associate them to features to be able to read them, which is time consuming and inefficient for the human being (Medina, 2008). Moreover, research has also been made on *Multiple intelligence theory* that recognises different forms of intelligence such as namely spatial, musical, linguistic, mathematical, interpersonal etc. (Gardner, 2011). Maria J. Krabbe Stichting Beelddenken, a non profit organisation from the Netherlands, also conducted multiple research approaches on picture thinking or '*Beelddenken*' to identify if children think primarily through visual support. Finally, the science has also confirmed several times that vision is the dominant sense. Neurodevelopmental optometrist, Dr. Bowan cites several optometrists such as Fixot and Skeffington, who tells us that "50% of neural tissue is devoted to vision" (Bowan, 2008) and even more when the eyes are open.

We could go further by citing way more sources in the area of storytelling, pattern recognition, neuroscience, to prove and confirm the importance of using Visual Thinking to make intangible or complex ideas visible. But let us move on by explaining how Visual Thinking actually works. To understand this, we have to go back to the studies carried out in the 1960s by Roger Sperry. He is the first who shows evidences that the brain has a **left and a right hemisphere** that perform different tasks (Trevarthen, 1990). The right, non-verbal hemisphere is responsible for the more artistic, creative, holistic thoughts, while the left

hemisphere is more responsible for logical, written, mathematical things. Using Visual Thinking tools helps you thus appealing to both side of the brain, allowing them to run in parallel and to give two different meanings and contexts to the message. By using simultaneously different senses and both parts of the brain, people will learn and retain far more effectively the received information (Bowan, 2008). If we hear a piece of information, we will only remember 10% of it three days later, but if you join a picture to that peace of information to combine different senses, 65% of the information will be retained (Medina, 2008).

Albert Einstein, for instance, is a **great visual thinker**. He believed that words and numbers do not play a significant role in the thinking process (Michalko, 2011). He had some trouble for explaining his science with words because he thought schematically about everything. This is the case of many **geniuses** such as da Vinci and Galileo Galilei, who constantly make new combinations. These combinations do not need to be revolutionary, but when you always see different concepts visually, you simple combine concepts that may never have been linked together before. When thinking visually, you may be able to see different things from new angle of perspective when contemplating the same world as anyone else (Sicinski, 2011).

As we already discussed through the literature, the science has already proven that the brain of the majority of the population transforms information into images within their mind. Furthermore, we spend most of our day staring at visual medium, but when we face a business problem, we will still go to words and documents to solve it (Williams, 2015). Visual Thinking has thus provided a **myriad of tools** to improve our thinking processes in organisation by appealing to our visual senses. These tools and techniques are very easy to apply immediately in a business environment. They will procure a simple and rapid way to expand our thinking potential and bring clarity to complex concepts that need to be communicated out into simple visual formats (Morasky, 2012). However, many organisations yet underestimate the proven competitive advantage that Visual Thinking can bring to

business at various levels from strategic and product management to marketing and of course project management (Williams, 2015).

In our today's rapid changing business world with huge amounts of data, people will need to understand and make use of the adage that says "a picture is worth a thousand words". Project managers need to digest tons of information in very short time lapses, so visualisation techniques to communicate faster and with greater impact become inevitable. Unfortunately, where there is a need, there will always be an industry that wants to make money from it. Consequently, a simple search on the web about it and you will find a large bunch of companies trying to sell their expensive visual tools. However, Visual Thinking is much more than only a common set of date visualisation tools. It is a way of thinking that needs to be supported by its agile business environment.

2.3 Agile and Visual Thinking, the two concepts combined

While reading the literature about the Visual Thinking concept, it is impossible not to combine it to all the agile values and methodologies as both ideas are highly related. Yet, besides the range of software vendors trying to sell their visualisation tools to agile and scrum teams, these two concepts are often treated separately and almost no research has been made to link these two concepts together. *Is Visual Thinking truly serving the initial values of the Agile Manifesto?* As Dave Thomas explained well in his speech '*Agile is dead, long live agility*' (2015), the values have been totally lost behind the implementation in organisations. In this section, we would like to analyse whether Visual Thinking as a concept at least respects the core ideas of agile, before going deeper in the discussion and overviewing the different Visual Thinking tools. Indeed, **when introducing new concepts** into the agile environment, we need to firstly **start thinking about basic agile** (Larman & Vodde, 2014). If these new concepts do not meet the basic values of agility, it will never work in an agile environment (Holub, 2014).

First of all, Visual Thinking promotes clearly *individuals and interactions* (Beck et al., 2001). As most challenges during an IT project are related to human behaviour, trying to communicate in a more visual way will improve the interactions between the people of an agile team. Indeed, as we have seen, a **team** is composed of various roles ensured by people with a lot of **different backgrounds**. While a developer has a more scientific, mathematical, IT background, he will have to closely interact with a business people and the customer who could have any background. In spite of good intentions, these conversations will too many times **result in misunderstandings** as the different actors will process the received information by comparing it to the knowledge they already have. It is natural that everyone interprets information to fit it into his own world view. (Chapman, 2013). Consequently, Visual Thinking is not only an individual thinking process, it can also be applied in group situations, and this is then called ***visual dialogue*** (Quirk, 2008). Since people have various different mental models, when someone tries to communicate A, other may understand B. This situation can lead to confusion and failure during a software development project. This is well illustrated in the well-known *Tree swing cartoon* (see Figure 3). Visual dialogue encourages people to use images and visual support to explain their thoughts in conversations (Idon Thinking Resources, 2003). Besides avoiding misunderstandings, this will also include the whole team and **provide multiple perspectives** to the problem by combining creatively the ideas. As a result, the whole becomes greater than the sum of the parts of it.

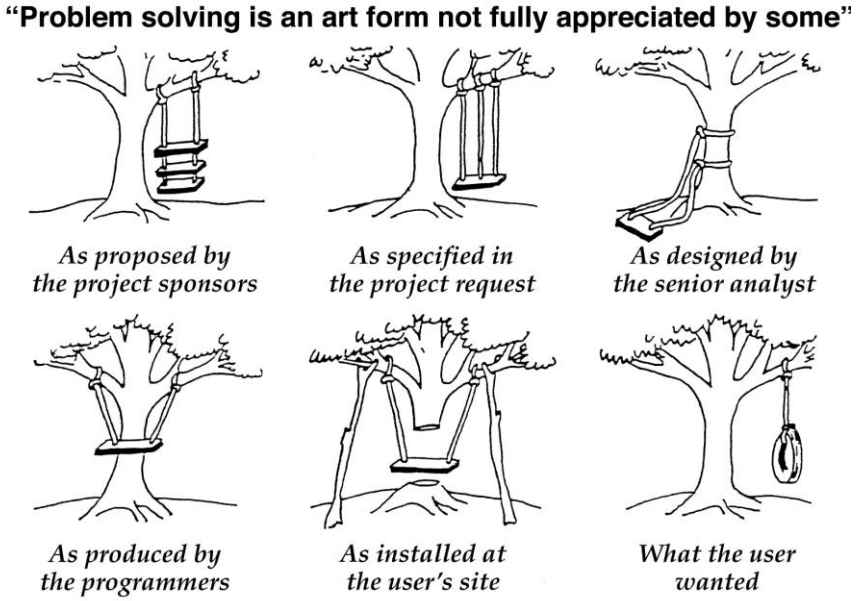

Figure 3: Tree swing cartoon
Source: Chapman (2013)

Fun is the last very simple reason that explains why Visual Thinking ensures better interactions between individuals. Using images and visual support is simply **enjoyable and engaging**. Indeed, fun matters in a highly collaborative team and it enables effective teamwork and increases productivity of the agile team (Kotermansky, 2016).

However, if software vendors rely only on their tools and lose their goal to support interactions between individuals, the first value of the Manifesto is not respected anymore given that tools will prevail instead of individuals. We will discuss these issues in the next chapter.

Next comes the final goal of an IT project, which is producing a *working software over comprehensive documentation* (Beck et al., 2001). *Does Visual Thinking helps an agile team to deliver a working software to the customer?* As we have explained in the previous part, it is clear that using visual support **substitutes the comprehensive documentation** in an effective way given that it has been proven scientifically several times that images are way more easier to retain for the human brain in comparison with texts (Medina, 2008). The

majority of the documentation in an IT project is never read by anybody anyway, and is thus a waste of time to even write it down (Holub, 2014). It happens too often that a development team get bogged down in details, and then loses twice the time by documenting those details which will remain unread (Svarytsevych, 2015). Moreover, evidence proving that Visual Thinking is playing an active role in the development of a *working software* is more difficult to give. As a working software is the final reason of any IT project, the Visual Thinking idea will only serve this latter by respecting all the other values that lead to this working software. If the majority of the values of both concepts are shared, then and only then will we be able to say that the combination of visual and agile principles could lead to a working software.

The reasons why *customer collaboration* (Beck et al., 2001) is also improved through Visual Thinking are similar to those for the first value of the Manifesto. Indeed, Visual Thinking improves interactions taking place both in the team and out of the team with the customer. As the customer comes from another organisation and has a different view on the project, he is in fact the actor who is the most likely to misinterpret the information. This strengthens even more the importance of using images when communicating with the customer. This visual way of communicating will **reduce the probability** of a project failure due to a **communication breakdown** which is one of the most common challenges in a IT project (Kotermansky, 2016). Figure 3 illustrates this well. In that way, using simple visual tools will lead to overall **transparency** and both the agile team and customer have more chance to share a common understanding of the features to develop.

Regarding the fourth and last value of the Agile Manifesto, no definite link can be discussed between *responding to changes* (Beck et al., 2001) and Visual Thinking. The only argument we could give is that the speed of **processing images is infinitely shorter than text** (Medina, 2008). As a result, if changes in requirement have been made by the customer, managing the project in a more visual way will permit faster and more smoothly overall communication and the agile team may respond quickly to the change and turn it into an opportunity (Layton, 2012).

The empirical process is also supported by Visual Thinking. We discussed some characteristics of this kind of process in the previous chapter. It encourages you to take a small step forward to experiment and get feedback. But so many times during an agile project following an empirical process, the team members get lost in the small step details and need to take a step back to see the big picture (Svarytsevych, 2015). This is the moment when visualisation tools are crucial to help you to think in a more holistic way. Furthermore, using Visual Thinking techniques in an empirical process also helps the development team to see the problems **from new perspectives** by appealing to different intelligences and "accelerates learning on the job" (Idon Thinking Resources, 2003). Taking problems from different view angles will indeed support the team to continuously find new ways to create their own most effective processes.

We already discussed it briefly here above, the **whole team gets involved** when Visual Thinking is used on a project. As a matter of fact, unlike complicated texts and reports, a **picture** or draw can be **understood by everyone** regardless his background or expertise. Consequently, everybody can contribute to the conversations as everybody can draw with a little bit practice. This provides a transparent, trustworthy environment in which it becomes easy and appealing to participate and closely work together. This can also be illustrated with the post its, one of the favourite visual tools of agile teams, on the wall of a project room. So when someone walks into the room, even if he does not know anything about the project, he will get immediate insight of the progress of the project. It is important that those post-its are visible and understandable by everybody (Fowler & Ford, 2013) because every member needs to participate. To obtain this, the environment has to be set up to encourage trust, openness and understanding. A comfortable and flexible layout with adequate whiteboards, flip charts and projection equipment that supports the Visual Thinking of the team members is a must-have (Idon Thinking Resources, 2003).

Chapter 3: Visual Thinking tools in an agile environment

3.1 Introduction

We already mentioned it in the previous chapter, given the undoubted benefits provided to a team of using Visual Thinking in an agile environment, the software vendors jumped at the opportunity to sell all kind of different Visual Thinking tools (Williams, 2015). **Pushing these tools** into existing agile teams would be however **contrary** to the basic agile values of the Manifesto (i.e. *individuals and interactions over processes and tools, Beck* et al., 2001). The challenge is thus to use these tools in order to support the interactions between the team members and not the other way round, adapting his own processes which are supposed to be the result of continuous improvement, to the new Visual Thinking tools when adopting them. If the tools are simply constraining and controlling the work of the team, we are going in the wrong direction (Cobb, 2015). Moreover, companies like to buy big expensive tools when looking for a solution to address their needs (Svarytsevych, 2015), but the tools absolutely need to remain as simple as possible. Indeed, the more complex the tool, the more time you spend using that tool instead of providing value to the customer (Shiralige, 2012). This **simplicity** value also appears as a core principle in the Manifesto (Beck et al., 2001) and is promoted by the XP methodology (Beck, 2000; Lindstrom & Jeffries, 2004). Very often, a whiteboard and sticky notes are more than sufficient to use our visual sense during interactions and thinking processes. (Holub, 2014). Simply drawing on a whiteboard while conversing can already help a lot. Everybody can draw and drawings can be understood by anyone, which will support effective communication with the whole team to share a common understanding of the features that need to be developed (Patton, 2014). Leonardo da Vinci already said it "Simplicity is the ultimate sophistication". Finally, before discussing the various existing tools, we need to highlight the fact that there is **no one size fits all solution** (Highsmith & Fowler, 2001), every tool needs to be constantly rethought to fit the team in the best possible way.

3.2 An overview of the literature

Given the wide range of Visual Thinking tools used by the agile practitioners, the goal of this section is to list all these tools proposed in the literature and extract the most widely used tools it. In Section 3.3, the Visual Thinking tools will be further classified. We will base our research on formal as well as informal sources for two reasons. First, the number of formal sources discussing the topic is somewhat limited. Secondly, we believe that the agile philosophy is not only created by agile experts in formal books, but also by all the agile practitioners who daily use these tools in an agile environment. The choice of the formal sources is thus also made by considering the number of times these latter have been cited by the agile practitioners. As a result, the research has been conducted in 15 formal sources, all well cited by the agile practitioners. Moreover, a web search on Google has been done with different combination of the following key words 'Visual Thinking', ’agile', 'visualisation tools', 'eXtreme programming' and 'Scrum'. This allows us to collect a set of 60 relevant informal sources. The informal sources encompass various personal blogs and websites which were preferred over company websites trying to sell their visualisation products. During the research, the occurrence of every encountered Visual Thinking tool has been recorded. This leads to an overview of the occurrences of the Visual Thinking tools in the literature. The results of the research are summarized in Table 1.

Visual tool	Formal sources	Informal sources	Total
Kanban Boards	**15**	**48**	**78**
Prioritized Backlog	**11**	**23**	**45**
Code Visibility tools	1	7	9
Story Mapping	**6**	**11**	**23**
Mind Mapping	**8**	**9**	**25**
Road Map	3	2	8
Burndown charts	**4**	**18**	**26**
Prototyping	**7**	**2**	**16**
Business Process Model	**7**	**5**	**19**
Use-case diagram	**6**	**7**	**19**
Whiteboard	**14**	**9**	**37**
Sticky notes	**12**	**7**	**31**
Process activity diagram	3	3	9
Gamification board	0	1	1
Earned value diagram	0	1	1

Table 1: Occurrences of the Visual Thinking tools in the literature

Some simplification rules have been set up during the search to present as clear as possible results. Firstly, given that the Kanban and the scrum board are usually mixed up in the literature (i.e. a Kanban board is actually a scrum board with a work in progress limit), we have decided to bring the two tools together as a 'Kanban board'. Secondly, a dashboard

combining several tools appeared a few times in the literature, we decided to record each time an occurrence for every tool included in the dashboard. Last but not least, sticky notes and index cards are usually mentioned together throughout the literature, we thus decided to merge both tools by taking the average of their occurrences (i.e. formal sources for sticky notes and index cards: respectively 11 and 13, which makes an average of 12; informal sources: respectively 8 and 6, which makes an average of 7).

In order to give a higher importance to formal sources, we doubled the occurrences of these and added that number to the occurrences of the informal sources to compute the total. For the purpose of keeping only the most cited Visual Thinking tools, we decide to select all the tools with a total amount of occurrences higher than 10 throughout our research material. The selected tools are highlighted in bold print in Table 1.

3.3 Visual Thinking tools throughout the agile SDLC

As shown in Table 1, agile practitioners as well as the literature discuss a wide range of different tools. As these seem to be overviewed in a slightly disparate way, it may be interesting to see how all these tools fit together in the agile Software Development Life Cycle (SDLC) and understand clearly in which phase of the project life cycle an agile team can use these tools. Consequently, the analyse will also tell us more about when a team needs visual support to develop successfully a useful working software.

The agile SDLC looks on a high level very much similar to the traditional SDLC with its phases of requirements definition and analysis, design, implementation, testing and deployment (Leau et al., 2012). However, when we take a more detailed view, the agile SDLC is very different given that the life cycle supports highly collaborative, iterative and incremental approaches. The purpose of Figure 4 is not to discuss in detail the agile SDLC but rather to position each selected visual thinking tool of Table 1 on a very high level view of the agile SDLC. Doing so, we took the phases of an agile SDLC, adapted them to represent them

linearly in order to spatially dispose the widely used Visual Thinking tools underneath, according to when these latter are mostly used during the project life cycles.

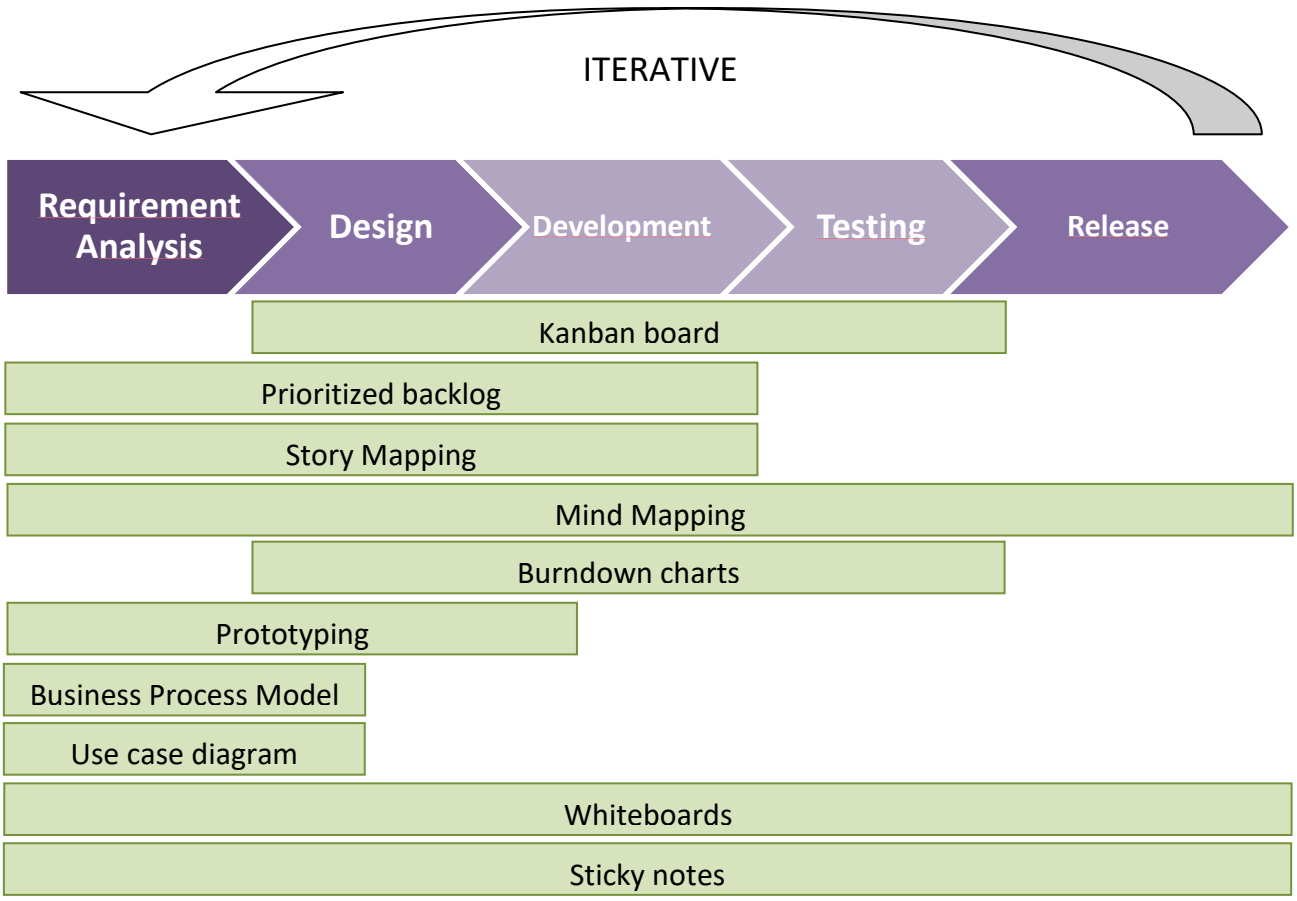

Figure 4: The use of Visual Thinking tools throughout the high level Agile Software Development Life Cycle (SDLC)

Source: Adapted from www.gridgit.com/post_agile-sdlc-phases-diagram_1413173

We are able to identify three types of tools from Figure 4. Firstly, we have the Mind Maps, whiteboards and sticky notes which are tools that are used during the whole project life cycle. A common characteristic to these tools is that they are extremely simple and easy to use, we will thus call them the fundamental Visual Thinking tools and discuss them further in Section 3.4. Furthermore, we can group the 'Kanban board' and 'Burndown chart' together to form the visual reporting tools group. These tools do not really provide support for the

actual development and testing phases, but help the agile team to plan those activities and report the progress continuously in a visual way. Finally, we can group the 'Prioritized backlog', the 'Story Map', the 'Prototyping', the 'Business Process Model' and the 'Use case diagram' as these are visualisation tools supporting the first phase of Requirement analyse. Although these tools are selected with a total score higher than 10, this last group remains relatively unknown in the informal literature with amount of occurrences of respectively 11, 2, 5, 7 for the 'Story Map', the 'Prototyping', the 'BPM' and the 'Use case diagram'.

The goal of the next sections is to overview these three groups of tools in detail. The first part will be dedicated to depicting each tool accurately while keeping an attentive eye on the values of the Agile Manifesto in general. The second part will then show and discuss to what extent each widely used visualisation tool addresses the underlying challenges related to the core agile principles depicted in Chapter 1.

3.4 Fundamental Visual Thinking tools

3.4.1 Mind mapping

Mind Mapping is a technique for mapping visually ideas which has been created by Tony Buzan, a British psychologist in the 1960s. It is today the most popular Visual Thinking tool in organisation and in the literature (Williams, 2015). Buzan was looking for a very simple and fast way to **visually organise and structure ideas on paper** in order to use both hemispheres of our brain (Buzan & Buzan, 2006) and stimulate creativity during meetings or conversation. The technique consists of starting with a central theme in the middle of the page. From that central concept, we find key words and concepts directly related to it, these are connected with lines to the central idea. Every idea can branch out in sub ideas when it triggers new associations. As a result, we get an organised page full of ideas that helps us to visualise various string of thoughts emerging from the central concept and all the relationships

between these (Zujewska, 2009). In addition, one could also add symbols and colours to highlight flagged key words and encourage creative associations. As an example, a sample of a Mind Map is illustrated in Figure 5.

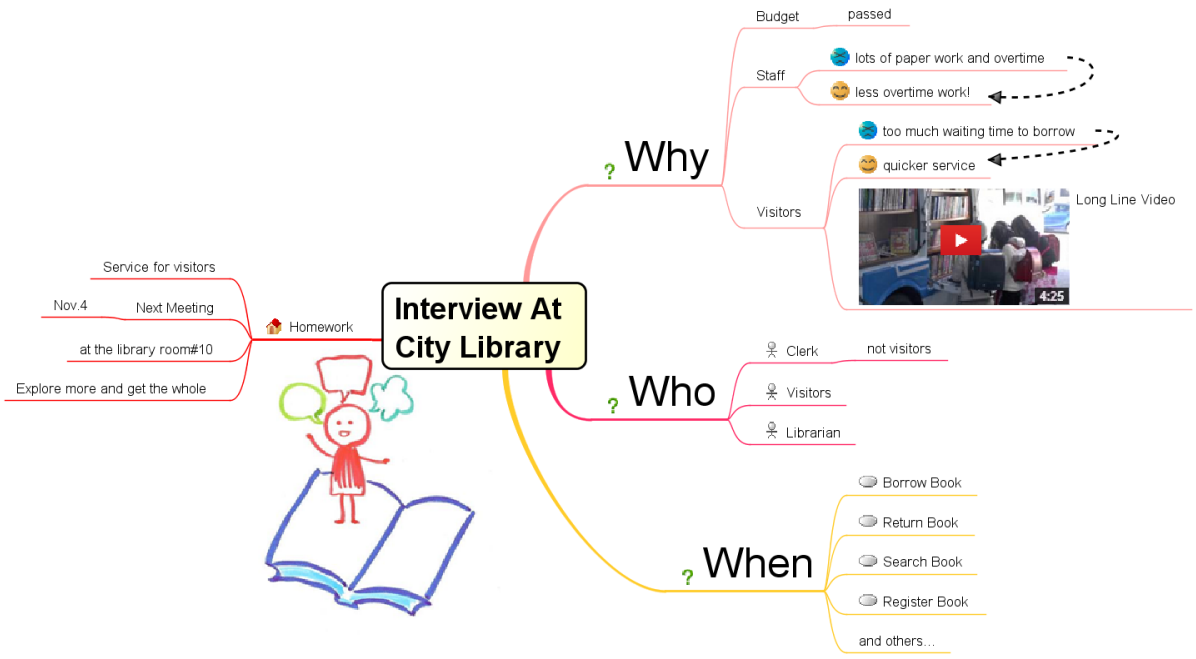

Figure 5: Sample of a Mind Map
Source: https://www.infoq.com/articles/kenji-modeling-agile

As we can see in Figure 5, mind mapping is not a tool which is exclusively developed for software development project, but can be used in almost **all kind of environments**. Thus, employing it within an agile team also makes perfect sense as the technique meets most of the agile values. Indeed, by stimulating interactions through visual and easy to understand key words and relationships, the tool improves customer collaboration and helps the whole team together with the customer to get the big picture of the project goals quite rapidly (Williams, 2015). Let us take for instance user requirements, Hiranabe proposes Mind Mapping to conduct a preliminary exploration of those. As no standard process exists for documenting user requirements, the primary success factor becomes smooth and effective communication, which is precisely offered with Mind mapping (Hiranabe, 2008). However,

Tony Buzan proposed a whole set of rules in his different books to create a mind map (Buzan & Buzan, 2006), but in our opinion, we do not need to strictly stick to all these rules. Indeed, an agile practitioner should remain free to set up any rule that fits the best to his team and project. This freedom and flexibility is precisely the core advantage of mind mapping in comparison with other visualisation tools and their rigid limitations (Williams, 2015).

3.4.2 Sticky notes and index cards

People usually wrongly say that doing agile is automatically working with post it notes. This association should not be generalised but we get close to it. Sticky notes are one of the major artefacts employed in an agile environment. This visual tool has a limited space in order to only write what matters on it. This short message is thus easier to understand and to share with the rest of the team and with the customer for collaboration. Using sticky notes during a meeting makes it more interactive, as everybody can see and move these small and tangible objects on the table (Leffingwell, 2011). This also stimulates group reflection and effective interactions while trying to combine and categorise your thoughts in a new visual way on the table. Doing this, color coding could also be used to differentiate or cluster the sticky notes. User stories for example are regularly written on sticky notes or index cards to encourage agilists to create clear and concise user stories. As such, the focus of writing user stories is not anymore about writing features, but rather discussing them (Cohn, 2010). Thanks to the movable cards, the user stories can be easily arranged by size or iteration to analyse the different relationships between them (Leffingwell, 2011). Finally, post-it notes are very easy to combine with other tools. We will see in the next parts that a lot of visual tools are based on this fundamental artefact. Some critics about this tool were made by Jackson (2015) who says that sticky notes are unfortunately temporal, after each iteration, they are removed from the boards and no history of the previous iterations is recorded. Moreover, he also believes that such a tangible item becomes useless when we

have distributed teams, but those distributed teams come along with a whole range of other challenges as well, which we will not address in this thesis.

3.4.3 Whiteboards and display walls

A last fundamental Visual Thinking tool is related to the environment in which an agile team is working. Indeed, it is important to set up a single room in which the entire project team can physically gather together to stimulate effective interactions and collaboration (Williams, 2015). The workspace should furthermore contain a flexible seating arrangement with a variety of big, visible charts and whiteboard on the walls (Cohn, 2010). As such, the room becomes itself a communication tool that provides visual support to any discussion in that room as everybody has easy access to the whiteboard or can simply point to some charts on the wall to illustrate his thoughts (Patton, 2014). Ron Jeffries (2004), one of the inventors of XP, suggests some charts to displays on XP project room walls, however it is crucial to repeat that we have to leave the choice to every agile team to decide which are the most significant tools to display given their own project environment.

The fundamental Visual Thinking tools depicted in this part have all three in common that they are low-fidelity techniques. This means that these are tools that people and agilists are used to employ regularly and thus do not require much technical expertise (Alexander & Maiden, 2004). They can focus on how to use it in the most effective way rather than on learning how the tool works. Indeed, sticky notes, mind maps and rooms with whiteboard and charts can be effortlessly used by almost anyone. Team can furthermore adapt these tools to their own processes and needs easily which is in our opinion one of the key reasons why these are so popular in agile project management and are used throughout the whole agile Software Development Life Cycle (see Section 3.3).

3.5 Visual reporting tools

Visual tools aiming at tracking the project progress are also important in an agile environment. We distinguish here two kind of reporting tools. Firstly the tools are used during the short iterations to give the team more visibility into the status of the work items. Daily stand-up meetings in front of big visualisation boards are set up to share information about status and if necessary adjust when deviations from the plan occur (Leffingwell, 2011). Secondly, higher-level status views are as well crucial for an agile team so that it can step back and see the big picture of where they stand in the whole project lifecycle.

3.5.1 Lean development & Kanban boards

Lean software development has been first brought up by Charette (2002) and Poppendieck and Poppendieck (2003). They presented it as a translation to the software development domain of the lean manufacturing principles developed to reduce the seven kinds of waste listed by Poppendieck and Poppendieck (2003) (i.e. overproduction, transportation, motion, waiting, overprocessing, inventory and defects). The idea behind lean is to deliver at any time the highest possible value to the customer with the minimum cost, time and effort. (Brechner, 2015). As a result, any activity that does not create value for the customer is considered as waste. In order to achieve this, Poppendieck and Poppendieck (2003) summarized seven different principles to apply during the software development process (i.e. eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build integrity in, see the whole). Similarities with the initial agile principles of the Manifesto (Beck et al., 2001) can be clearly noticed.

The Kanban system is an approach inspired from this lean development to organise and manage visually intangible software work. The *Kanban* word means in Japanese "visual signal" or card. The approach behind the word provides thus, via a Kanban board, a view of the workflow of the project by presenting the progress of work items that need to be

produced, what is in production and how much to produce. This is visible through the whole system, from task definition when items enter it, till customer delivery when they leave it (Leffingwell, 2011). Mike Burrows (2014) identified the four foundational principles that every leader should adopt when employing Kanban:

- Start with what you know
- Respect the current processes, roles, responsibilities and job titles
- Agree to pursue improvement through incremental, evolutionary change
- Encourage acts of leadership at all levels

David Anderson, considered to be the founder of Kanban applied to knowledge work and IT, identified five core properties on teams who used the Kanban approach (Anderson, 2010).

- Visualize the workflow
- Limit the WIP
- Manage flow
- Make process policies explicit
- Improve collaboratively, evolve experimentally

Visual thinkers often have a hard time with traditional task management. A Kanban system allows them to visualize this workflow of items which usually are user stories, features or other requirements. As such, a kanban board "differs from a task board, which generally focuses on visualizing the current tasks" (Leffingwell, 2011). This visualisation supports the agilist who needs to monitor and manage the flow of work items. Next, by limiting the WIP, Kanban prevents you from starting more unit of values than you actually can implement (Brechner, 2015). The WIP limit also helps you in the decision-making process about when there is capacity to start a new unit of value. This kind of pull system let you visualise quickly any bottleneck in the workflow and provide information on how to change processes to improve them continuously (Leffingwell, 2011).

A question remains, does a Kanban board which is used to implement this Kanban system, meet the fundamental agile values of the Agile Manifesto (Beck et al., 2001)? We have seen

that the lean principles mostly agree with the Manifesto, let us discuss whether this statement can be extended to the Kanban board. Firstly, from our experience in an agile organisation, daily meetings were organised in front of a Kanban board to report progress of every member of the team. This meeting stimulates interactions about upcoming issues while moving the post it notes on the board. As an updated picture of the work is transparently showed on the wall of the project room, it makes it easier to collaborate with the customer because as soon as he enters the room, he has a shared understanding of the workflow and process. The foundational principle to pursue continuous improvement through incremental change is in accordance with the fourth value of the Manifesto (i.e. *responding to change over following a plan*; Beck et al., 2001) and encourages a team that is using a Kanban system to embrace changes in requirements and even consider them as competitive advantages (Poppendieck & Poppendieck, 2003). However, the second foundational principle promoting to respect processes and job titles is considered, according to Larman, to be a common mistake in an agile environment (Larman, 2008). This mistake could be a residual of the lean manufacturing that considers people as machine parts in which we try to optimize the chain regarding keeping people busy instead of looking at people and their creativity (Larman, 2012). The kicker in a rugby team who takes the ball forward is a great illustration of the thinking mistake of local optimisation instead of system optimization (Brechner, 2015). Next, the Kanban approach will encounter difficulties when the amount of work items becomes substantial, the team member will lose the big picture of the progress. Finally, Bradley also highlighted the lack of built in feedback loops in a Kanban system (Bradley, 2013). Although many authors (Poppendieck & Poppendieck, 2003; Leffingwell, 2011; Brechner, 2015) describe Kanban to be a light-weight method using an adaptive and experimental approach to software development, there is no clear loop supporting constant questioning in an empirical process if only through generated discussion in front of the Kanban board.

3.5.2 Burndown charts

Before discussing the burndown charts, we firstly need to very briefly define the concept of velocity. According to Cohn (2010) this is a measure of how much work a team can complete in a given period of time, which is usually a sprint or a day. The velocity measurement is based on the velocity in previous sprint and will thus vary from sprint to sprint due to different factors such as the team size or a new team that never worked together before (Cohn, 2010). Velocity is however only an estimation and should be continuously adjusted while the project progresses. A useful tool that makes you visualize the actual and the projected velocity is the burndown chart. It is a powerful tool for visualizing progress by showing the amount of completed and remaining work in function of the allowed time (Cobb, 2015). A burndown chart can have different scopes, it can show the progress of the whole project, the release or the sprint. As a picture is worth a thousand words, we can find in Appendix 2 an illustration of a burndown chart. The chart starts with the full amount of work to do and then slowly decreases during the project till every work item is done at the end of the time period.

This Visual Thinking tool is mostly in accordance with the values of the Agile Manifesto. However, the biggest issue of the burndown charts according to Allen Holub (2014) is the fact that they are based on estimations. Estimations are always based on guess work and will therefore always be wrong, no matter how much continuous adjusting work has been done before. Estimation work has thus never created any value for the customer and calculating them will remain consequently a complete waste of time according to Holub. However in our opinion, being so definite and conclusive, makes us lose sight of the fact that burndown chart still provides the agile team a strong visual track of the current state of the project and how quickly the team is 'burning' through the work items.

3.6 Visual Thinking tools supporting Requirement Engineering

3.6.1 Modelling in an agile environment

Modelling is a core practice for visualising requirements, information and the final software during an agile project. Models are fundamental when building complex software, given that these are impossible to understand in their entirety. Some visualisation tools could thus be useful to help a development team to capture the system requirements and architecture and communicate it unambiguously to the whole team (Abrahamsson, Salo and Ronkainen, 2002). In other words, visualising the large amount of requirements and the complex product as a range of simple models is crucial to share the same big picture in mind (Patton, 2014). However, modelling has never been a value or principle of the Agile Manifesto. Indeed, the agile principles rather support the text based requirement artefacts such as User stories and thus hamper the development of visual modelling tools in requirement engineering. Moreover, a major issue of modelling is that it is a very time consuming activity, which goes straight against some basic agile principles of responding quickly and effectively to changes in requirement. Indeed, excessive modelling will slow down the development speed considerable. Ambler proposes light-weight modelling techniques in the context of agile development to find the balance "where you have modelled enough to explore and document the system effectively but not so much that it becomes a burden that slows the project down" (Ambler, 2002). If a team focuses too much on modelling, it would be at the expense of bringing the whole team together to develop a working and useful software. Consequently, models need to be part of any agile development to better understand and visualise the problem, but agile practitioners have to keep an attentive eye on the software development itself at any time to still deliver a valuable software to the customer, as this remains after all the main objective of any IT project (Ambler, 2002).

3.6.2 Visualising requirements through models

A major shift in the agile environment is the attitude towards requirements. While requirement engineering activities were done upfront in traditional software development projects, they take place continuously along an agile project. The activities involved in requirements engineering vary widely depending on the project and the organisation, however, according to Sommerville (2007), there are some generic requirement activities common to a lot of processes. The first activity is requirement elicitation which refers to the practice of collecting, identifying and understanding the needs and requirements of the customer (Leffingwell, 2011). Then comes the requirement analysis during which the requirements are analysed and prioritised in order to define solutions that meet the needs of the customer. Third, the requirements are validated with each small and frequent release at the end of the short iterations. Last but not least, the changes in requirements are managed by adjusting the solutions at every iteration when feedback is received by the customer (Leffingwell, 2011). Given that effective requirement engineering is one of the most frequent issues along the development lifecycle, agile practitioners use visual models to improve the visualisation and understanding of those requirement engineering activities.

In order to analyse how visual practices can solve requirement issues, we firstly start to overview briefly the major requirement elicitation challenges faced by agile teams. Rajagopal classified all these challenges as the following (Rajagopal et al., 2005):

- **Problems of scope**: Every IT project should start by determining the boundaries of the software to build, so that the team does not get confused with useless details out of scope.
- **Problems of understanding**: The majority of the IT projects do not fail because of a technological issue, but rather because of a communication breakdown (Fowler & Ford, 2013). This is particularly true when defining the customer's requirements. Analysts and customers usually come from different domains, this means that they speak different languages or give different meanings to the same words. This issue

leads to misunderstandings and ambiguous requirements. Moreover, users themselves frequently do not know exactly what they want until they have something to see, or different users come up with conflicting requirements. Finally, developers usually do not even see the customers and are thus too focused on the technical aspect of the project. Figure 3 illustrates this problem well.

- **Problems of volatility**: Given the fact that customers may change their mind or that their needs change, requirements will for sure evolve over time. As already stated at the beginning of this part, requirement engineering in an agile environment is executed continuously so that the team can take the increased knowledge learned throughout the project into account.

Ramesh also identified in an empirical study a list of challenges created by the use of common requirement engineering practices (Ramesh et al., 2007): cost and schedule estimation, inadequate architecture, customer access and participation, minimum documentation, lack of requirement verification, prioritize a single dimension and neglecting non-functionals (see Appendix 3).

3.6.3 Three main Requirement Engineering approaches: Use cases, User Stories and Goal Oriented Requirement Engineering

Use cases are a first widely used approach for expressing the features of a system to build (Cohn, 2010). Cohn describes them as artefacts that help to capture the interactions between an actor which can be a human or any other external system, and the system, to achieve a certain goal. User stories are a second main requirement artefacts used for agile software development. Various definitions exist in the literature to describe User Stories. According to Cohn (2010), a User Story "is a short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system". Cohn also discusses how User Stories differ from other requirement methods

like the use case depicted here above. They differ in the level of completeness, in their much shorter scope they cover as well as in their longevity. Besides, User Stories and use case are written with different purposes in mind. While the User Stories serve for releases, iteration planning and conversations about the details requirements of the users, use case are rather written in an aim to generate discussion and agree on the features (Cohn, 2004). We will not spend more time defining the artefact as this would be a huge subject. However, a usual challenge that comes along with the artefact is the size. Indeed, a large User Story is called an epic and needs to be broken down into the smallest possible parts so that feature can be delivered quickly as well as the feedback of it (Patton, 2014). As User Stories are created to generate discussions instead of writing about features, talking about an epic is the best way for breaking down these big separable User Stories. Moreover, Cohn (2010) also describes themes as "a collection of logically related US (...) that is not decomposed into enough detail yet", but are not useful to group together (Patton, 2014). To better understand these three concepts, an illustration of it is shown in Appendix 4.

Lastly, we have goal-oriented requirement engineering (GORE) which is mostly known in the research community. This agent-oriented approach differs from the two traditional artefacts explained here above. It uses goals which have been recognized to play a crucial role for requirement elicitation, analysis, validation and management (van Lamsweerde, 2004). By focusing on the 'Why' questions, GORE can easily improve the completeness of the requirements by analysing alternative ways for achieving a goal. This approach provides thus a richer context for understanding requirements (Ye et al., 2011). The Rationale Diagram discussed in Section 3.6.9 is based on this approach.

3.6.4 Visualising requirements through use case diagrams

Various kinds of diagrams are used in the literature with the purpose of visualising requirements. These allow the team members to see the requirements from different angles and thus indentify missing requirements and provide a complete view of the system to

develop. A first angle of visualisation, with 6 formal and 7 informal sources talking about them, is the use case diagram that models the various use cases to show the associations and relationships between them and the different actors of the system. There are three types of relationships: inheritance, extends and includes. As shown in Figure 6, the use case diagram is very simple to understand and is thus an effective communication tool that can be used by the whole agile team as well as the customers to build a shared vision of the software. Siau and Lee (2004) examined the value of such a diagram for interpreting requirements and concluded that use case diagrams represent the system-to-be in a complete and very simplified manner. To illustrate this visualisation tool, an example of a use case diagram is shown in Figure 6.

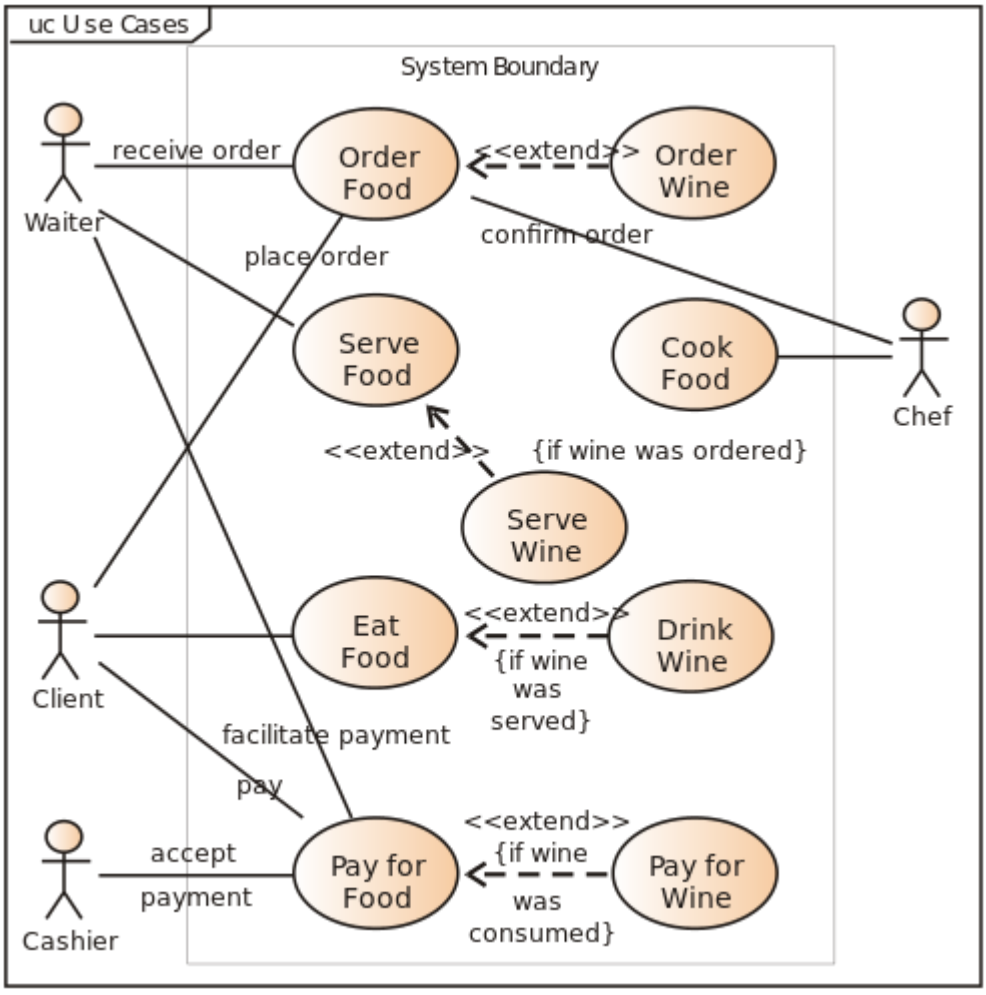

Figure 6: Sample of a UML use case diagram

Source: *https://commons.wikimedia.org/w/index.php?curid=7880320*

3.6.5 Visualising requirements through other diagrams

Another angle to visualise the developed software is through the **persona diagram** (see Appendix 5). If a system makes interactions with a lot of users, a team may need support to build a shared understanding of all these users. A persona diagram describes from your software point of view who are these users, their characteristics, goals and what they expect from the software. Building these will help the team "look at the software through the eyes of the users" (Patton, 2014). If an agile team does not know exactly who are the users of the system, they will create long unstructured wish lists as requirements instead of relevant User Stories. The challenge here is thus to create those User Stories from your persona diagram. Pichler (2014) proposes an approach for this. As we can see on Figure 7, when the persona diagram is built, the goals of the personas can be used to identify the key product features, in other words the epics. These can then be decomposed progressively in User Stories.

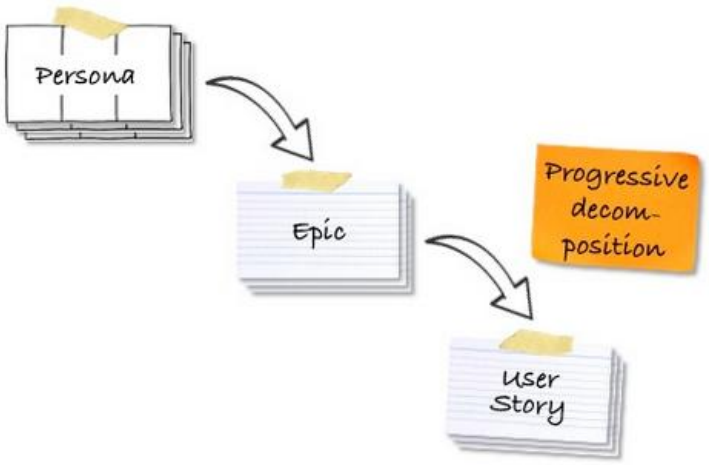

Figure 8: Deriving User Stories from Persona diagrams
Source: Pichler (2014)

A last perspective on the system is the **User Interface (UI) flow diagram**. When a software has a lot of interrelated user interfaces, team members usually get lost in the large flow of different screen options. Besides, according to Leffingwell (2011), a core challenge resides in incorporating the visual design into the short iteration to get rapid feedback on it. However, this diagram is more related to prototyping and sketching, which will be discussed in Section 3.6.8.

3.6.6 Visualising requirements through the BPMN

If your project is very process oriented, Business Process Model and Notation (BPMN) might be a visual process modelling technique that helps you to get a clear picture of the organisation. The non-profit organisation Business Process Management Initiative developed the BPMN in 2005 to create a single unified notation for business process modelling comprising the underlying values of the wide variety of methodologies at that time (White, 2008). The idea behind the BPMN is to provide an intuitive standard notation understandable by both the business and the technical users. White (2008) describes in his book the technique as an aid to communication that acts as a common language to share common understanding about the flow of business activities in an organisation as well as the role of the various business actors performing these activities, in order to improve them. The BPMN is thus very similar to the activity diagram from the Unified Modelling Language (UML) except the fact that BPMN is process-oriented while UML is rather object-oriented. However, the BPMN does not explain how to capture and design these business process models and leaves the choice of the methodology to the organisations (White, 2008). A sample of a Business Process Model is illustrated on Figure 8. We will not discuss the notation itself with its set of graphical elements, but rather focus on how this visual notation can help to address the challenges of requirement engineering in an agile environment.

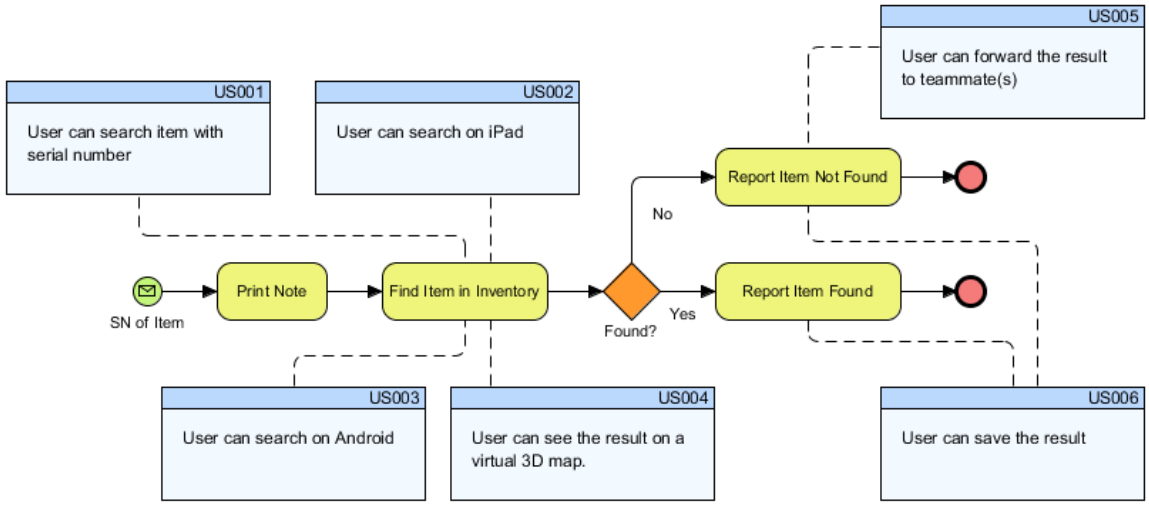

Figure 8: Sample of a Business Process Model visualising User Stories
Source: https://www.visual-paradigm.com/tutorials/business-process-to-user-stories-mapping.jsp

3.6.7 Visualising requirements through User Story Mapping

Given the different granularity levels of the User Stories (Liskin et al., 2014), we could understand that a visualisation tool is needed to get a shared big picture of all these small pieces of requirements. As an example of lack of vision, a common mistake in discussions is when the team is still talking on the theme level, the conversation is usually going deeper into the details way too early. If the team forgets to go wide before going into the details, they will lose the big picture (Svarytsevych, 2015) but also miss crucial requirements.

A traditional way to visualise this set of User Stories is by writing them down on sticky notes or index cards (see Section 3.4.1) and arrange them on big flipcharts on the wall. However, this becomes messy when the amount of User Stories is getting larger, the team as well as the customer will lose vision of the large product backlog even if this latter is prioritised such that the items creating the most value to the organisation are the highest (Cohn, 2010). Other visualisation tools are thus needed to keep the big picture in mind and structure the User Stories besides only prioritising them in a list (see Appendix 6).

One of these tools is the **User Story Mapping**. It is a very simple tool that helps organising and structuring up to hundreds of User Stories in order to get a shared vision on what the team is building. The approach has been proposed by Jeff Patton (2014) and consists in organising the User Stories spatially in two different ways along two axes. The horizontal axis arranges the User Stories in the same order as when you explain to someone what people do with the system. The vertical axis arranges the User Stories according to the priorities and level of abstraction of the User Stories. Besides this, the post it notes with the highest level of abstraction (i.e. on the top of the map) are called the *user activities* (in orange on Figure 9*)*, which refers to the list of activities done by the user with relevant characteristics to the software to develop. The layer under these user activities contains the *user tasks* (in blue on Figure 9) across the full span of the user experience, and finally under these latter, we find the lowest level of abstraction with the User Stories. This User Stories are divided vertically in different releases in order to plan visually and holistically the releases throughout the development lifecycle. A template of a User Story Map is showed in Figure 9.

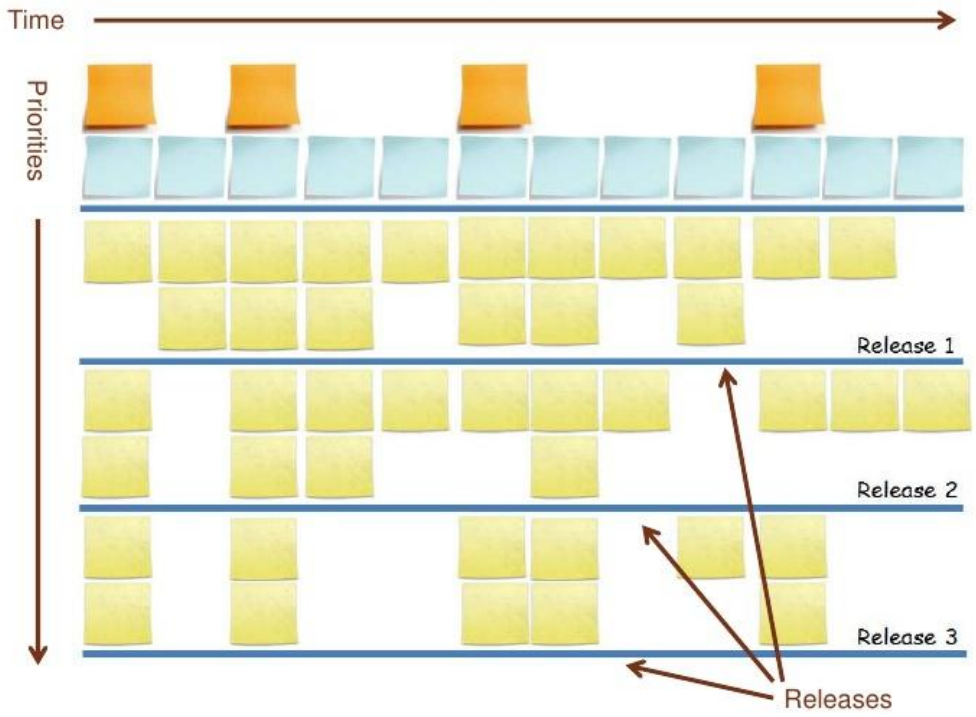

Figure 9: User Story Map
Source: Rogalsky (2011)

As we can see, the User Story Map is actually a product backlog which is spatially organised in a logical manner to envision effectively the big picture and help better understanding the different functionalities of the system as well as missing requirements in the backlog. From the advantages discussed by Patton (2014), we can enumerate a few strengths concerning the visual power of this tool in an agile environment:

- The User Story map allows you to take a step back and see the entire big picture of the software that the team is developing;
- It provides context for the User Stories and shows how larger User Stories are divided in smaller ones;
- It helps you to check visually the completeness of the product backlog;
- It identifies visually the next User Stories to build and shows in a logical manner what has been done and what is left to do;
- As a result from the last point, the User Story map also acts as a visual reporting tool that shows the overall progress of the agile team;
- It is also a roadmap that helps you to plan spatially when are going to be the releases;
- It supports the primary intent of User Stories: generating team discussion with visual support.

3.6.8 Visualising requirements and software through Prototyping and Sketching

Prototyping is the Visual Thinking tool that has been mostly mentioned by formal sources (i.e. 7) and only two times informally by agile practitioners. This means that the tool has already been analysed several times by agile experts but remains quite unknown in practice, however, the tool is very effective to overcome the challenges of presenting specifications to the users and customers (Alexander & Maiden, 2004). Indeed, we already discussed at

length the power of pictures and images in comparison with formal, text-based specifications. Prototyping and sketching your software-to-be help to visualize clearly the user interface in order to create a shared understanding of what needs to be developed as well as the effects of the requirements, rather than trying to use textual descriptions (Patton, 2014). Besides, this practice will generate short feedback loops by proposing simple mock ups to the users and work easily together with them to improve these before the actual development work (Leffingwell, 2011).

Different types of prototypes can be created. The first type is the so called mock-up or throwaway prototype which plays a crucial role in the requirement elicitation. This means that a simple reduced sketch of the software is built to explore and experiment with different solutions that might respond to the requirements of the users. This type helps understanding and clarifying some features as well as capturing the missing ones (van Lamsweerde, 2009). It is also this type that generates the early feedback in the process. More advanced prototypes with more details can be created afterwards to firstly evaluate if the software truly addresses the requirements (Patton, 2014). Next, we can have a functional prototype that focuses on a few complex functionalities of the software for the purpose of validating the requirements related to these functionalities (van Lamsweerde, 2009). Last but not least, van Lamsweerde describes the user interface prototype that concentrates on all the aspects of the user-software interactions such as for instance data input and output displays.

However, van Lamsweerde (2009) identifies a few drawbacks of prototypes in his book. By definition, analysts leave some functionalities aside when sketching a software product. A prototype can thus also be misleading as it does not cover all the aspects of the final software and will always provide a truncated picture of it. This drawback goes straight against the agile value of a *working software* (Beck et al., 2001), the agile team will spend a lot of time designing a prototype that will never work instead of focusing on the development of the working software. Furthermore, we need to build those prototypes in a

quick way, because if we spend too much time creating those, we will lose too much time for the valuable development which is the real value adding activity. However, going too quickly will be associated with 'quick and dirty' work, which is useless. The right balance needs thus to be found between building prototypes quickly without going too much into the details, and providing a vision of the software which is useful for the team and customer (van Lamsweerde, 2009).

3.6.9 Visualising requirements through the Rationale Diagram

An additional visualisation tool seems to be interesting to discuss in this section. If we actually look at all the widely used tools discussed in this chapter, they all serve as visual support for generating or validating requirement artefacts, or as a complementary view for discovering missing requirements. The Rationale Diagram (RD) proposed by Wautelet et al. (2016b) has an opposite purpose, it produces a graphical model from an existing User Story set in order to visualise and analyse these User Stories and their dependencies. This RD is based on a previous research of Wautelet et al. (2014), the Unified User Story Model which is inspired from the i* framework (Yu et al., 2011).

Following Wautelet et al., the need for such a model is twofold. Firstly, Liskin et al.(2014) identified the importance to reflect on granularity while modelling User Stories. Secondly, Larman & Vodde (2008) widely discussed the issue of poor scalability due to ineffective requirement representation. Departing from these two needs, agile practitioners can use this model to visually represent User Stories by grouping them around themes according to their links. The RD might look similar to the User Story Mapping depicted in Section 3.6.6, however it diverges from it in two ways. The RD introduces formal links between the User Stories, which is absent or vaguely represented with color codes in the User Story Mapping. Besides, the structure of the RD is much finer than the limited columns of the USM. The RD also stands apart from the UML transformation proposed by Wautelet et al. (2016a), which

manages to bring a high level view on the software to build, but fails to provide a decomposition approach.

This last point is precisely one of the main strengths of the RD. It helps an agile team to envision the big picture of the project by organising and grouping the User Stories on a macro-level, while in the mean team proposing an accurate decomposition approach on a micro level based on the Unified model. Following the researchers, this double approach enriches the information about the dependencies of the User Stories and also permits to eliminate the redundant items in the requirements. Moreover, getting an accurate overview of the links between the changing User Stories throughout the project helps the agile team to respond quickly to changes in requirements, as encouraged in the Agile Manifesto (Beck et al., 2001). Finally, the RD plays a more helpful role in the prioritization of the User Stories and the iterative planning in comparison with other techniques that only show themes that need to be treated in multiple iterations.

However, these advantages do have a cost. In order to be able to build a RD, the agile modeller needs to know some pieces of information about the granularity and the changing links between the User Stories. As a result, an accurate tagging work is needed upfront which could be time consuming. Wautelet et al. (2016b) nevertheless claim that this time is marginal compared to the benefits of the approach. Another little drawback is the complexity of the tool which could be harmful to the core simplicity value supported by various agile founders and practitioners (Beck et al., 2001; Lindstrom & Jeffries, 2004; Shiralige, 2012). However, this last challenge could be mitigated with the use of the CASE-tool that supports the construction and maintenance of the graphical model and partly automates it.

3.7 Visual Thinking tools and the existing challenges within the agile environment

Before discussing Table 2, one comment needs to be made about the scope and validity of it. Table 2 has to be read with caution, it is unfortunately not the result of a quantitative analysis. Such an analysis would have been difficult to conduct given that only a few busy agile practitioners know and use regularly the 10 Visual Thinking tools depicted in this chapter. Collecting a sufficient amount of valid responses during the quantitative phase would have been very challenging in such a short period of time. As a result, Table 2 has been completed carefully based on the knowledge acquired during the formal and informal literature overview preceding the writing of this chapter. Further quantitative research could be conducted to confirm this table.

When we look at Table 2, we can firstly see that the widely used Visual Thinking tools globally address most of the agile challenges. This is not really a surprise given the analyse made in Chapter 2. Apart from the issue of "ensuring job safety rather than role safety", every challenge has been addressed by at least one visual tool. This unaddressed challenge could rather be addressed by managerial practices instead of Visual Thinking tools. The same for the challenge of "having an organisational culture that supports agility" that can be addressed by a more top down approach in an organisation. Other challenges that are poorly addressed by Visual Thinking tools include "showing the software to the customer after each iteration" and "Including the customer for the ongoing planning" (highlighted in red in Table 3). Surprisingly, while Visual Thinking as a concept supports effectively customer collaboration, not many tools concretely help agile teams to include the customer on a daily basis in the project.

On the other hand, some challenges such as "Supporting interactions between individuals to avoid communication breakdown", "Writing only simple and useful documentation that will not undermine the value creation for the software" and "*R*esponding quickly and effectively to changes in requirements" (highlighted in green in Table 3) are effectively mitigated by the

majority of the Visual Thinking tools. This seems to be logical as Visual Thinking produces by definition simple, intuitive tools that help the agile practitioners to envision de requirements and solutions more quickly than long text-written documents.

As we analyse Table 4, we firstly notice that no widely used Visual Thinking tool is useless in terms of addressing core agile challenges. Indeed, they all play a certain mitigating role in at least 6 out of the 15 identified challenges. Furthermore, all amounts of challenges addressed by the tools are within a range of 6 to 10, which means that a combination of various tools needs to be chosen in an agile project to address as many challenges as possible given that no tool is stepping of line.

Table 2: Overview of the Visual Thinking tools addressing the challenges related to the core agile values

	Fundamental VT tools			Visual reporting tools		VT tools supporting RE				
	Mind Map	Sticky Notes	White board	Kanban board	Burndown ch.	Use Case D.	BPMN	US Map	Prototyping	RD
INDIVIDUALS AND INTERACTIONS										
Avoid communication breakdown	V	/	V	V	V	V	V	V	V	V
Processes supporting highly variable individuals	V	V	V	V	/	V	X	/	V	/
WORKING SOFTWARE										
Simple documentation that creates value	V	V	V	V	V	V	V	V	X	X - V
Whole team commitment to deliver working software after each iteration	/	/	/	V	V	/	/	/	/	V
CUSTOMER COLLABORATION										
Customer part of project on daily basis	/	/	V	V	/	/	/	/	V	/
Showing software after each iteration	/	/	/	/	/	/	/	/	V	/
Including for ongoing planning	/	/	/	/	/	/	/	V	/	V
RESPONDING TO CHANGES										
Responding quickly	V	V	V	V	V	V	V	V	V	V
Change own processes	V	V	V	X	/	/	/	/	/	/
EMPIRICAL PROCESS										
Promoting observation and experimentation	/	V	V	V	/	V	V	/	V	V
Self-organising teams ajdusting own tools	V	V	V	V	V	V	/	V	/	/
Short feedback loops	/	/	V	/	V	V	V	/	V	V
WHOLE ORGANISATION										
Organisational culture	/	/	V	/	/	/	/	/	/	/
Jobs safety > role safety	/	/	/	X	/	/	/	/	/	/
System optimization > local optimization	/	/	/	X	V	/	V	V	/	V

***Legend**:*

V : Addressing the challenge; / : Not addressing the challenge; X : Exacerbating the challenge

INDIVIDUALS AND INTERACTIONS	# VT tools addressing the challenge
Avoid communication breakdown	9
Processes supporting highly variable individuals	6
WORKING SOFTWARE	
Simple documentation that creates value	8
Whole team commitment to deliver working software after each iteration	3
CUSTOMER COLLABORATION	
Customer part of project on daily basis	3
Showing software after each iteration	1
Including for ongoing planning	2
RESPONDING TO CHANGES	
Responding quickly	10
Change own processes	3
EMPIRICAL PROCESS	
Promoting observation and experimentation	7
Self-organising teams ajdusting own tools	7
Short feedback loops	6
WHOLE ORGANISATION	
Organisational culture	1
Jobs safety > role safety	0
System optimization > local optimization	4

Table 3: Amount of tools addressing each challenge

A final observation could be made about the fact that almost no Visual Thinking tool is exacerbating a challenge. Only the Kanban board is undermining a few challenges because of residuals of the lean manufacturing that still too much sees team members as machine parts on a manufacture chain, and furthermore the second foundational principle identified by Mike Burrows (2014): "*Respect* the current processes, roles, responsibilities and job titles". According to Burrows, this principle aims at eliminating initial fears in a team, but goes straight against the challenge of promoting job safety over role safety (Larman, 2008).

	Fundamental VT tools			Visual reporting tools	
	Mind Map	Sticky Notes	White board	Kanban board	Burndown ch.
# Challenges addressed	6	6	10	8	7
	VT tools supporting RE				
	Diagrams	BPMN	US Map	Prototyping	RD
# Challenges addressed	7	6	6	7	7

Table 4: Amount of challenges addressed by each tool

As a conclusion, we could say that Visual Thinking has taken a rightful important place in the agile environment, mainly through a dozens of effective Visual Thinking tools that all can play a crucial role in mitigating the key challenges faced by every agile team. The point is thus to choose the right tools that best fit the team and project, as well as continuously adapt and improve these in order to address the risks that threaten the agile software development.

68.

Chapter 4: A Case Study of Visual Thinking in an agile project

In the first three chapters, we mainly focus on the theoretical part of Visual Thinking in the agile environment. The objective of this fourth chapter is to show how some of these tools are used in practice. Doing so, we will firstly build a fictional case study which is based on the real life projects faced during my short experience as business analyst intern in an IT consultancy firm using agile methodologies. We will then focus on the illustrated BPM which is a tool used by my firm to build User Story sets during the requirement engineering phase. Finally, we will apply the Rationale Diagram proposed by Wautelet et al. (2016b) on the User Story set.

4.1 Context of the Case Study

Let us assume a fictive fast growing event management company, *EasEvent*. Suppose the company has grown very quickly in the past few years and needs a new information system. Until now, they were able to manage the organisation of their events with excel sheets and email as main communication tool, but with their growing number of events, it will not be possible anymore in the near future. So concretely, the main objective of the project is to capture the acquired experience over the last years in a software in order to support the smart dispatching and organisation of the different tasks to prepare an event.

The consultancy firm where I did my internship is used to work according to a methodology derived from Scrum. What makes their approach interesting is that they include several Visual Thinking tools such as Mind Maps, sticky notes, whiteboards, a prioritized backlog, a Kanban board and finally the illustrated BPMN into their Scrum workflow. We will discuss in the next section how effectively we manage to include the illustrated BPMN in the Scrum workflow of the EasEvent project.

4.2 Approach of the illustrated Business Process Model

EasEvent is a process oriented organisation. For such an organisation, a Business Process Model is an effective visual process modelling tool to help us sharing a common understanding of the various processes and actors present in EasEvent and build clear requirements for the software development. The first step of the project is the requirement elicitation and consists thus in building a so called 'illustrated Business Process Model' together with the client. We organised several workshops for the customer in which we tried to represent their business processes visually with post its on a whiteboard. These workshops led to a shared understanding of the big picture of EasEvent and served to build the illustrated Business Process Model of the organisation.

But what exactly is this Visual Thinking tool? The illustrated BPMN is a visualisation tool derived from the standard BPMN to better fit the team and projects like EasEvent of the consultancy firm. The idea behind the illustrated BPMN is pretty simple: having a visual support that illustrates a process-oriented organisation and that is understandable by anyone with no need to learn any notation. Indeed, even if the traditional BPMN is designed to be very intuitive (White, 2008), someone who has never seen any process modelling before will not be able to read and use the model. However, this lack of knowledge about any notation is usually the case of the customers who may have a wide variety of different expertise domains. For the EasEvent project, the customers in contact with us come from operations, HR and coordination departments and are not likely to know the standard Business Process Model Notation. As a result, the need of a completely intuitive tool is crucial to continuously collaborate with the various customers.

Concretely, the processes of EasEvent as well as the actors who carry out these processes are spatially structured in a very similar way as the traditional BPMN. However, the notation is substituted by intuitive icons. Any user of this illustrated BPMN, regardless his expertise, will thus immediately get an overall picture of the processes without even reading any single word. The complete intuitiveness and picturing has however a price. The illustrated BPMN

provides unfortunately not as much information on the processes as its traditional version and given that no tool has been developed yet to facilitate the creation of the illustrated BPMN, its construction might take us slightly more time. In addition, besides being a visual requirement elicitation tool, the tool visually supports us during collaborative discussions with people of EasEvent throughout the whole project. When all the information and requirements are collected, we then create the illustrated BPM of EasEvent as shown on Figure 9.

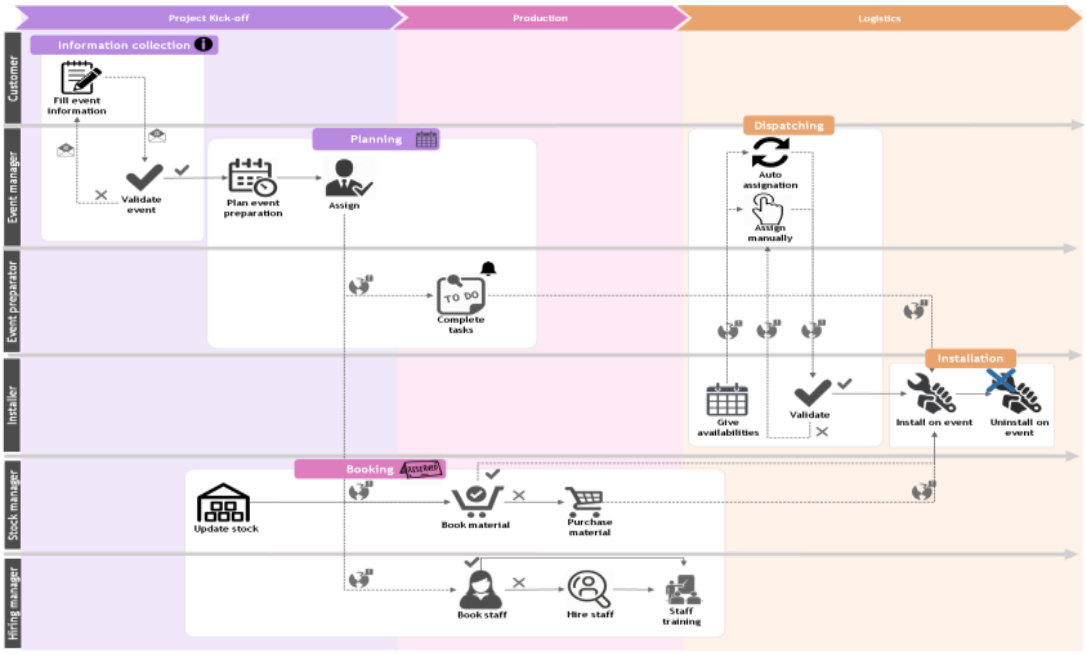

Figure 9: Sample of an illustrate BPMN
Source: Fictional project
Go to Appendix 7 for a large size version

When the illustrated BPM is built, we can then derive the key software features to develop, also called the high-level epics. The epics correspond to the white rectangles on Figure 9 and are identified as the following for EasEvent:

- Information collection
- Planning
- Booking

- Dispatching
- Installation

As shown on Figure 10 for the 'Information collection' epic, we can then progressively start decomposing these epics in smaller User Stories in order to create the product backlog and finally prioritise the User Stories. The non prioritized product backlog of the EasEvent project is illustrated in Appendix 8. Note firstly that the User Stories are grouped by themes (Set up, Roles management, Information collection, Event planning and Dispatching). Secondly, in the product backlog, we find User Stories with the classic template: *As* [the WHO], *I want* [the WHAT], *so that* [the WHY], but also technical spikes which are non functional features for the software. Finally, we have a last column with a 'How to demo' which is a precise description of the US from the perspective of the user. This "How to demo" forces every member of our team to agree on what the User Story is about and know in advance what will be demonstrated to the customer during the demo at the end of the sprint. The prioritized backlog works well for EasEvent as it is a project with a limited amount of User Stories, however when the amount of User Stories grows, the prioritized product backlog lacks visualisation and becomes ineffective.

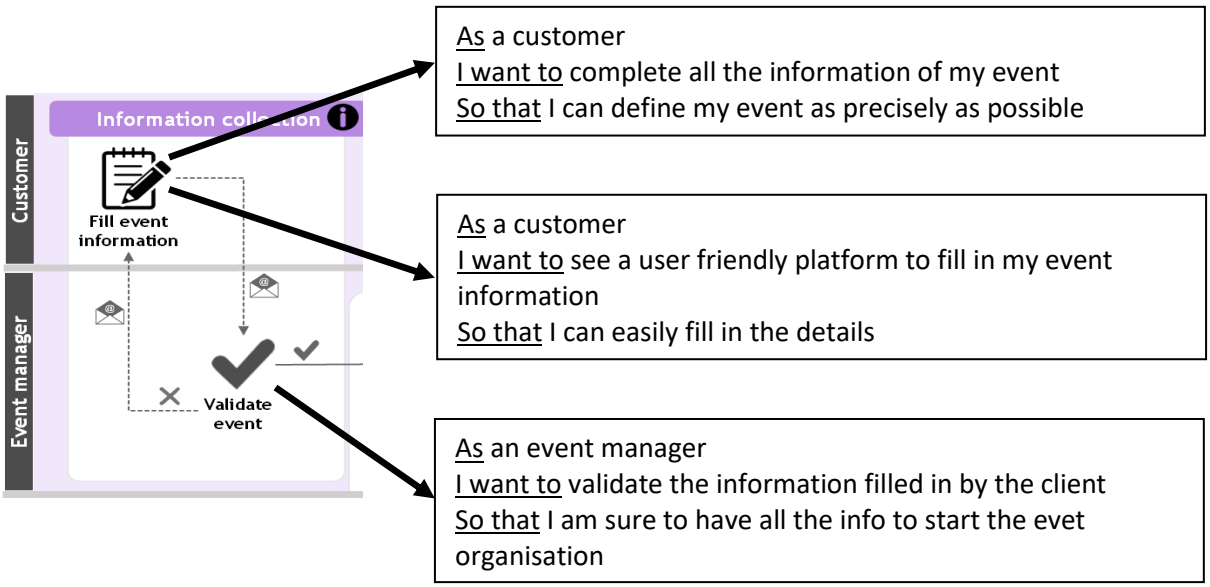

Figure 10: Decomposition in US of the 'Information collection' epic
Source: Fictional project

4.3 Approach of the Rationale Diagram

The process decomposition approach of the illustrated BPMN is very effective to visualise intuitively a complex process oriented organisation such as EasEvent. However, we face some challenges related to this approach. Firstly, we are able to group the User Stories around themes according to the processes, but the link between those remains quite vague. Moreover, the multiple granularity levels of the created User Stories are not clearly represented and could thus lead to missing and fuzzy requirements. Finally, this first approach also fails to provide a decomposition approach.

For all those reasons, it could be interesting to build a Rationale Diagram from the User Story set created by the illustrated BPM. This will help visualising and analysing these User Stories and their dependencies. We thus start the upfront tagging work of the User Story set. The result is shown in Appendix 9. With these tagged User Stories, we are now able to build the Rationale Diagram with a CASE-Tool as presented in Figure 11.

From the Rationale Diagram on Figure 11, we can now better understand the links between the User Stories and their granularity levels. Indeed, while building the diagram, we realise that some User Stories are missing and that the initial requirements identified through the first approach are incomplete. This was impossible to see with the illustrated BPM. The Rationale Diagram provides thus a much finer view of the dependencies between the User Stories. However, the validity of this view is based on an accurate upfront tagging work. But given that the User Story set is incomplete, that tagging work turned out to be difficult to perform as well as building the actual Rationale Diagram.

From the Rationale Diagram, we can now identify the missing User Stories to create a complete set of requirement items. The list of 9 missing User Stories has been identified and is illustrated in Appendix 10.

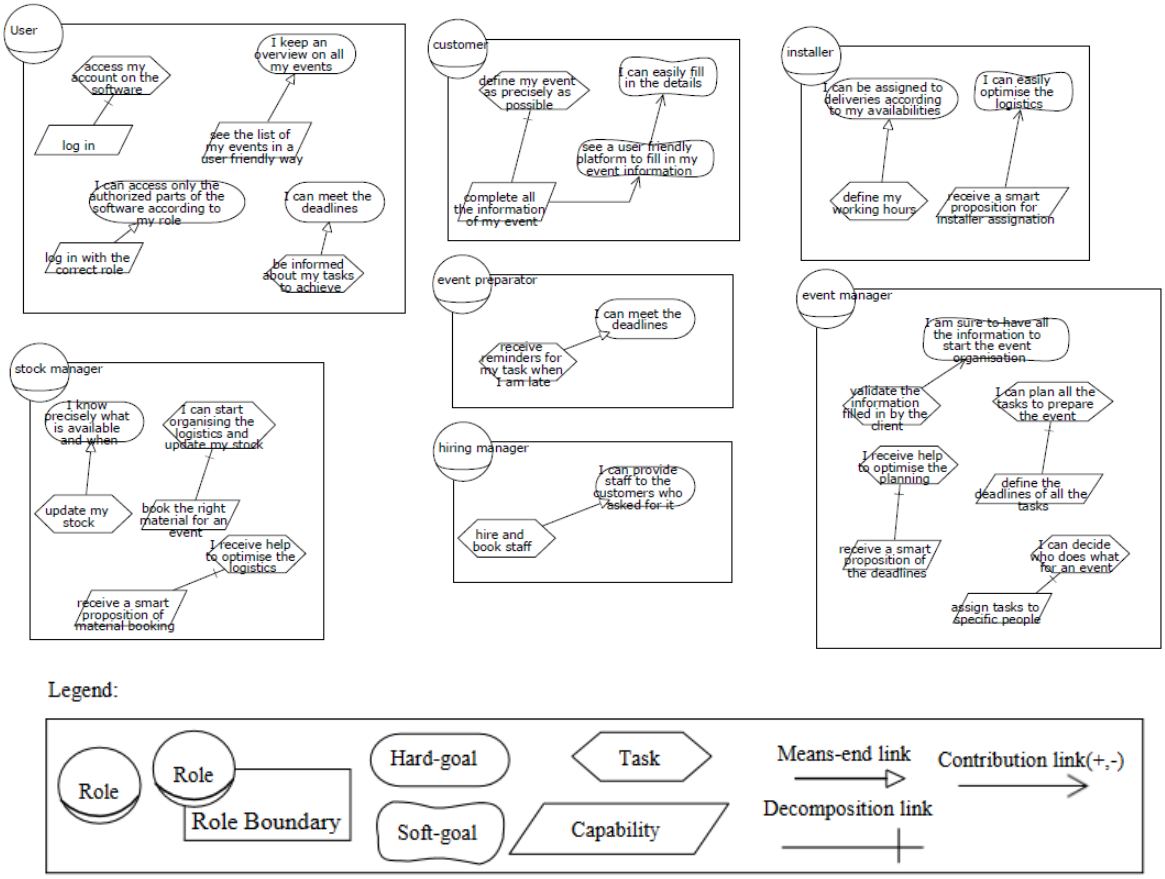

Figure 11: The Rationale Diagram created by the supporting CASE-Tool

Source: Fictional project

4.4 A comparison of both approaches

Illustrated Business Process Model	Rationale Diagram
Used for building a User Story set	Built from an existing User Story set
Group the User Stories around themes according to the processes	Group the User Stories around themes according to their dependencies
Provides no decomposition approach	Decomposition approach based on Unified Model
Vague link between User Stories	Formal links between User Stories
Completely intuitive	Based on i* notation
Provides a process oriented big picture	Provides an agent and goal oriented big picture
Capacity of identifying missing requirements is limited	Real capacity of identifying missing requirements
Time consuming to build it	Time consuming to tag and build it, but mitigated with CASE tool

Table 5: Comparative table between Illustrated BPM and Rationale Diagram

Both approaches are thus relatively complementary. Starting with the illustrated BPM approach to build the User Story set of a very process oriented organisation provides a valuable shared vision on the whole project in the case of EasEvent. But this approach does not ensure the capture of a complete set of requirements. The Rationale Diagram could then provide a complementary agent and goal oriented view to validate these requirements or identify missing items.

76.

Conclusion

The purpose of this thesis is to provide an answer to the following research question: **to what extent does Visual Thinking address the existing challenges within the agile environment?** In order to do so, we divided this thesis in 4 different chapters.

Chapter 1 firstly overviews the basic agile principles from the Agile Manifesto as well as two additional concepts that appeared to be crucial in an agile environment. During this overview, the core underlying challenges related to each principle and concept were identified. This first chapter discussed how agile practitioners are subject to numerous challenges throughout their projects while they are trying to respect the initial agile values.

Chapter 2 introduces thus the Visual Thinking concept and positions it within an agile environment. This second chapter shows to what extent the concept of Visual Thinking is in accordance with the foundational agile principles and how it could play an important role in addressing the challenges identified in Chapter 1.

Chapter 3 dives deeper into the Visual Thinking concept by reviewing the existing Visual Thinking tools in the formal and informal literature that are used by agile practitioners to meet their visualisation needs throughout their software development projects. The most widely used tools are selected and classified according to their order of involvement within the agile SDLC. Each of them are then carefully analysed while keeping an eye on the fundamental agile principles of Chapter 1. Chapter 3 concludes with an overview of the role played by each tool in the mitigation process of the identified challenges in Chapter 1.

Chapter 4 gives a more practical perspective to this thesis with a case study of how the illustrated BPMN and the Rationale Diagram are used and adapted in practice during a real life project.

Globally, we can conclude that Visual Thinking and its tools manage to address most of the underlying challenges related to each core agile principle and concept. So one could legitimately extend the benefits of Visual Thinking to the whole agile environment and its

challenges. It is no surprise that we can observe a shift towards Visual Thinking within the agile environment given that the values of both concepts are relatively consistent. Visual Thinking tools have thus the potential of addressing a substantial part of the future challenges that the agile environment will have to face.

Thesis Contribution

The agile literature is rapidly evolving as a response to the fast changing agile environment that keeps encountering new challenges. Besides this, plenty of agile practitioners more and more share their own experience on the internet in an informal and sometimes fuzzy way. As a result, the concepts of Visual Thinking and agile software development were increasingly coupled in a loosely way, but still no formal literature has been written to analyse how both concepts work out together. In that context, this thesis tries to combine these two concepts in a more structured way by clearly reviewing the formal and informal literature and analyse how introducing Visual Thinking could be beneficial to an agile environment.

Further Research

Some limitations could be identified within the research methodology and could be the subject of future research. Firstly a quantitative research among agile practitioners could be conducted to validate the selected Visual Thinking tools in the literature. Besides, a quantitative approach to analyse whether the Visual Thinking tools address the key agile challenges would also further validate the findings of this thesis which are exclusively based on a qualitative approach. Further research could also be conducted in order to collect agile practitioner feedback on the relevancy and validity of the findings. Finally, the case study is based on a relatively short set of User Stories, it could be interesting to study the benefits of visualisation on a case study with a larger sets of User Stories to investigate the scalability of the Visual Thinking tools.

References

Abrahamsson, P., Salo, O., & Ronkainen, J. (2002). *Agile software development methods*. Oulo: VTT Electronics.

Agile Alliance (2013). *What is Agile Software Development?*. Online : https://www.agilealliance.org/the-alliance/what-is-agile, retrieved on 18 May 2016.

Aitken, A., & Llango, V. (2013). A Comparative Analysis of Traditional Software Engineering and Agile Software Development. *46th Hawai International conference on system sciences.*4761-4760.

Alexander, I., & Maiden, N. (2004*). Scenarios, Stories, Use cases through the Systems development life-cycle*. New York: John Wiley & Sons.

Ambler, S. (2002*). Agile modeling: Effective Practices for eXtreme Programming and the Unified Process*. New York: John Wiley & Sons.

Ambler, S. (2009). *The Agile Software Development Life Cycle (SDLC).* Ambysoft Inc. Online : http://www.ambysoft.com/essays/agileLifecycle.html, retrieved on 16 June 2016.

Anderson, D. (2010). *Kanban: Successful evolutionary change for your technology business*. London: Blue Hole Press.

Arheim, R. (1969). *Visual Thinking*. London: University of California Press.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Boston, MA: 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*. Online : http://agilemanifesto.org, retrieved on 17 February 2016.

Bennet-Lovsey, R. (2015). The *need to respond to change over following a plan. The App Business*. Online: www.theappbusiness.com/blog/the-need-to-respond-to-change-over-following-a-plan, retrieved on 24 June 2016.

Bowan, M. (2008). *Integrating vision with the other senses*. Online: http://simplybrainy.com/wp-content/uploads/2011/01/2008-Int-Vis-Other-Senses-All-Illustrations.pdf, retrieved on 16 September 2016.

Bradley, C. (2013). *Kanban vs Scrum: Kanban is not for software development but scrum is*. The scrum crazy blog. Online: https://scrumcrazy.wordpress.com/2013/02/04/kanban-vs-scrum-kanban-is-not-for-software-development-but-scrum-is/, retrieved on 18 June 2016.

Brechner, E. (2015). *Agile project management with Kanban*. New York: Microsoft press.

Brown, C., & Topi, H. (2003). *IS management handbook (8th ed.)*. Florida: Auerbach Publications.

Burrows, Mike (2014). *Kanban From The Inside*. London: Blue Hole Press.

Buzan, T., & Buzan, B. (2006). *The Mind Map book*. Essex: BBC Active.

Charette, R. N. (2002). *Foundations of Lean Development: The Lean Development manager's guide*. Spotsylvania: ITABHI Corporation.

Chapman, A. (2013). Tree swing pictures. Businessballs. Online: http://www.businessballs.com/treeswing.htm, retrieved on 14 April 2016.

Cobb, C. (2015). *The project manager's guide to mastering Agile: Principles and practices for an adaptive approach*. New York: John Wiley & Sons.

Cockburn, A. (2002). *Agile Software Development*. Boston, MA: Addison-Wesley.

Cohn, M. (2004). *User stories applied*. Boston, MA: Addison-Wesley.

Cohn, M. (2010). *Succeeding with agile*. Boston, MA: Addison-Wesley.

Deza, M., & Deza, E. (2009). *Encyclopedia of distances*. Berlin: Springer.

Fink, A. (2010). *Conducting research literature reviews*. Los Angeles: Sage Publications.

Fowler, M., & Ford, N. (2013). Why does Agile software development work so well? [Conference Video]. *USI Events, Paris*. Online: https://www.youtube.com/watch?v=GE6lbPLEAzc, retrieved on 23 April 2016.

Garder, H. (2011). *Frames of Mind: The Theory of Multiple Intelligences*. London: Basic Books inc.

Gothelf, J. (2014). Bring Agile to the whole organisation. *Harvard Business Review website*. Online: https://hbr.org/2014/11/bring-agile-to-the-whole-organization, retrieved on 18 September 2016.

Gupta, J. (2009). *Handbook of research on Enterprise Systems*. New York: Information Science Reference.

Hazzan, O., & Dubinsky, Y. (2008). *Agile Software Engineering*. London: Springer.

Highsmith, J., & Fowler, M. (2001). *The Agile Manifesto*. Online : http://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf, retrieved on 24 February 2016.

Hiranabe, K. (2008). *Exploring User Requirements through Mind Mapping*. Online : http://www.change-vision.com/en/ExploringUserRequirementsThroughMindMapping_Letter.pdf, retrieved on 16 October 2016.

Holub, A. (2014). The death of agile [Conference video]. *Software Architect 2014*. Online: https://www.youtube.com/watch?v=HZyRQ8Uhhmk, retrieved on 11 June 2016.

Hubbard, D. (2007). *The IT measurement inversion*. Online: http://www.cio.com/article/2438748/it-organization/the-it-measurement-inversion.html, retrieved on 17 June 2016.

Humble, J. (2015). Why scaling agile doesn't work. *GOTO Berlin 2015*. Online: https://www.youtube.com/watch?v=2zYxWEZ0gYg, retrieved on 17 May 2016.

Idon Thinking Resources. (2003). *Visual thinking*. Online: http://www.idonresources.com/ct/visualthinking.html, retrieved on 5 May 2016.

Jackson, J. (2015*). No more sticky notes*. Online: https://sprint.ly/blog/no-more-sticky-notes, retrieved on 14 September 2016.

Kahkonen, T., & Abrahamson, P. (2003). Digging into the fundamentals of extreme programming: Building the theoretical base for agile methods. In: *29th Euromicro Conference*.

Kniberg, H. & Skarin, M. (2010). Kanban and Scrum - Making the Most of Both. USA: InfoQ Enterprise Software Development series.

Kotermansky, R. (2016). *Uniting agile and design thinking methods. Summa*. Online: http://www.summa.com/blog/uniting-agile-and-design-thinking, retrieved on 28 May 2016.

Küpper, S. (2016). *The impact of Agile methods on the development of an agile culture*. Technische Universität Clausthal, Clausthal.

Lacey, M. (2012). *The scrum field guide: Practical advice for your first year*. Boston, MA: Addison-Wesley.

Larman, C. (2004). *Agile and Iterative Development: A Manager's Guide*. Boston, MA: Addison-Wesley.

Larman, C. (2012). *Scaling agile with large scale scrum* (Ericsson Keynote video). Online: https://www.youtube.com/watch?v=Gw1lLt18KzE, retrieved on 14 May 2016.

Larman, C. & Vodde, B. (2008). *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Boston, MA: Addison-Wesley.

Larman, C. & Vodde, B. (2014). *Large-Scale Scrum: More with LeSS.* Boston, MA: Addison-Wesley.

Layton, M. (2012). *Agile Project Management For Dummies*. New York: John Wiley & Sons.

Leau, Y., Loo, W., Tham, W., Tan, S. (2012). Software Development Life Cycle Agile vs Traditional Approaches. In: *2012 International Conference on Information and Network Technology (ICINT 2012)*, Singapore.

Leffingwell, D. (2011). *Agile software requirements*. Boston, MA: Addison-Wesley.

Lindstrom, L., & Jeffries, R. (2004). Extreme Programming and Agile Software Development Methodologies. *Information Systems Management*, *24*(3), 41-52.

Liskin, O., Pham, R., Kiesling, S., & Schneider, K. (2014). Why we need a granularity concept for user stories. In: *Agile Processes in Software Engineering and Extreme Programming*, 179, 110-125.

Martin, R. (2014). *The true corruption of agile*. Online: https://8thlight.com/blog/uncle-bob/2014/03/28/The-Corruption-of-Agile.html, retrieved on 25 June 2016.

Medina, J. (2008). B*rain Rules : 12 principles for surviving and thriving at work, home, and school*. Seattle: Pear Press.

Michalko, M. (2011). *Cracking creativity, the secrets of creative genius*. Berkeley, CA: Ten speed press.

Moraksy, M. (2012). *Design Thinking vs Visual Thinking.* Online: http://www.xplane.com/buzz/insights/preview/design-thinking-vs-visual-thinking/view/, retrieved on 14 August.

Otis, L. (2016). *A new loot at Visual Thinking. Psychology Today*. Online: http://www.creativitypost.com/psychology/a_new_look_at_visual_thinking, retrieved on 12 September 2016.

Patton, J. (2014). *User Story Mapping, Discover the whole story, Build the right product*. Sebastopol, USA: O'Reilly Media.

Pichler, R. (2014). *From Personas to User Stories*. Online: http://www.romanpichler.com/blog/personas-epics-user-stories, retrieved on 19 September 2016.

Poppendieck, T., Poppendieck, M. (2003). *Lean Software Development: An Agile Toolkit*. Boston, MA: Addison-Wesley.

Quirk, J. (2008). *Meaning through metaphor: Visual dialogue and the picturing of abstraction.* Master thesis in Art, University of Massachusetts Boston, Boston.

Rajagopal, P., Lee, R., Ahlswede, T., Chiang, C. Karolak, D. (2005). A new approach for software requirements elicitation. In: *Sixth International Conference on Software Engineering*.

Ramesh, B., Cao, L., Baskerville, R. (2007). Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal, 20*(1), 449–480.

Rogalsky, S. (2011). *User Story mapping - rounding out your backlog*. Online: http://www.slideshare.net/SteveRogalsky/user-story-mapping-9950670, retrieved on 19 June 2016.

Rogers, B., & Howard, K. (2011). *Individuals and Interactions: An Agile Guide.* New York: Pearson Education, Inc.

Rosenberg, D., Stephens, M. (2003). *Extreme Programming Refactored: The Case Against XP*. Berkeley, CA: Apress.

Rossberg, J. (2014). *Beginning Application Lifecycle Management*. Berkeley, CA: Apress.

Rother, M. (2009). *Toyota Kata: Managing people for improvement, adaptiveness and superior results*. New York: McGraww-Hill.

Rubin, K. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Boston, MA: Addison-Wesley.

Sicinski, A. (2011). *Visual thinking, the path to genius?.* Online: http://www.visualthinkingmagic.com/path-to-genius, retrieved on 5 August 2016.

Siau, K., Lee, L. (2004). Are use case and class diagrams complementary in requirements analysis? An experimental study on use case and class diagrams in UML. *Requirements Engineering*, *9*(4), 229-237.

Schwaber, K, & Beedle, M. (2001). *Agile Software Development with Scrum*. Upper Saddle River, NJ: Prentice Hall.

Scrum Alliance (2014). *What is Scrum? An Agile Framework for Completing Complex Projects - Scrum Alliance*. Online: https://www.scrumalliance.org/why-scrum, retrieved on 17 April 2016.

Shiralige, A. (2012). *Agile principle: Simplicity - The art of maximising the work not done.* Online: http://www.agilebuddha.com/agile/agile-principle-simplicity-the-art-of-maximising-the-work-not-done/, retrieved on 21 June 2016.

Sommerville, I. (2007). *Software Engineering (8th ed.)*. Boston, MA: Addison-Wesley.

Strode, D. (2005). *The agile methods: An analytical Comparison of Five Agile methods and investigation of their target environment.* Master thesis in Information Sciences in Information Systems, Massey University, Palmerston North, New Zealand.

Sutherland, J. (2004). *Agile Development: Lessons learned from the first Scrum*. Online: http://csis.pace.edu/~marchese/CS616/Agile/Scrum/FirstScrum2004.pdf, retrieved on 16 September 2016.

Sutherland, J., & Schwaber, K. (1995). Business object design and implementation. In : *ACM SIGPLAN OOPS Messenger*, *6*(4), 170-175.

Svarytsevych, D. (2015). *How Visual thinking can solve project management problems. Techwell insights*. Online: https://www.techwell.com/techwell-insights/2015/05/how-visual-thinking-can-solve-project-management-problems, retrieved on 25 April 2016.

Taylor, F. (1911). *The Principles of Scientific Management*. New York: Cosimo Classics.

Thomas, D. (2015). Agile is dead, long live agility [Conference video]. *GOTO Amsterdam 2015, Amsterdam*. Online: https://www.youtube.com/watch?v=a-BOSpxYJ9M, retrieved on 11 June 2016.

Trevarthen, C. (1990). *Brain circuit and functions of the mind: Essays in honor of Roger W. Sperry*. New York: Cambridge University Press.

van Lamsweerde, A. (2004). Goal-oriented requirements enginering: a roundtrip from research to practice [enginering read engineering]. In: *Requirements Engineering Conference, 2004,* 4-7.

van Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. New York: John Wiley & Sons.

van Waardenburg, G., & van Vliet, H. (2013). When agile meets the enterprise. *Information and Software Technology*, *55*(12), 2154-2171.

van Valkenhoef, G., Tervonen, T., de Brock, B., & Postmus, D. (2011). Quantitative release planning in Extreme Programming. Preprint submitted to: *Information and Software Technology*.

Waters, K. (2007). *Definition of done*. Online: http://www.allaboutagile.com/definition-of-done-10-point-checklist, retrieved on 9 May 2016.

Wautelet, Y., Heng, S., Hintea, D., Kolp, M., Poelmans, S. (2016a). Bridging User Story Sets with the Use Case Model. In: *International Conference on Conceptual Modeling,* 127-138.

Wautelet, Y., Heng, S., Kolp, M., Mirbel, I.(2014). Unifying and extending User Story Models. In: *Advanced Information Systems Engineering*, 8484, 211-225.

Wautelet, Y., Heng, S., Kolp, M., Mirbel, I., & Poelmans, S. (2016b). Building a Rationale Diagram for Evaluating User Story Sets. In: *Research Challenges in Information Science (RCIS)*, 477-488.

White, S. (2008). *BPMN modeling and reference guide: Understanding and using BPMN*. Florida, USA: Future Strategies.

Williams, P. (2015). *Visual Project Management*. Raleigh, NC: Lulu Press.

Yu, E., Giorgini, P., Maiden, N., & Mylopoulos, J. (2011). *Social Modeling for Requirements Engineering.* Massachusetts: MIT Press.

Zujewska, B. (2009). *Visual thinking: Mind maps*. Online: http://www.idc.ul.ie/anu/CS6022/Group%20Assignment/Visual%20Thinking,%20Sketching%20and%20Mindmapping%20Beata&Rachel/MindMaps%20Beata%20Zujewska.pdf, retrieved on 15 October 2016.