KTH Computer Science
and Communication

Agile Development in a Lonely Environment

How to Develop Software Using Agile Techniques Within Small-Scale Projects

MATHIAS LINDBLOM

Master's Thesis at KTH Royal Institute of Technology
Computer Science and Communication (CSC)
Supervisor: Sten Andersson
Examiner: Olle Bälter

Abstract

Most agile software development methodologies are aimed towards large-scale projects and companies. The purpose of this report was to identify agile development methodologies and strategies suitable for small-scale projects. Initially a comprehensive study was conducted on a few selected agile methodologies in order to find the one most suitable to be used for a defined small-scale project. The result of the study led to the methodology Kanban being used experimentally on the project. The report goes through how Kanban was utilized and modified to suit the project as well as bring forward interesting details regarding the outcome of the actual project.

The resulting methodology, with Kanban at its core, that was designed can especially be of use for small IT companies looking for an agile methodology fitting their project(s). It can also be used to further develop alternative variants of the agile methodology Kanban with focus on small-scale projects.

Referat

Agil Utveckling i en Ensam Miljö

De flesta agila utvecklingsmetodiker riktar sig mot storskaliga projekt och företag. Syftet med denna rapport var att identifiera lämpliga agila utvecklingsmetodiker och strategier för småskaliga projekt. Inledningsvis genomfördes en omfattande studie av ett fåtal agila metodiker med målet att hitta den mest lämpliga för ett definierat småskaligt projekt. Resultat af studien ledde till att metodiken Kanban experimenterades med och användes för projektet. Rapporten går igenom hur Kanban nyttjades och modifierades för att passa projektet i fråga.

Den framtagna metodiken, med Kanban som grund, kan vara speciellt användbar för mindre IT-företag som letar efter en agil metodik som passar deras projekt. Den skulle även kunna användas för att vidareutveckla alternativa agila Kanban-metodiker för småskaliga projekt.

Acknowledgements

I would like to thank Per-Arne Forsberg. He has been my mentor, advisor and my morale support. His feedback as well as insight regarding agile development has been tremendously helpful. I am also thankful that Per-Arne entrusted me to work on his project, despite knowing my lack of experience within web development.

Contents

Chapter 1

Introduction

This chapter goes through the problem statement including goals, the target audience and limitations. The last section will outline the structure and layout of the rest of the report.

1.1 Problem Statement

Agile software development can be seen as a trend in today's computer industry. The variety of agile methodologies as well as their definitions changes frequently, which is quite appropriate given what agile development advocates. Most agile methodologies mainly targets established companies with advanced software systems. They talk about things like continuous improvement, rapid changes to new customer demands and the collaboration between self-organizing and cross-functional development teams using an iterative development process. However the information available regarding how-to practically establish an agile development process in small companies with small-scale projects is limited. The available literature can give hints but has a strong focus on the full blown methodologies and leaves it up to the readers to decide what is practically possible for their limited sized development group or company.

The vast information available, complexity and focus on established companies can make it hard and confusing for developers wanting to adopt an agile mindset when starting their new company with only a few people, or even by themselves. Trying to follow most agile methodologies completely, or other methodology types for that matter, would likely be overwhelming and lead to unnecessary early abandonment of that methodology. By finding a good approach to how small development teams and/or companies can adopt an agile development process, more of them could benefit from the potential positive outcomes that comes from utilizing agile development.

1.1.1 Goals

The main goal of the thesis was the following:

- Find an agile development methodology and potentially modify it in order to make it suitable for small-scale IT projects and/or companies consisting of up to five members.

Secondary goals:

- Do a theoretical comparison of popular agile development methodologies suitable for small-scale IT projects and/or companies.
- Overall clarification of agile terms and thinking/reasoning. For example differentiate *agile* from *lean*.
- Produce a working prototype of the interactive tutoring framework described at section 5.1 Project Description.

To summarize and clarify: An actual small-scale project was to be developed, alongside theoretically analyzing agile development, with the common goal of finding a suitable agile methodology and/or strategies.

1.1.2 Target Audience

The target audience are mainly small IT companies with limited experience in software development methodologies. However, anyone having an interest in developing something using an agile approach, either alone or with a handful of people, might find this report useful.

For those lacking prior knowledge regarding agile development, chapter 2 Agile Development together with chapter 3 Agile Methodologies contains enough content to make the reader acquainted to the subject and understand the rest of the report. After all, given the target audience, it would be inappropriate not to offer the readers a comprehensive background.

1.1.3 Limitations

This report does not seek to investigate whether or not agile development as a whole is advantageous, compared to other methodology types, for small-scale projects. Instead it makes the assumption that agile development is advantageous and investigates which agile methodologies and strategies are more suitable for small-scale projects.

The large scope of the agile development methodologies, the many social parameters, and variety of projects means that the outcome of this report can't come to definite conclusions. It can't for example conclude that a certain methodology is the absolute best choice for small-scale projects. This also explains the vague word

suitable in the goals section above. This is brought up again in chapter 6 Discussion. The report should be seen as an comprehensive, theoretical and experiential based investigation.

1.2 Thesis Outline & Information

The report will begin by giving a background in chapter 2 Agile Development. This background is meant to clarify what agile development means and also provides the answers to the second secondary goal mentioned in section 1.1.1 Goals. It includes interpretations and simplifications by the author in order to assist the inexperienced reader, this is especially true for the figures supplied. Following is chapter 3 Agile Methodologies which is a comprehensive background and walkthrough of four agile methodologies and their practices. Some of these practices will be mentioned throughout the rest of the report. Note that whenever a practice is mentioned it's written in camelcase. This is to emphasize to the reader that this is a practice and has a section in the agile methodologies chapter, without having to fill the report with references.

The report does not have a strict method or result chapter. Having them strictly separated is not suitable for this kind of report. The reason for this is that the workflow of finding suitable agile methodologies and strategies, adapting and modifying them for a project, goes through sort of iterations and slowly build up piece by piece. This, together with the fact that the report is meant to be usable as a form of guide for inexperienced small project teams, makes it more natural to let the reader follow the same workflow as the project went through.

In chapter 4 Methodology Evaluation a short theoretical evaluation is conducted with the agile methodologies background as the basis. This chapter is all about analyzing the methodologies for small-scale projects and finding a good candidate to be used in the actual project. Afterwards, in chapter 5 The Project, the project is initially described. This chapter goes through both how the chosen methodology developed and the practical results of the actual project. It ends with a short interview with the projects product owner. The reasoning behind important decisions are incorporated in the chapter in order to give the reader some insight into the development process.

The report ends with chapter 6 Discussion. Here the developer, who is the report writer, gives his experience on the development process. The overall results of the project are discussed and interesting information and critique regarding the report and project are brought forward.

Several figures in the report make heavily use of colors. Therefor it's recommended, for the reader that uses a black and white printout, to have access to a digital colored version. Also, each reference to figures and chapters/sections are colored red. What might be most important to point out is that each glossary entry are colored red to emphasize to the reader that more information is available in the glossary at the end of the report.

Chapter 2

Agile Development

This chapter goes through agile development as a concept and highlights its core ideas and principles. It also explains the differences between agile development and the waterfall model as well as lean development.

Most developers have experienced the nightmare of working on a project and slowly witness the increasing number of bugs, errors, defects and overall time spent when adding new functionality. In order to combat this disheartening scenario, developers tend to create constraints and routines around their activity, forming a development process based on prior mistakes. However big development projects can be complex and even though experienced based constraints helps, the problems remains and builds upon each other. One solution for this is to keep adding more constraints and routines. Which means that the development process grows and grows until the process itself becomes so cumbersome and advanced that it creates the problems it was designed to prevent, which would be technical dept and diminishing efficiency [1].

This so called runaway process inflation was common among big companies around the millennium shift and at the time it was a growing negative trend [1]. Software teams around the world experienced this, including a few industry experts calling themselves the Agile Alliance who sought to fight it [1]. The ideas that make up agile development were not new at the time but it was the Agile Alliance that formulated the underlying concepts that defines agile development. The report will come back to this at section 2.2 The Agile Manifesto.

2.1 Component Abstraction

It's common that agile development is directly compared to the waterfall model which is strange given how they situate on different abstraction levels. The Waterfall model is a software development methodology just like for example Scrum. Agile development on the other hand is not a followable methodology by itself. It's more of an ideology or perhaps a collection of methodologies. So how is it even possible to compare the two of them? The answer is that people are comparing the waterfall

model to some popular agile methodology such as Scrum or some common practices like Sprints and standup meetings, indirectly claiming that these elements represent agile development as a whole. This leads to confusion and unfair comparisons. So before continuing the background regarding agile development, this section gives a short overview of different components within agile development. Bare in mind that this is a simplified overview meant to give the reader a better sense of how agile development and the methodologies with its practices correlate to each other.

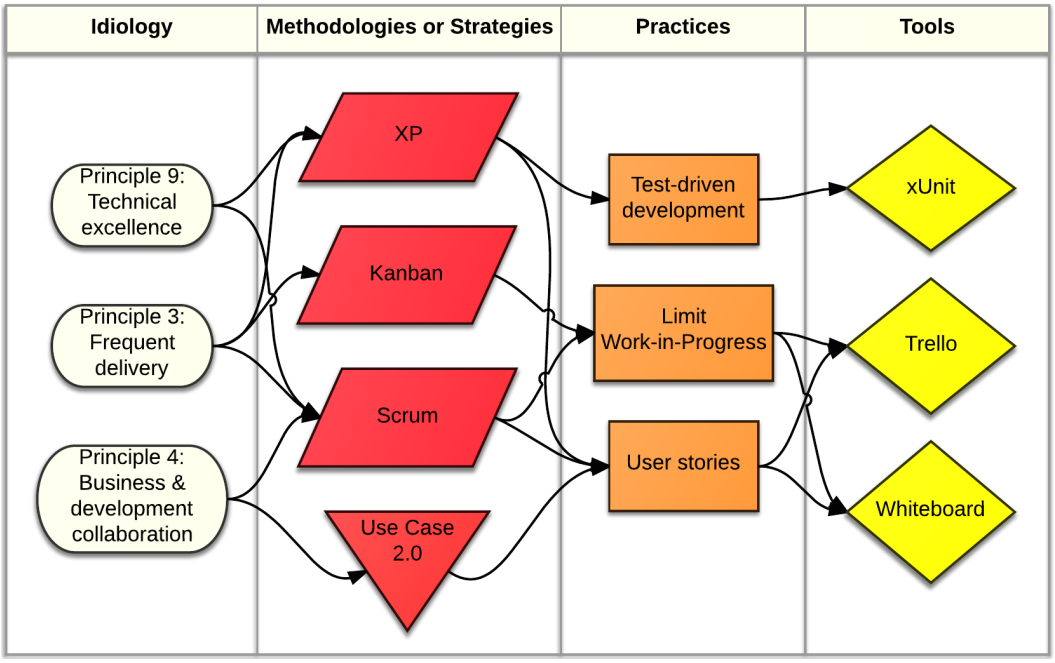


Figure 2.1 – Agile development divided into four different abstraction levels starting with the ideology that defines agile development and ending with physical tools. It also shows how the example elements within them correlate to each other. The triangle under the second column is to emphasize that it's an agile strategy and not a methodology.

In Figure Figure 2.1, agile development have been divided into four different abstraction levels. First we have the ideology that consist of the core ideas such as The Agile Manifesto and principles. This is basically what defines agile development, which will be explained at section 2.2 The Agile Manifesto and forward. Secondly we have the methodologies and strategies that fulfills some or all of the agile principles and philosophies. The third abstraction are the practices that defines the methodologies, similar to how the agile principles defines agile development. However, the practices are more practically applicable and at a much lower abstraction level than the agile principles. Lastly, at the forth abstraction level, we have the actual tools that are needed to make the practices a reality. These tools can be very simple such as a physical room for meeting practices or more advanced such as the xUnit framework for test-driven development.

The report will frequently talk about these different components so it's a good idea to keep a mental image of Figure 2.1. Moreover, this report will refer to agile development as an ideology in order to not confuse it with its methodologies.

2.2 The Agile Manifesto

The Agile Alliance is an organization created by a group of computer industry experts in 2001. They decided to come together in order to form general development values with the goal of improving the software development process for companies around the world, and to give an alternative to the popular heavyweight methodologies [1] [2]. The result was The Agile Manifesto that states the following:

- **Individuals and interactions** over processes and tools.
- **Working software** over comprehensive documentation.
- **Customer collaboration** over contract negotiation.
- **Responding to change** over following a plan.

The group that wrote The Agile Manifesto consisted of the following people: *Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas* [1] [2]. Several of these people are the authors of the literature examined for this report, this is especially true for chapter 3 Agile Methodologies.

The following four subsections will give a short explanation for each of the statements of The Agile Manifesto that stands as the highest abstraction or the core that defines agile development. These sections gives the reader a good overview of the thinking and reasoning regarding agile development.

2.2.1 Individuals and Interactions over Processes and Tools

> Process and technology are a second-order effect on the outcome of a project. The first-order effect is the people.
>
> Alistair Cockburn [1]

Within football most can agree on that a football team of average players, that communicate well, can beat a team of egoistic superstars. The same goes for software development (and probably many more things) where a team of average programmers working together outperforms a team of expert programmers that fail to communicate. At least according to the thinking and reasoning regarding agile development [1].

Processes and tools are of course important as well but not as important. After all, agile development methodologies are processes that usually requires tools so it would be quite ironic to deem them as unimportant. Team managers have a habit of setting up the development environment and then assigning a team, assuming that the team will work well since they have the tools they need [1]. However this is focus put at the wrong end of order. Tools and processes can rarely repair a team that doesn't play ball but a functioning team is usually more than capable of configuring their own environment.

2.2.2 Working Software over Comprehensive Documentation

> The program is the specification and documentation.
>
> Douglas Crockford [3]

This might not be that much of a shocker, that functioning software is more important than documentation. The keyword here is *comprehensive* since it's common that teams get hung up on the quest of having close to 100 percent coverage on documentation [1]. The problem with this is that it takes a lot of time and effort to always keep in sync with the code and as soon as you lose the sync, the documentation begins telling lies which is worse than having no documentation. Instead of having every technical part of a program described in a thick book it's usually preferred, both for the reader and the writer, to have a short rational and structured document that explains the software at a high abstraction. When a new member of a team needs more technical information they will get it when working closely and interacting with their team members and the actual software.

A common misconception is that code is for the computer and documentation is for the human to understand the code [4]. Instead of translating the code into human language, why can't we write code so that humans can understand it in the first place? We already do it by naming things like classes and functions with understandable names, but is usually ends there. We make it *good enough*. If we instead focused more on the quality and logic of the code, the value of documentation would be greatly diminished.

2.2.3 Customer Collaboration over Contract Negotiation

Many have tried to create strict contracts with fixed specifications, deadlines and prices for software development. Many have failed [3]. You simply can't treat software as a commodity expecting everything to be known before hand. Trying to do so can lead to all sorts of problems like poor quality or even complete failure. It's not only the complexity of programming that is the issue but also how fast our technical world changes over short time periods.

This is one of the core problems that agile development tries to avoid. By having a tight collaboration with the customer the specifications, deadlines and prices don't need to be as strict. Instead of having a contract specifying the exact

requirements of the entire project it should instead address how the collaboration between customer and developers should be conducted [3]. This leads to a more dynamic development approach where technical problems can be addressed directly. Moreover, needed changes don't risk breaking the contract.

2.2.4 Responding to Change over Following a Plan

The previous section regarding customer collaboration is an indirect example of this so called mantra. Customers have a habit of not knowing what they really want and changing their requirements after seeing some of the functionality coming to life [3]. Also as stated in the previous section, the IT world we live in rapidly changes and so can the requirements of a software project. It's nice and tempting to have the full project planned out in some advanced planning software with every functionality written. However as the development team and customer moves forward the plan is bound to change. Some functionality will no longer be deemed necessary and previously unthought of functionality might be added. Some parts of the strict plan might then have been a waste of time and more importantly, the strict plan might hinder these kind of changes that are needed or at least advantageous to the project.

By having a more abstract and lose plan with details only written for the near future, for example a couple of weeks, you get a planning more susceptible to change [3]. That detailed plan can be strict but since it's only for a short time period the risks of having unneeded functionality or hindering change is limited. The further away in time we look the weaker the plan should be since we want flexibility to changes and the likelihood of changing requirements grows as the timeline increases.

2.3 The Agile Principles

Together with the values, or mantras, stated by The Agile Manifesto, twelve principles were created that act as the characteristics for agile practices [1]. You can interpret these principles as another formulation of The Agile Manifesto where you get a clearer explanation of what agile development means. These are the principles:

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

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

An explanation for each of the principles does not follow in this report, as was done with The Agile Manifesto. This is because they are much more self-explanatory, and also quite similar to The Agile Manifesto which would mean writing very similar explanations. However principle 10 about simplicity is quite hard to interpret, at least compared to the other principles. What it means to say is that it's important to prioritize. Only do what is essential and leave the rest, for now. Features and procedures in a project that adds little value are not only a waste of time to implement but can also add unnecessary complexity. So maximizing the avoidance and removal of low value components is essential for a project, in the long run. Keep it simple.

2.4 Agile vs Waterfall

This section gives a short and illustrative view of the differences between agile development and the more established software development methodology called the waterfall model. As explained in section 2.1 Component Abstraction, one should be careful doing detailed comparisons between agile development and other methodologies since agile development is more of a collection of methodologies. For this reason the comparison is held highly abstract and kept factual without valuing the differences.

As seen in Figure 2.2, a project moves sequentially from one logical phase to the next in the waterfall model and rarely moves back to an prior phase. This means that once a project leaves a phase, it's assumed that that phase is completed for that project. Feel free to visualize a multi-step waterfall moving a log (project) from one water level (phase) to the next. The phases can hold different projects at any given time. Once a team in a certain phase is finished with a project, that project is moved to the next phase and the team is ready to receive a new project from the previous phase.

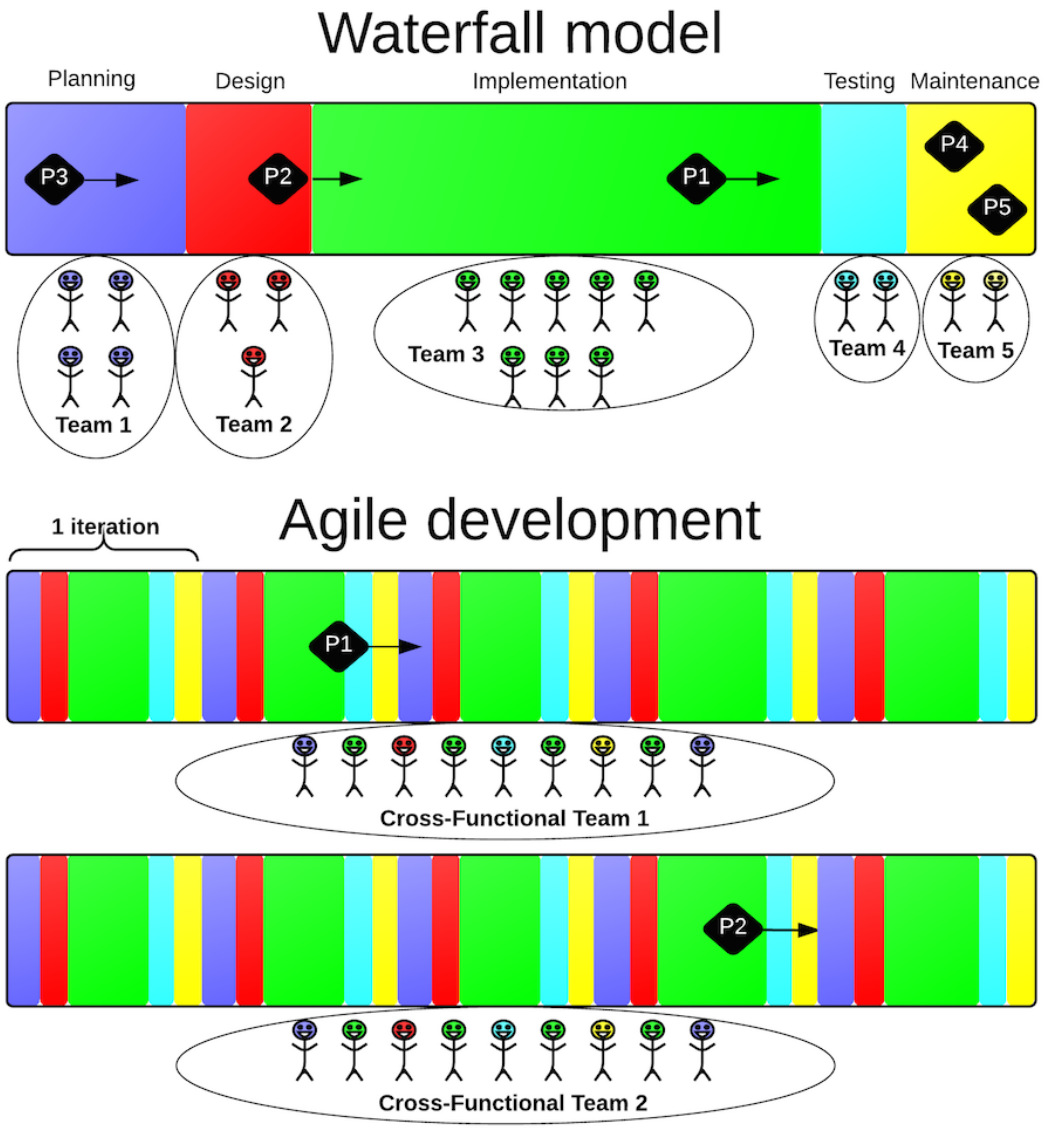


Figure 2.2 – Simplified highly abstract image showing the difference between the waterfall model and agile development. The image portrays how projects move forward in an arbitrary timeline through difference phases. P1 to P5 stands for Project 1 to Project 5. Each color represents a logical phase for a project such as planning or design. The stick figures represents team members and their color represents expertise within a certain phase. The image is further explained in section 2.4 Agile vs Waterfall.

All of this means for example that the implementation team does not care about design, or at least design is not their area of concern. In Figure 2.2, when the design team is finished with project 2 (P2) it's moved to the implementation phase and the design team awaits the arrival of P3 from the planning team.

Do note that the word *team* is a simplification. In reality in the waterfall model, each phase can contain multiple development teams or so called functional units. The point is that each phase can be much more advanced than to simply contain a development team.

In agile development, the project does not normally move from one distinct phase to another. Instead, the project moves through layers of phases in a loop like fashion, as seen in Figure 2.2. Another difference is that the same team holds a project until it's completely finished before starting to work on another project. In order to make that work it's vital that entire team incorporates all necessary skills within a project. A team that satisfies this is called a cross-functional team which are often mentioned within agile development. If another project starts, another cross-functional team is assigned that project, which the previous figure tries to illustrate by having two separate project flows for agile development.

2.5 Agile vs Lean

When you read about different software development methodologies they can sometimes be referred to as *lean*, sometimes *agile* and sometimes *lean and agile*. Often without explanation of what specific parts makes the methodology lean and what makes it agile which can be very confusing. To understand the confusion it's best to give a short explanation of what lean development means:

Lean

Lean Product Development (LPD), or lean manufacturing, dates back to the mid 20th century. The history of LPD begins with Toyota that sought to improve car manufacturing efficiency, quality and flexibility. One way to achieve this, according to LPD, is to have the entire chain from design to full assembly at one physical place [15]. This is comparable to how agile development promotes having cross-functional teams develop a project from start to finish.

Lean Software Development (LSD) is simply the adaptation of LPD to the software industry that originates from the book *Lean Software Development: An Agile Toolkit* by Mary Poppendieck and Tom Poppendieck [16]. Like with agile development, LSD has a few principles that gives a good overview of what defines the methodology framework or ideology:

1. Eliminate waste
2. Amplify learning
3. Decide as late as possible
4. Deliver as fast as possible
5. Empower the team
6. Build integrity in
7. See the whole

Note that LSD has just over half the amount of principles compared to agile development. Without going into detail for each specific principle we can see that several of the lean principles have similarities to the agile principles. L1 (lean principle number one) says to remove waste while A10 (agile principle number ten) talks about avoiding low value components. L2, about increasing learning, is implied in several of the agile principles. L3 is about delaying decisions as late as possible to make the best informed decision which is a simplified version of A2. L4 and A3 both talks about fast delivery. L5 basically says the same things as A5, just worded differently.

So by looking at the principles it's safe to say that the lean and agile ideologies have several similarities. They both talk about adaptive planning, empowering the people, fast delivery, continuous improvement etcetera. The differences lies in focus and scope that becomes more apparent when reading literature about them. LSD focuses on minimizing waste and streamlining workflow while agile development focuses on adaptivity and communication. LSD reaches a higher range than agile software development by going all the way to talk about costumer vision, staffing and maintenance while agile development has a more strict focus on the project development phase.

What is important to note is that agile and lean do not go against each other. This means that methodologies trying to adopt these ideas do not have to be either agile or lean but can in fact be both, with perhaps a focus towards one of them. This chapter will end with a suitable long quote from one of the authors of The Agile Manifesto.

> So as you can see, lean and agile are deeply intertwined in the software world. You can't really talk about them being alternatives, if you are doing agile you are doing lean and vice-versa. Agile was always meant as a very broad concept, a core set of values and principles that was shared by processes that look superficially different. You don't do agile or lean you do agile and lean. The only question is how explicitly you use ideas that draw directly from lean manufacturing.
>
> Martin Fowler [17]

Chapter 3

Agile Methodologies

Up to this point the ideology, that is agile development, has been described. In this chapter the more common methodologies that try to adapt these ideas and mentalities will be brought up. Moreover, the methodologies described here will contain more practically applicable agile practices. What is important to note is that every agile methodology is subject to change. Most authors and advocates of the agile methodologies encourages changes in the methodology itself to suit your development team and/or company. This means that the descriptions of the methodologies can look more or less different depending on the source, especially when they have been written years apart. Another reason for this is that some parts of a methodology can be vague resulting in different interpretations. However this does not necessarily mean that some descriptions of the practices are wrong. The vagueness can be seen as a way of encouraging changes and alternative implementations of the methodologies which the authors often advocates.

The methodologies can be extensive in their documentation and this report does not seek to explain each methodology in their deepest details. That would simply require too much work and be overwhelming for the readers. For the interested reader, each methodology will have recommended further reading in its introduction.

One thing that every agile methodology has is a common set of practices that explain the most important components. Some methodologies refer to them as properties but they all seek to explain the core of the methodology similar to how the agile principles, in section 2.3 The Agile Principles, relate to agile development. It was therefor suitable to use them as the base for this reports overview of the methodologies in order to ensure that the description and comparisons are based on somewhat equal grounds.

A warning to the reader is that going through every practice of all four methodologies in order might be quite heavy reading. A suggestion is to read the methodologies introductions and the titles of their practices, and later go back to read the content when needed as the report starts investigating the practices at chapter 4 Methodology Evaluation.

3.1 Scrum

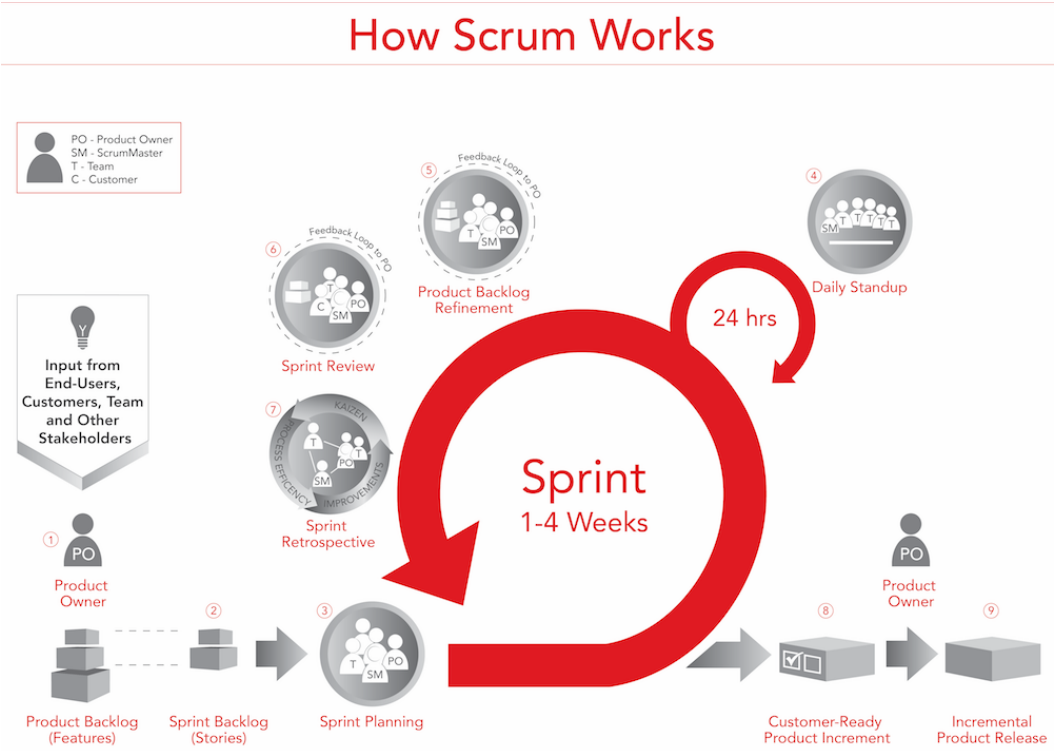


Figure 3.1 – Image showing an abstract view of how Scrum works with focus on all eleven practices. The three roles, with the addition of customer, are listed in the upper left corner of the image.
Source: Scruminc, http://www.scruminc.com. Scruminc has given their consent to reuse their image.

Scrum is one of the most known and used agile methodologies today. Scrum was originally codeveloped during the early 1990s by two of the soon-to-be authors of The Agile Manifesto, Jeff Sutherland and Ken Scwaber [14]. Several other mentionable people, such as Mike Beedle, have further contributed to the development of Scrum. Since the agile methodology Scrum was conceived just a few years before The Agile Manifesto and by several of the authors, it's perhaps not that shocking that people sometimes wrongfully put an equality mark between agile development and Scrum.

Scrum goes against the traditional predictive way of developing and instead has a more empirical approach. What this means is that Scrum assumes that problems and challenges, to a great extent, can't be foreseen in software development. Therefor focus lies on maximizing the Development Team's ability to deliver quickly and respond to changing requirements.

The information regarding the practices of Scrum is mainly taken from the book *Agile Project Management with Scrum* by Ken Schwaber, one of the authors of The Agile Manifesto, together with updated information from the official Scrum web page: www.scrumguides.org [14] [10]. There are a total of eleven practices in Scrum divided the categories Roles, Events and Artifacts. Both the book and the web page are recommended for further reading regarding Scrum.

3.1.1 Roles

There are three distinct roles in Scrum. The Product Owner, Development Team and Scrum Master. The cooperation and relation between these roles is a key component in Scrum. The Scrum team consists of every member within these three roles.

Product Owner

The Product Owner is a single person. He/she is responsible of maximizing the work conducted by the Development Team which in turn means responsibility to maximize the product value. The practical priority is managing the Product Backlog. The managing of the backlog means taking care of the following:

- Ensuring that the Product Backlog items have clear and distinct descriptions.
- The ordering of backlog items to maximize value and reaching goals.
- Optimization the value of the work done by the Development Team.
- Making the Product Backlog visible, transparent, clear to all project members and include information about upcoming tasks.
- Having the Development Team on a satisfactory level of understanding of the items in the Product Backlog.

Depending on the size of the project and company. The Product Owner may choose to distribute some of the tasks to other people such as the Development Team. However, the Product Owner is always the accountable person for the above tasks [10].

Development Team

In Scrum the Development Team is in charge of delivering working software, as one might expect. They are in a large sense self governed and chooses how they want to work, within some guidelines of course. The Development Team can be summarized with these characteristics:

- Self-organizing. The Development Team alone chooses how they want to turn the items in the Product Backlog into functionality.

- Cross-functional. The team should encompass all necessary skills to work with everything in the Product Backlog.
- No titles within the group. Everyone is simply a developer.
- No sub-teams. Just like there are no titles for the members, there can be no sub-teams such as a testing team. Not to be confused with multiple teams which is allowed and common in large-scale projects.
- All for one, one for all. Members might have specialized knowledge giving them natural areas of focus but the Development Team should always be accountable as a whole and never rely on individual expertise.

It's recommended that any Development Team consists of 3-9 members [10]. The reason being that too few developers limits the interaction and the team is more likely to have skill constraints. To many members lead to too much complexity that can make coordination between the members difficult.

Scrum Master

Deciding as a company to use a methodology and then explaining how the methodology works to all employees does not mean that the company will be developing according to that methodology. This is where the Scrum Master comes in. The Scrum Master is in charge of keeping the Scrum wheels turning. He/she makes sure that the Product Owner and Development Team are collaborating and generally doing what they are supposed to do. The following is a summarization of the responsibilities of the Scrum Master:

- Remove barriers regarding development so that the Product Owner is free to directly drive development forward.
- Teach the Product Owner how to maximize Return-On-Investment (ROI) in order to meet his/her milestones using Scrum.
- Improve the quality of life for the Development Team by empowering them and encouraging creativity.
- Assist the Development Team in increasing productivity in any way possible.
- Ensure that each increment of functionality is potentially shippable by improving engineering practices and tools.
- Give transparency to the Development Team's work progress for all parties and ensure it's up-to-date.

The position of the Scrum Master is normally filled by the project manager. Ken Schwaber indicates in his book, *Agile Project Management with Scrum*, that a Scrum Master should be a leader figure and never act as a regular boss [14].

3.1.2 Events

In Scrum there are five so called events. However the first event called Sprint is the most important one and all the other events are basically a part of the Sprint.

Sprint

As seen in Figure 3.1 the Sprint is very central in Scrum. The Sprint is a short period of time, usually a couple of weeks up to a month [14]. As soon as one Sprint ends, another starts. The Sprint is about dividing the entire project into small iterations. The length of a Sprint should be small enough to avoid complexity but large enough to avoid too much project overhead. The following four practices are all parts of the Sprint and describes more in detail what the Sprint is all about.

Sprint Planning

Each newly starting Sprint is initialized with a Sprint Planning. The entire Scrum team should be present during this planning. The Product Owner brings up the Product Backlog and its priorities. These are discussed by all parts and the Development Team goes through what will be achievable for the next Sprint. So to summarize the planning is about deciding what should be done and what can be done for the coming Sprint. The meeting usually has a time limit of around eight hours [14].

Daily Scrum

Also referred to as daily standup as seen in Figure 3.1. This is something that the Development Team does every day and should take around 15 minutes [14]. They do this to synchronize and coordinate their work efforts. There are three important questions to be answered during this meeting which are: What work have each member done since the last Daily Scrum? What is everyone planning on doing before the next Daily Scrum? What hinders exist that stand in the way of the team meeting the current Sprint goals? It's the Scrum Master's job to enforce that the Daily Scrum takes place and that it's only the members of the Development Team that participate [10] [14].

Sprint Review

The Sprint Review is held close to the end of a Sprint. The entire Scrum team should be present and if possible the stakeholders as well. This is supposed to be an informal meeting where the added functionality from the latest Sprint is shown. The Development Team shows what they have done and the Product Owner shows what has happened to the Backlog. The idea is to have a revised Product Backlog at the end that can be used for the next Sprint. The durations of this meeting is usually limited to four hours [14].

Sprint Retrospective

The Sprint Retrospective should take place after the Sprint Review but before the upcoming Sprint Planning. Once again the entire Scrum team is present, but without the stakeholders. This meeting should take about three hours [14]. The focus of this meeting is to improve the work process during a Sprint. The team reflects about things like tools, relationships, processes and anything else that can improve the workflow of the team and the next Sprint.

3.1.3 Artifacts

The three artifacts are the most basic parts in Scrum that are worked on during the different events.

Product Backlog

The Product Backlog is the dynamic requirements list in Scrum. It's an item list comprising of anything that can add any form of value to the product. Typical items in a Product Backlog are new features, bugs, enhancements and functions. Each Product Backlog item also has certain attributes such as a description, priority order, complexity estimate and value estimate.

What separates a Product Backlog from a simple requirements list is that it's a dynamic process and not a static list. The items in the Product Backlog that are to be considered for the "near future", as in having a high priority order, have more details than other items. This is to allow and encourage changes to the distant Product Backlog items as the actual product develops. The Product Backlog is in focus for both the Sprint Planning and Sprint Review. The refinement of the Product Backlog items is sometimes counted as a practice by itself as seen in Figure 3.1. However the items can be tuned and refined at any time. All members of the Scrum team have a saying when it comes to the Product Backlog but the Product Owner is the main responsible person.

Sprint Backlog

The Sprint Backlog is the selected items from the Product Backlog for the ongoing Sprint. It's the list of work to be done during the Sprint. All items in this backlog have clear details since the idea is to have them all completed at the end of the Sprint, which is a relatively short time period. The Sprint Backlog changes during a Sprint as the items become implemented and moved to "done". New items can also be added as unforeseen needed functionality is identified. It's the Development Team that works with the Sprint Backlog and uses it as their progress report and todo list for the current Sprint. At each Daily Scrum the Sprint Backlog is updated and refined with informations such as estimated remaining work hours.

Product Increment

A Product Increment is the sum of all implemented Sprint Backlog items for a Sprint. It's a sort of "next step" or version for the product. The requirements for an item to be counted in the Product Increment is that it's fully usable for the product, regardless if the Product Owner chooses to release it or not.

3.2 Extreme Programming (XP)

Kent Beck is the founder of Extreme Programming (XP). He is also one of the members of the Agile Alliance that was described at section 2.2 The Agile Manifesto. XP was conceived shortly before the creation of the Agile Alliance. In 2000, only a year before the Agile Alliance surfaced, the first book about XP called *Extreme Programming Explained: Embrace Change* was published by Kent [9]. For this reason, when reading early literature about XP their is no mentioning of the term *agile*, which is not strange since at the time the term did not exist. However more published articles and books have surfaced after the year of 2001 that have incorporated the agile ideology, such as the second edition of Kent's first book released in 2005 [9].

Unlike Scrum, XP almost solely talk about the developers and helping them be as efficient as possible. It does not talk much about the overall business outside the development team. Although if you read further about XP you will find business related information but it's not at all in focus. XP put emphasis on team work, feedback and code quality. The term extreme comes from the idea that it takes a few ordinary development practices and impose them to an extreme. Originally XP was designed for small to medium sized teams but has over time grown to incorporate large teams as well.

There are a total of eleven main practices in XP that will be shortly described in the following sections. There are several corollary practices as well that talk more about the business and goes even further into coding details, which are not brought up in this report. To read about these and much more, the second edition of *Extreme Programming Explained: Embrace Change* is recommended [9].

3.2.1 Fine-Scale Feedback

The following practices describes how the members of the team develop their software with feedback in focus. It goes through the basics of how they work in pairs and always use Test-Driven Development (TTD) to guide them. It also describes the relations between business people and developers.

Pair Programming

This is perhaps the most notable practice regarding XP. Pair Programming means that for each developer, there is a another developer observing and assisting without

coding. They work together similar as in rally racing where one person is the driver and one is the map reader. The main difference is that in Pair Programming the coders switch places regularly. The purpose is that the observing developer can have their mind set on a higher abstraction level and is not distracted by coding, making it easier to ensure high code quality [1].

Not only are bugs detected more easily but the overall resulting code is simply put better. This way of developing can be quite motivating for the developers since you are never alone and if you grow tired of typing you just switch places with your pair. If the developers also switch partners every other day, information and knowledge is spread rapidly and smoothly within the entire team. This is especially good when you have specialists with knowledge that could benefit the other developers. Studies conducted by Williams and Nosek has suggested that pairing does not reduce the development efficiency but does show a decrease in defect rate in the software [8].

Planning Game

The Planning Game is played between the business people and the developers. It's about figuring out and deciding witch features to implement and basically form a prioritization of what needs to be done. The business people focuses on the importance of each feature [1]. The developers focuses on the cost of developing the features.

A simple and natural way to play the game is: After each iteration, which for example might be a two week period, the developers updates the cost of future features. They also come up with a budget for how much they will be able to develop for the next iteration. Then it's the business people's turn. They decide the features, within the budget, that they want the developers to work on for the next iteration. This game plays out after each iteration.

In reality it's much more complicated and a whole chapter is dedicated to the Planning Game in the book *Agile Principles, Patterns, and Practices in C#* by Martin C. Robert and Martin Micah [1]. Things like velocity and user stories are described in detail which are all important components to the Planning Game. What is stressed is transparency since everybody might not be up to play ball just relying on some developers estimation.

Test-Driven Development

The following steps describes TTD:

1. Write a unit test for some functionality.
2. Make sure it fails before implementing it, since it would be strange otherwise.
3. Write the code that implements the functionality.
4. Run the unit test and if it fails, fix it until the test passes.

5. Repeat for every future functionality.

This can seem overwhelming at first but it leads to having a close to 100 percent test coverage which can be very beneficial as the project grows. Another positive aspect is that writing test cases before coding leads to all code being testable by definition. Some may think: *Isn't all code testable?* As soon as you build a system without the slightest thought about testability you will get the answer. Further more, by using TTD you tend to get code that is clean, less coupled and better structured [1]. The reason for this is simply because your mind is set to create code that can be tested independently, indirectly leading to decoupled module like structure.

Whole Team

Saying that XP focuses on the entire team does not simply mean the entire development team. Everyone that is connected to the XP project should be collaborating closely with each other to solve problems. This goes as far as preferably having the customer in the same room or within 100 meters from the developers [1]. The more time the customer can spend with the team and the shorter distance the customer is from the team the better. If the customer is unable to be within close vicinity then it's advisable to have a stand in representant for the customer.

3.2.2 Continuous Process

These practices describes the cycles such as sprints and release plans for the project. Things like frequent integration are brought up and the importance of continuous refactoring.

Continuous Integration

For those that frequently work with multiple people using Git, Continuous Integration will sound familiar. Each programmer should integrate their work several times a day. If you are lucky (or fast), you are first and simply check in your work. If you are not so lucky, you have to merge the changes. In other words, XP teams use nonblocking source control [1]. This means that every developer can work on any part of the system, simultaneously with the other developers. It's important that each system test and acceptance test are run before anything is checked in. If something is broken, the developer fixes it before checking in the changes.

Refactoring

Refactoring is not a backup practice when quality ensuring practices fail. Refactoring is a quality ensuring practice. Sometimes it can be hard to convince non-programmers the benefits of refactoring. The main reason is that feature wise, nothing is added. At most some optimization might be noticeable.

Refactoring is about changing the foundation of the software. It's to make adding new features a simpler task. Code duplication is a sign that refactoring is needed. When developers start to choose bad coding practices simply because they see no other way of implementing something, refactoring is needed. When changing code affects unrelated code sections, refactoring is needed. However the idea is not to code poorly, fix it with refactoring and continue. Most developers knows that coming to a situation where refactoring is advisable is pretty much unavoidable, the important thing is not to add fuel to the flames. Pair Programming for instance seeks to diminish the need of refactoring and make sure quality is ensured [9].

In XP refactoring is not something you do only at the end of an iteration or cycle, not even something you only do at the end of the day. You do it as soon as you see that it's needed. This means that refactoring will come in really small packages that alone do not bring much value. Together however, all the refactoring keep the clone clean, simple and as expressive as possible.

Small Releases

Small Releases means release often and release small. Releasing software can be a pain but that is usually the case when you seldom release, thus having no routine for it. The releases in XP can be split into two types. The iteration plan and the release plan:

The iteration plan
: This is a sort of minor release. In XP the usual interval for minor releases of new software is every two weeks [1]. However as short as possible is preferred. The two week period is just a well working standard. At the end of each iteration cycle, decisions regarding user stories and features should be made according to the the Planning Game. It's not obligatory that something gets released at the end of this cycle since the short time period might mean that too few features with value have been implemented.

The release plan
: This is basically the bigger version of the iteration plan. This is a more abstract plan that usually spans around three months [1]. A clear difference is that the iteration plan should not change but the release plan is subject to change during its lifetime. A simple explanation for this is that it's expected to foresee implementation issues two weeks ahead while this is not the case for three months. It's also expected that something gets released at the end of a release plan cycle.

3.2.3 Shared Understanding

The following practices are about giving the developers a team feeling and shared understanding of the entire project. The practices talk about code design, abstraction and that sharing knowledge is caring.

Coding Standards

Pair Programming explained how knowledge can be spread swiftly by routinely changing programming partner. For this to work well it's important to have a coding standard so that the programmers don't spend time trying to understand each others semantics and syntax. Without a coding standard, whenever a programmer starts working on someone else's work, he/she might feel the need to format or even refactor the code.

When setting a coding standard it's important with proper communication so that every developer feels comfortable with the standard and voluntarily wants to uphold it [9]. There is really no reason not to have a coding standard since it will make the software easier to understand which indirectly increases code quality.

Collective Code Ownership

There will always be developers with different specialities. The common way of just working on ones speciality means being more locked to that speciality. XP instead seeks to minimize this speciality difference between the members. Both Pair Programming and Coding Standards helps in evening out the specialities between team members.

Everybody can choose to work on any part of the software such as on the database, the Graphical User Interface (GUI) or the server. As an XP developer you are not confined to a specialty and assistance should always be offered to those who seek to broaden their expertise by coding on unfamiliar components of the software [1]. This is to increase motivation, reduce risks by spreading knowledge and aid the personal development of the developers.

Simple Design

XP seeks to keep it simple and expressive as stated by the practice Refactoring. The infrastructure is not in place when developing according to XP. The infrastructure gets created when it's currently needed. There are three so called mantras that an XP developer follows regarding simple design:

Consider the simplest thing that could possibly work

When developers are about to implement a new user story. They think about the simplest possible scenario that satisfies the story. Then they choose the simplest possible practical way of making that scenario a reality. This means that XP developers do not think about future user stories when implementing the current story.

You aren't going to need it

Sometimes you just know that a database will be needed in the future or that a web server is going to be created. In that case, is it not important to make sure the code currently being implemented is easily hooked to these services even if the services don't exist yet? The answer is maybe. The developers only do it

if they are absolutely certain that the service will come in the future together with the conclusion that creating hooks for this service, before it exists, will be more valuable than creating them later.

Once and only once

If duplicate code exists, make sure it doesn't. The best way to remove duplication is creating abstraction [1]. For example if a couple of code parts do almost the same thing, use templates. The important thing is that code duplication is removed as soon as it's seen.

System Metaphor

This is the most vague practice in XP. In the book *Agile Principles, Patterns, and Practices in C#*, Martin C. Robert and Martin Micah describes the System Metaphor by describing a jigsaw puzzle [1].

How do one know what pieces go together in a jigsaw puzzle? A blind person could only go by the shapes and try to fit pieces that seem to go together. A more powerful evidence that the pieces go together is the image. If the image seems correct, we don't have to look at the shape, we just know they somehow go together. More importantly, if the pieces do not go together even though the image is correct we know for sure that the puzzle maker has failed with the pieces. A blind person would have a much harder time coming to this conclusion.

In other words, the metaphor correlates to how important it's to know the big pictures in order to make sure that the modules or components are what we really want them to be. No offense meant to any blind person out there. Also note that the description of the System Metaphor using the jigsaw puzzle is a metaphor by itself, mind blown! By using metaphors we can create an abstraction, or the big picture, for complicated software or systems. When using metaphors you often use a system of names for different elements that on a high level describes the overall mechanisms.

The metaphor is the vision for a system. It's the guide for the developers that help them name classes and functions, select appropriate locations for new components and overall helps ensuring that everyone knows where, what and why they are currently implementing something.

3.2.4 Programmer Welfare

This section only has one practice where the focus is on the wellbeing of the developers. The practice Sustainable Pace is about avoiding burning out the developers with stress and overtime.

Sustainable Pace

Like a marathon runner must conserve his/her energy to not burn out before the race is finished, the development team in XP must keep a Sustainable Pace. A software

project takes a long time to finish and sprinting from start comes with several drawbacks. It can affect motivation negatively in the long run. Estimations becomes wrongfully optimistic as time passes. Also working extra hard from start, when you have minimum knowledge of the heading of a project, can lead to redundant work.

Therefor in XP you are not allowed to work overtime with the only exception of the final week before a big release [1]. The big releases usually comes once every third month like stated by the practice Small Releases. That does not mean that the practice is to always work overtime before every release, it simply means that it's allowed if a team happens to be extremely close to reaching an important release goal in the final moments before release. Making sure that everyone works in a Sustainable Pace also have the effect of conserving energy so the team is alert for that seldom occurrence when an extra push is needed.

3.3 Crystal Clear

Crystal Clear is an agile development methodology developed by Alistair Cockburn, one of the authors of The Agile Manifesto. Her role in the creation of The Agile Manifesto was to focus on efficiency rather than handling rapidly changing requirements and thus Crystal Clear has focus on efficiency, similar to XP [5]. Just like agile development can be seen as a family of methodologies, Crystal is a family of methodologies. Some of the other variations that exists are Crystal Yellow, Orange, Red, Maroon, Blue and Violet. The darker the color, the bigger project/team it's suppose to handle. Crystal Red is for example for a project consisting of 50-100 people. Crystal Clear, like the name clearly suggests, is designed for the smallest project consisting of 5-8 people. It's for that reason that Crystal Clear was the chosen sub methodology for this report since we seek to find agile methodologies suitable for small-scale projects.

Crystal Clear has seven properties that stand as the practices of the methodology. The first three properties, Frequent Delivery, Reflective Improvement and Osmotic Communication are the only required properties and the last four are recommended, especially for experienced teams. For a more detailed description of Crystal Clear read the book *Crystal Clear* created by Alistair Cockburn himself [5]. The book is mainly based on expert interviews of development teams and project managers. The following descriptions of principles are taken from that book if no other reference is mentioned.

3.3.1 Frequent Delivery

It's important to have a strict delivery frequency and not fall into the temptation of extending deadlines since it can be demotivating for the team and jeopardize the future schedule. By ensuring that the team deliver new software on a regular basis the team learns how much it's able to deliver and can improve estimations. This also gives the developers a feeling of early victory.

Frequent delivery is not only for the developers, it also aims to satisfy the customer by giving clear progress reports in the form of actual software. The delivery frequency is usually about 2 weeks but depending on the project and development team this can be either extended or shortened.

3.3.2 Reflective Improvement

> Did you get together at least once within the last three months for a half hour, hour or half day to compare notes, reflect, discuss your group's working habits and discover what speeds you up, what slows you down, and what you might be able to improve?
>
> Alistar Cockburn [5]

This is similar to what was mentioned at chapter 2 Agile Development in the first paragraph. If you never stop, reflect and evaluate your work you don't know if what you are doing is really working. Agile development is about being open to changes but you have to know the worth of those changes. It does not have to be statistical assessment, it can be something as simple as just sitting down with some colleagues over a cup of coffee and discuss what is working and what is not. The important thing is to make it happen by allowing the team to do this on a regular basis. Crystal Clear is a lot about the social environment and trusts the team to make the right choices when given the right circumstances. Reflective Improvement is important because it allows the team to evaluate their decisions and improve for the future.

3.3.3 Osmotic Communication

This is an important principle for Crystal Clear. In fact, it's so important that it's the only principle that exist in all methodologies within the Crystal Family. Osmotic Communication is the effect of having a team physically close together and picking up important information in the background chatter. So individuals holding lots of important information will, either consciously or unconsciously, gradually spread it out which is similar to the effect of osmosis. A simple way of ensuring osmotic communication is simply grouping the team in one single room, or the so called "war room", whose name might inspire people resisting leaving their private offices. For large teams this can be an logistic issue but an alternative is to have several rooms next to each other forming sub teams.

3.3.4 Personal Safety

This refers to something that most agree on is important but is generally hard to establish. To be able to give critique at any level of the company or project chain. In order to allow this, the most important thing is to somehow remove the fear of reprisal. The members must feel safe and have trust in one another in order have

the confidence to bring out weaknesses and problems in the project. Especially when weaknesses goes against the word of project leaders or managers. This means that it's important with good leadership to ensure Personal Safety. Simon Sinek, who is a leadership expert, stresses and explains this importance well in a TED Talk [6]. There he makes an example of how the militaries focus on great leadership influence the soldiers and draws a parallel on the positive effects this would have on companies. In short terms, he brings up the hard to measure positive effects with having leaders truly and fully trusting the team and the team truly and fully trusting their leaders.

A way of ensuring trust, that leads to *Personal Safety*, between leaders and the members is to let the leaders have private discussions with each member. During these discussions the leaders try to make the member expose weaknesses, such as a failure to reach a deadline or any form of work-related information that they don't feel comfortable exposing. An experienced leader can then show a positive attitude and even gratefulness for the information. Furthermore, the leader can cover for the member and offer assistance to demonstrate that when the member revels a weakness or mistake, he/she will actually get assistance. Another tool is to have meetings where difficult problems are discussed where all members of the team are present, including the leader(s). This can lead to heavy argumentation, opening up to critique and hopefully showing that they can solve problems together that would not be easily possible alone.

3.3.5 Focus

> Do all the people know what their top two priority items to work on are? Are they guaranteed at least two days in a row and two uninterrupted hours each day to work on them?
>
> Alistar Cockburn [5]

As a developer, at some point you have come across the situation on working on some functionality, detecting some side problem or challenge, spending lots of time working on it, and finally asked yourself the question: *What am I doing?* As a programmer it's easy to find yourself in this situation of spending valuable time on non-valuable things. It's mainly the executive sponsors job to make sure that this happens as little as possible and that the developers stay on the correct course, speaking about value to the project or company.

When working on several projects as a developer, it takes time to mentally switch between one project to another. Think of how a computer reacts to cache misses. Inexperienced managers underestimate this cost and keep assigning new projects to the same developers before previous projects are finished. This is not only overwhelming to the developers, it's also motivationally destructive to regularly report the lack of progress on assigned projects. A simple repair is that the executive sponsor makes a clear prioritization list for each individual. For example prioritizing two projects or items much higher than the rest.

Focus time for the team members is also something important. It can be as simple as guaranteeing two full work days on a project before being allowed to switch, or even looking at another project. This limits the overhead of switching between projects, thus guaranteeing some progress to be made before any switch to another project is made. Another level of practically ensuring focus time is to have a couple of hours each day where interruptions are forbidden, with the exception of certain critical scenarios. This can be very important because just like how project switching leads to overhead time waste, frequent interruptions from colleagues can create the same idle time. Especially when the interruptions are about matters that don't directly involve what the developer is currently working on.

3.3.6 Easy Access to Expert Users

> Does it take less than three days, on the average, from when you come up with a question about system usage to when an expert user answers the question? Can you get the answer in a few hours?
>
> Alistar Cockburn [5]

Expert users can provide qualitative feedback regarding functionality and design and are generally a great source of information for the developers. Easy access to expert users is a good addition to Frequent Deliveries since it allows rapid and valuable feedback when deploying and testing newly added features. A study conducted in 1995, by Keil and Carmel showed the importance of having direct links to expert users and the positive effects it could have on a project [7]. Their research led to the recommendation to "Reduce Reliance on Indirect Links", as in rely more on access to real users. If a programmer has a design question and it takes several days before a reply comes from an expert user, it can lead to delays or the programmer making decisions based on his own assumptions. This time delay between questions and answers can have grave impacts on the project. Some practical ways of ensuring Easy Access to Expert Users are:

Weekly or semi weekly user meetings with additional phone calls
: Pretty straight forward, allow communication to flow freely between expert users and developers with regular weekly meetings. It can be as little as one or two hours per week. It's advisable to have the meetings more frequent in the early phases of the project. The availability of phone calls is a good addition for those extra important questions that needs a quick reply.

Having experienced users directly on the development team
: This can be hard to achieve and is rarely used since it highly depends on the availability of the expert user. However, if it's possible it's perhaps the optimal way of providing expert user feedback.

Send the developers to become trainee users for a period

This is sort of converting developers to expert users. This leads to the developers seeing the project from a different perspective and can limit the need of having regular feedback from actual expert users.

3.3.7 Technical Environment with Automated Tests, Configuration Management & Frequent Integration

Neither of these technical environment elements are crucial for a successful project but they can make the developers lives easier.

Automated Testing

Many projects do just fine using manual testing. However every programmer, that had switched from manual testing to automated testing, that Alistair Cockburn interviewed swore to *never to work without them again.* The benefits are apparent when several people work on the same code on different days. The next developer can just run the automated tests the day he/she starts and be confident that nothing is broken. It gives the developers a freedom to change without worrying about whether or not something got broken along the way. For those with no prior experience with automated testing, the X-unit framework is advisable to take a look at.

Configuration Management

The configuration management system gives a structure to the project that allows asynchronous work, to revert changes, to select a specific configuration for release and backup to a stored stable configuration when problem arises. You can loosely say that the configuration management system is to an entire project as Git is to software development. According to Alistair Cockburn, configuration management is often cited by development teams to be the most critical non-compiler tool.

Frequent Integration.

Frequent integration leads to easier detection of bugs and also fewer bugs/problems that surface simultaneously. The added code is more fresh in the developers minds and smaller in size, thus leading to fewer hiccups when checking the code to be integrated. It's advisable to integrate as frequent as possible. It's good to be able to integrate several times a days. If that is not realistically possible then once every day is not bad either. When several days passes by before integrating the problems starts to stack up.

3.4 Kanban

When reading about Kanban it can get quite confusing. This is because Kanban is originally a lean scheduling system for Just-In-Time (JIT) production developed by Taiichi Ohno at Toyota in 1953 [11]. It's still used today within manufacturing but the ideas have been taken and modified to fit software development. The confusing part is that this software development version of Kanban is called Kanban. If not clearly stated otherwise, future referencing to *Kanban* in the report refers to the software development version of Kanban. A fun fact is that "kan-ban" is Japanese for "signal card" [11]. Some disagreement regarding Kanban is weather it's more of a lean or more of an agile methodology. The easy answer to this is to simply ignore the question since lean and agile methodologies don't go against each other nor have precise definitions as explained in section 2.5 Agile vs Lean. One thing most can agree on is that Kanban is both lean and agile to some extent.

As David J. Andersson stated in his first book about Kanban, the methodology is lightweight and does not have a clear formal definition in order to promote changes and modifications [11]. This has led to several alternative methodologies, that build upon the original, popping up such as Kanban Ace and LKU Kanban. For these reasons, it's not that strange that the core practices can vary slightly in both numbers and definition. Depending on where you read you might find three or seven practices. For this report the chosen core practices are taken partly from the book *Kanban: Successful Evolutionary Change for Your Technology Business* by David J. Andersson [11], and partly from the overall consensus from various resources available online.

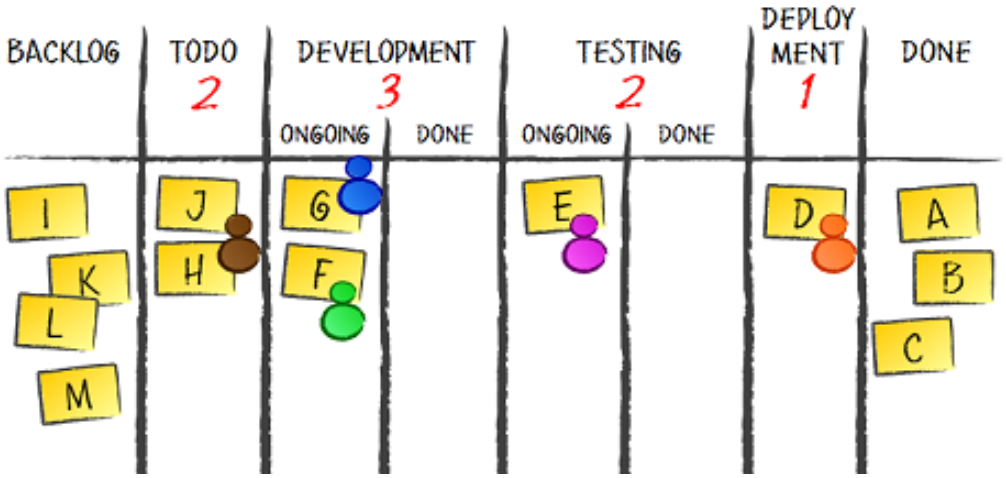


Figure 3.2 – Image showing the basics of a Kanban board. The cards move from left to right. The number under each category represent the Work-In-Progress (WIP) limit.
Source: Pawel Brodzinski, http://brodzinski.com. Pawel has given his consent to reuse his image.

3.4.1 Visualize

This practice is about visualizing workflow. To have transparency between developers, managers and anyone else connected to a project. The standard way of doing this is using a so called Kanban board with cards, see Figure 3.2. This board can either be a physical board with sticky notes, or a virtual board. This board is a visual control system for all the tasks within a project. The board allows for self-organizing by just pulling a new available card after finishing a task instead of waiting for a superior to supply the tasks.

The cards are not only meant to simply hold programming tasks. They are meant to represent some customer value. Therefor the design of the cards is important in order for the developers to make easier decisions on what to work on next. There are no exact rules on what information the cards should hold but a title, description, estimated size and some estimated value number is a good start [11].

3.4.2 Limit Work-in-Progress

Limiting WIP is about having a limit at each step in the product chain, shown in Figure 3.2, that corresponds to the amount of tasks allowed in that step. A simple WIP limit is having it equal to the amount of available working people at each step. When deciding the WIP limit, all members from from the stakeholders to the developers should take part and form an agreement [11].

The WIP limit is important for several reasons. It shows when a pull from one step can be made to the next. It ensures focus on one or a few tasks at a time. It keeps the workload balanced and avoids stock piling of items at some middle step in the product chain. The most important benefit is that it gives the development team a limit to rely on and fall back to when put under too much stress, especially if everyone participated and formed a consensus when choosing WIP limit.

3.4.3 Manage Flow

In Kanban the workflow or task flow is central so it makes sense having a practice focus on how to manage and improve it. In order to improve the workflow one must first be able to measure it somehow. Otherwise when changes are made, how do we know that it was an improvement? Some things are easier to measure and improve such as items stock piling at a certain step. An easy fix would be to add more manpower to that particular step. Some software such as *Kanbanery* offer automatic tools for measuring that can give useful information about how changes affect the workflow [12]. Something important to stress out is that the more precise the estimated complexities and values for the cards/tasks are, the easier it gets to analyze the workflow.

3.4.4 Make Process Policies Explicit

In the practice Visualize the importance of having a clear design on the cards of the Kanban board was mentioned. This practice is about just that. Without clear rules and a common understanding of how the card system works the discussions becomes subjective and emotional instead of factual [13]. Having the Process Policies explicit makes them easier to understand and therefor easier to follow. They help choices regarding the process to become more rational and empirical, and the team have an easier time reaching consensus simply because everyone is working from the same policy configuration.

The Process Policies are the rules and procedures of the project, with focus on the Kanban board. It can state the type and amount of data that each card should contain. It explains the flow of cards traveling across the board. It can contain default implementation details about different classes of cards. The Process Policies basically defines the structure of the Kanban board and its cards.

3.4.5 Improve Collaboratively

> If you are not continually improving, but you are doing all of the other parts of the Kanban method, you are missing the point. It's a little like the concept of "doing" agile but not being agile.
>
> Matthew Powell [13]

This practice is more social and less practically oriented than the others. When reading about Kanban you sometimes stumble upon the word *Kaizen* meaning continuous improvement. *Kaizen culture* is also often mentioned that talks about empowering the workforce and encourage them to "do the right thing". Kaizen culture promotes high levels of collaborations and freedom to self-organize work. It's about putting the company and business first and this requires lots of trust which Simon Sinek talks about in his TED Talk about leadership [6].

Kanban promotes using a scientific approach when trying to introduce new changes. That means to evaluate a situation, suggest suitable change, predict the outcome, observe the results and compare with the prediction and previous results. If the team or business have a Kaizan culture together with using the scientific approach, then that can lead to spontaneous and regular local improvements to the process [11].

3.5 Other Mentionable Methodologies

The informed reader might wonder why only these four methodologies were chosen for the report. After all there are at least a dozen more agile methodologies worth mentioning. For instance Dynamic Systems Development Method (DSDM) that is a broadly used agile methodology.

The reason for the chosen methodologies are several. First of all we must acknowledge that going through every possible agile methodology was simply not possible. Kanban was chosen since it's both broadly used and very lightweight, making it a good candidate for small-scale projects. Crystal Clear was chosen as it clearly tries to address small development teams in its definition and there are not many agile methodologies that does so. Scrum is the most popular agile methodology, for the time being, and therefor deserves a spot. XP was chosen since it's popular but still quite different from the other methodologies, making it interesting for comparison reasons.

DSDM was skipped simply because its scope is too large, larger than both Scrum and XP. This makes it impractical for this report since it talks about business cases and sponsorship which might not be the number one priority for small-scale projects. Feature-Driven Development (FDD) is another popular methodology but it has a clear focus on large projects by discussing things like automated build systems. Some other methodologies are Adaptive Software Development (ASD), Scaled Agile Framework (SAFe), Disciplined Agile Delivery (DAD), Projects in Controlled Environments (PRINCE2) but they were all skipped for similar reasons, or lack of documentation.

Chapter 4

Methodology Evaluation

This chapter aims to theoretically compare the four agile methodologies to each other with focus on small-scale projects. At the end one methodology is selected to be used for the project, see section 5.1 Project Description. For the reader that skipped some parts of the prior chapter, it might be interesting to read section 3.5 Other Mentionable Methodologies that explains how the four methodologies were chosen for this report.

There is no exact definition of what a *small-scale project*, *small team* or *small company* means. For the sake of clarification and consistency it's assumed that a small-scale project consists of 1-5 member(s). They can either all be part of the development team or one or two having other responsibilities such as being a costumer, project leader or product owner. We further assume that the small team is within a small IT company and not some sort of small project team within a large established company. For example, it could be a handful of friends starting an IT consulting business after getting their master's degree in computer science. So whenever a small company, small-scale project, small team etcetera is mentioned from here on, this is the setup to have in mind.

4.1 The Agile Principles for Small Teams

The agile principles are the backbone of agile development. Therefor it's only natural to compare agile methodologies to these principles in order to estimate how agile they are and what their differences are. However the principles aim to satisfy large-scale projects and big teams since after all, the bigger the project the bigger the need for a methodology. Even if it's hard valuing the agile principles, or not even meant to by the founders and followers, it does make sense to reestimate their value for small-scale projects and small companies. The following evaluation of the principles is only based on logical reasoning and should therefor be read with a critical mindset. The principles are listed at section 2.3 The Agile Principles.

The first three principles regarding satisfying the customer, welcoming changing requirements and delivering software frequently should be equally important

no matter the size of the project. The fourth principle however, about business people and developers working together daily, is usally naturally satisfied since in small-scale projects they tend to be the same people. Meaning that the developer(s) also tend to the company business. The fifth principle lowers greatly in importance since it's clearly focused on larger companies where you can choose between different teams and people regarding what project they should work with. As a new small company you kind of have to build the project around the hopefully motivated individuals. The only time face-to-face communication, principle 6, would significantly decrease in importance would be when you work on a project completely alone. Principle 7, 8, 9 and 10 all talk about, either directly or indirectly, keeping a good structure and code quality high over time. The size of the project have no obvious impact in this regard. Principle 11, which is about self-organizing teams, is pretty much self-fulfilling in a small new company. The last principle, number 12, is about team reflection and should be important in any sized project.

To summarize, the 9 principles 1-3, 6-10 and 12 comes out as more important to satisfy than the rest for small-scale projects and small companies. Note that this does not mean that the other principles are less important agile principles overall. This short evaluation of the agile principles facilitated in keeping a better focus when evaluating the methodologies for small-scale projects.

4.2 Scrum

The practices of Scrum satisfy the agile principles well. You can almost match each practice in Scrum to each different agile principle and perhaps this is why Scrum is often used as the example methodology for agile development. The only principle that it does not address is principle 9 that talks about technical excellence and good design. The only thing remotely connected to good code design is getting tasks done before the end of each Sprint and not leaving loose uncompleted ends. This is talked about throughout the event practices seen at section 3.1.2 Events.

The methodology clearly revolves around the Sprints. It talks a lot about having short iterations together with several short meetings and ensuring that the Product Backlog is kept updated at all times. Also notable is that every element in Scrum is tied together as seen in Figure 3.1. Each practice is strongly connected to several of the others. Within the events all three roles, the Scrum Master, Product Owner and Development Team described in section 3.1.1 Roles, work together tightly and always have the artifacts in focus, see section 3.1.3 Artifacts. Generally this is a good thing since it shows coherence and makes it easier to grasp the meaning and logic behind each practice. However to be able to include all practices one would at least need three people to fill the roles. For the defined small-scale project in this report, that would mean a Development Team of maximum three which can be a problem. An approach to get around this issue would be to let everyone be a part of the Development Team where two of the developers have extended roles as a Product Owner and Scrum Master.

It's certainly possible to include all practices for a small-scale project but it would also most certainly lead to lots of overhead work when so few people are involved.

4.3 Extreme Programming (XP)

XP is quite opposite of Scrum and talks quite a lot about code quality. The practices Pair Programming, TTD, Coding Standards, Simple Design and Refactoring all emphasizes how they have a positive influence on code quality. At first glance one might believe that XP is much more "zoomed in" than Scrum and only talks about details regarding the development team's coding procedures. However, the practice called The Planning Game is very large and includes components that are similar to several practices in Scrum, such as iterations (Sprints), small stories (Product Backlog) and planning meetings (Sprint Planning). So one could also see XP as a sort of huge framework where Scrum is replaced by The Planning Game. So for companies using both Scrum and XP it's a matter of perspective weather they are using XP on top of Scrum or Scrum on top of XP.

Since The Planning Game together with Small Releases is loosely put another version of Scrum, XP satisfies the same practices that Scrum satisfies. The principle that Scrum did not satisfy, principle 9 regarding code quality, is quite clearly satisfied by many of the practices in XP. Seeing how focused XP is on code quality it's easy to understand the critique it receives regarding being called agile. After all, agile development is usually talked about in respect to cross-functional teams, short planning and regular meetings across the entire company hierarchy.

XP as a whole is quite clearly too extensive for small-scale projects. They do however have many stand-alone practices that would be easy to start with and then slowly built upon, practice by practice. For example, all the quality ensuring practices mentioned. On the other hand, how agile would it be to have a bunch of practices satisfying only one agile principle? To ensure that the other parts of the agile principles are included one would have to look into the Planning Game. But instead of diving into that practice it would make more sense starting with Scrum where the different parts of The Planning Game are more clearly distinguishable.

4.4 Crystal Clear

Ironically the practices of Crystal Clear is the most vague of all four methodologies. They have chosen a higher abstraction on their practices, somewhere in between the agile principles and the practices of the other methodologies. Instead of describing actual tools to achieve positive outcomes it focuses on describing positive outcomes with tools as examples. Half the practices, Osmotic Communication, Personal Safety and Focus, all talk about social behavior in group environments. The methodology revolves around building safety and trust within the team. In other words, it has a clear focus on the human mind and communication.

Since it has this high abstraction level it's harder to compare the practices to the agile principles. However the practices line of argument means that the more socially oriented principles 4, 5, 6 and 8 are satisfied or are at least are in focus.

Crystal Clear is clearly more lightweight than XP and Scrum which naturally means it would be an easier methodology to follow for small-scale projects. To try to establish a sort of trust oriented culture early in a company, Crystal Clear is certainly a good candidate to follow. However it somewhat lacks practical practices to follow so it might be a good idea to complement it using another agile methodology or several agile strategies. The methodologies Frequent Delivery and Reflective Improvement is basically the Sprint and Sprint Retrospective which means that Scrum could be a good candidate to pair together with Crystal Clear.

The last principle seen at section 3.3.7 Technical Environment with Automated Tests, Configuration Management & Frequent Integration, is certainly more practical than the others within Crystal Clear. However it feels forced and misplaced, both in terms of content and title naming. The way it's described makes it sound more like general good coding tips than a practice within Crystal Clear. The reason for this might have something to do with the fact that Crystal Clear is a methodology within a collection of Crystal methodologies, as explained in section 3.3 Crystal Clear. Studying the entire Crystal family might have given more insight into that particular practice.

One thing you can safely say is that Crystal Clear put heavy emphasize on the part of the The Agile Manifesto that states: *Individuals and interactions* over processes and tools.

4.5 Kanban

Kanban is the most lightweight of the four methodologies. Like Scrum has a clear focus on Sprints, Kanban has a clear focus on the Kanban board. Only one of the five practices does not directly revolve around the Kanban board, the practice called Improve Collaboratively. It's so lightweight that it's arguable that Kanban is more of an agile/lean strategy than a complete methodology. So it's not that strange that it's often mentioned, when reading literature about Kanban, that the methodology is meant to be expanded as time progresses and the project/company grows.

Kanban certainly does not fully satisfy all the agile principles but the practices do nibble on almost all of them. In Scrum, each of the principles 1-3 can be clearly linked to a practice such as for example the Sprint Review, Product Backlog and Sprint. In Kanban these principles are all linked through the Kanban board but in a more subtle way. Like with the other methodologies, except XP, Kanban has no strong focus on code quality. Also, even if the Kanban board is about visualizing workflow for every member of a project, it does not talk much about the collaboration between business people and developers. To summarize the principles 4 and 9 are left unsatisfied. Other than that, Kanban seems to do have a covering but broad spread when it comes to satisfying the agile principles.

The Kanban board is easy to understand and easy to implement. It should give value even if you are developing completely alone since it provides structure and abstraction for the developer. It also offers transparency for anyone involved in a project meaning that the development process should be easy to follow. The freedom of details on the Kanban board and its cards, with the use of the Process Policies, means that it would be easy to start small and slowly buildup the organization of the board.

Kanban seems like a suitable choice to start with for a small company looking for an agile methodology. It does a decent job covering the agile principles, it's relatively lightweight and most of the practices are easy to practically follow. However as a company/project grows it can perhaps become too lightweight and should perhaps be expanded or complemented using agile strategies or another methodology. An example of this is the methodology Scrum-ban which is a combination of Scrum and Kanban.

4.6 Chosen Methodology

Kanban became the chosen methodology for the project, see section 5.1 Project Description. There are several reasons for the methodology to be favorable for the project. The methodology tries to cover most of the agile principles but is extremely lightweight compared to Scrum and especially XP. For a small-scale project this is essential since the smaller the project team the larger the overhead for a methodology. And in this particular project there will be only one developer together with the product owner. Moreover, the fact that Kanban is meant to be expanded on demand, and that methodologies like Scrum-ban exists, means that it should be easy to let the methodology grow as the project/company grows.

Another reason for choosing Kanban was because of the fact that initially the project description was rather unclear and unplanned. Both the features to be implemented and the time it would take to implement them were uncertain. All of these unknown parameters meant that the ability to change, as in flexibility, was an important factor. Since Kanban does not strictly use iterations, such as Sprints in Scrum, it's easy to quickly and continuously change the requirements on a moments notice, which was seen as advantageous for a project with a timespan of only a few months.

Regarding Crystal Clear, the focus on social environment makes it a pretty poor choice when working solo as a developer. Even for a handful of developers it can be hard to practically follow such abstract ideas. The methodology also does not have the same apparent coherency for its practices as the other methodologies which makes it feel fuzzy.

Chapter 5

The Project

This chapter has four sections. The first is a short section that explains what the project initially was all about. The second section goes through the workflow of the project that has been developed using Kanban, and at the time of this reports writing development is still ongoing. The third section goes through the practical results of the project. The final section is a short interview conducted with the product owner.

5.1 Project Description

This section is written in present tense to emphasize that it was the initial description for the project before any work had been commenced. Note that this is a real project and not just an experiment for this report.

The project, to be developed using agile strategies, is a private initiative from Per-Arne Forsberg who has 25 years of successful international experience and knowledge in managing technically skilled units, products and projects around the world at Ericsson. Having been involved and engaged in change management, using the latest lean and agile methodology.

The project is developing a dynamic, scalable and robust interactive tutoring framework prototype for pre-academic students. Assisting students today according to the "old school" is social but also ineffective since it's usually one to one communication or one to a few in some geographical location (you have to meet at pre-determined location at pre-determined times, like classrooms). The long term goal is to assist in communication between students and teachers (tutors willing to help out with their knowledge from their preferred location) in Sweden and in turn combat the negative (knowledge) grade curve that can be seen around the country, according to international studies, PISA.

In short the idea is to create an interactive website that can be used for many to many type of communication between students and teachers. An exact specification of the project does not exist in order to encourage the agile development process.

5.2 Working Agile

This section goes though how agile development techniques were utilized for the project, especially how Kanban was used. It tries to follow a natural flow but does not follow a strict timeline. Since the components of the development process were themselves developed over time, as for example the Process Policies, each component has its own section. This is to improve readability

5.2.1 Timeboxing

Since the project was a private initiative by the product owner, Per-Arne Forsberg, the developer worked from his home. This was a problem since it impeded the collaboration between the two when they didn't get to see each other regularly face to face. To combat this timeboxing was used. Timeboxing is a time managing technique often used within agile development that means having fixed time periods in a project where each period has its own budget, deadline and deliverables [2]. Sprints in Scrum is for example a timeboxing technique, see section 3.1.2 Events.

Kanban does not talk about timeboxing. The reason is partly because Kanban is very lightweight and partly when using Kanban to its full extent, timeboxing is not necessary. Complexity and scope variables, together with prioritization, for each work item in Kanban should be enough to estimate how many items that will be completed within a certain timespan. In this project however, the developer was unexperienced working on web based projects, he worked alone and the time period for the project was only a few months. Therefor it made sense to use some sort of timeboxing technique instead of guessing complexity variables that would likely not be good estimates until the end of the project.

The timeboxing used within the project was simple. Every tuesday at 10:00 the developer and product owner met for two to three hours and the primary items, or cards, to be completed until next tuesday were selected. In other words, a fixed time period of one week was chosen for the timeboxing. Secondary items were also selected in case development went smoother than expected. If the goal wasn't reached for a week, this was discussed and information was added to the cards in question. Sometimes this led to cards being split into several independent smaller items and sometimes the cards were simply moved to the next weeks deadline together with added information. Deadlines not reached were never seen as a failure, instead they were viewed as a learning experience to be able to better estimate the workload in the future. This correlates well to the Kanban practice seen in section 3.4.5 Improve Collaboratively.

Besides prioritizing and estimating a weeks worth of work, the meetings were used to iteratively update the cards, especially the use-case cards, on the Kanban board which is brought up in section 5.2.3 Kanban Cards. During the course of the project extra meeting times were periodically added, especially at the start of the project to get the ball rolling.

5.2.2 Process Policies

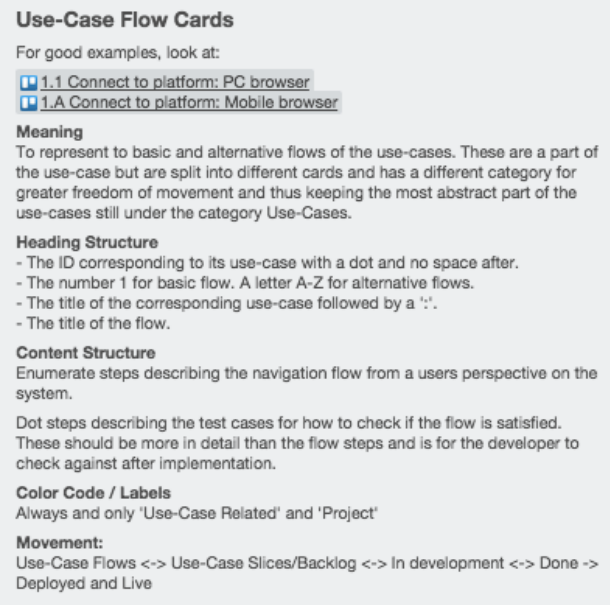


Figure 5.1 – A sample of the projects Process Policies. This sample shows information regarding the type of cards called Use-Case Flow Cards. For a full list of the Process Policies, see Appendix A Project Process Policies.

One of the first things that was looked into, after deciding on using Kanban as the base methodology, was the structure of the Kanban cards, in other words the Process Polices. Typically in Kanban, these cards are either simply tasks and/or user stories. Since Kanban itself does not explicitly state the structure or layout of the cards, the strategies described in Use-Case 2.0 were selected to form the basic structure [18]. This will be described further in section 5.2.2 Use-Case 2.0.

Each card type got it own set of rules to follow, see Figure 5.1. Whenever a card was made it was checked against these rules at the next meeting, in order to make sure it followed the defined structure. Moreover, the Kanban board had several rules and procedures connected to it such as that every item in the Done column, see Figure 5.3, waits for inspection at the next meeting. All of these rules and procedures were written in a card called *Rules / Process Policies* that can be seen in most upper left card in Figure 5.3. This ensured that they were always easily accessed, modified and clearly visible when working with the Kanban board.

The projects Process Policies are written in Appendix A Project Process Policies, which is a mix of Process Policies connected to Use-Case 2.0, Kanban and more. The Process Policies were not all written at one point in time. As the project grew more policies were added. As soon as new procedures and rules were developed or discovered they were written down. However a few were not written in text form for various reasons such as being to obvious or excessive for a project consisting of two people. For example the timeboxed meetings every tuesday.

Use-Case 2.0

User stories is a common approach in agile development to represent project requirements and specifications in an easy to understand, abstract and layered structure. A use-case is a similar alternative approach. The difference is in the scope and its focus. While user stories tend to be more abstract and use everyday language to keep it open for interpretation, use-cases instead are more user-to-system goal focused and tend to use a more business oriented language. Use-cases also tend to have larger scope that includes components such as Unified Modeling Language (UML). It's common that project requirements are built using both user stories and use-cases techniques in some sort of combination. The definition for both of these techniques are not that strict in order to have an easier time adapting them to any project type and size.

Use-Case 2.0 is a popular technique meant to be lightweight, scalable, versatile and easy to use [18]. If it delivers what it promises it sure fits the description of what to look for in a small-scale project with only a few members. Also the technique is meant to be used with methodologies such as Kanban and Scrum which was suitable. Without going into too much detail the technique, when used with Kanban, can be summarized as following:

1. Agree on the goals and scope of the system and identify all actors and ways of using the system. Repeat this regularly as things can change as the project develops.

2. Each use-case represents a meaningful logical interaction between some actor and the system.

3. Create a narrative for each use-case that explains the actors goal and interaction on an abstract level and in everyday language, similar to how user stories are written.

4. Create basic and alternative flows for each use-case that in a step-by-step manor goes through a users perspective in navigating the system to achieve the use-case goal.

5. Create basic test cases for how to check if the use-case and use-case flows are satisfied.

6. When development on a use-case grows "near", as in actual implementation is soon to come, create slices using the use-case flows. As the name suggests, try to create each slice so they correspond to a suitable and somewhat equally sized work. The slices are created by either splitting a use-case flow or combining multiple flows. The slice can also go deeper in detail if wanted or needed, both in terms of the use-case flow steps and test cases.

7. Work on a certain number of slices at a time (depending on the size of the project team) and when implemented, check them against the test cases.

8. Keep updating the use-cases with information as slices are completed.

A more abstract view of the steps are: Find the project goals, create use-cases describing the interaction between user and the system, prioritize the use-cases and the closer development comes to a use-case the more information it should contain. Also regularly keep finding and updating the goals. In the project, a part of the timeboxed meetings was allocated to indentfy and update the use-cases.

Note that these steps are a simplified summarization of Use-Case 2.0 that is influenced by both Kanban and the project. Read more about how to technique works in in the e-book *Use-Case 2.0, The Guide to Succeeding with Use Cases* by Ivar Jacobson, Ian Spence and Kurt Bittner [18].

5.2.3 Kanban Board

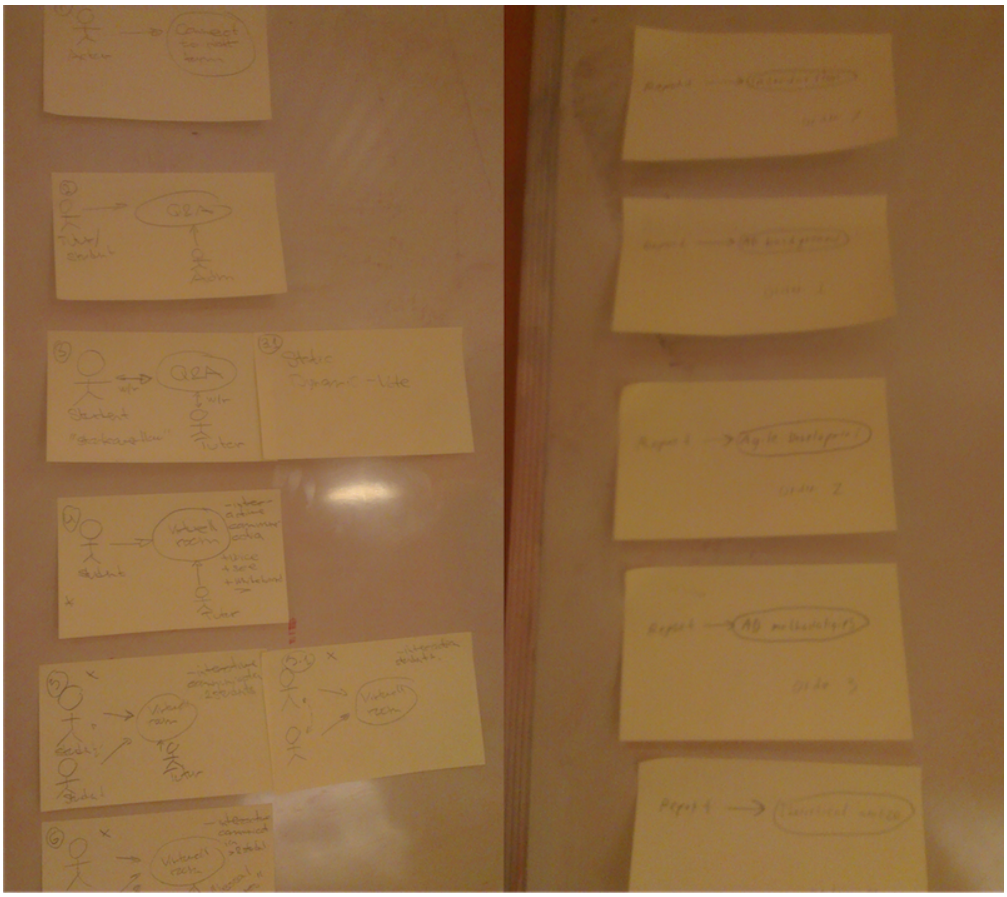


Figure 5.2 – Two combined pictures of the earliest stage of the Kanban board and use-cases. The left image shows the use-cases of the project where the most important components are already visible. The right side shows the initial use-cases for the report. The author sends his apologies for the poor image quality.

After deciding to use Kanban together with Use-Case 2.0, about five longer meetings corresponding to almost three full work days were spent on brainstorming the initial use-cases for the project. Each use-case was represented by a card and each card was a post-it note on a physical whiteboard. These brainstorming sessions were kept highly abstract with focus on detecting the most important components of the project as well as all necessary components to make the website prototype a reality. This early stage of the project can be seen in Figure 5.2.

The physical whiteboard became highly impractical as progress was made. The cards had to be taken up and down at every meeting since meetings took place at different public locations. Moreover, the overview became blurry with the use of only colored pencils and manual card linking. The most devastating issue was that modifying the content became a hassle and being able to easily change and modify the requirements is one of the most important aspects of agile development. So it was decided to move on to a virtual Kanban board. Many online options exists such as Kanban Tool, Kanbanize, Trello, Kanbanchi, SmartQ and many more. The selected online tool became Trello with the main reasons being that it's well established, free, lightweight and very easy to use. The advanced tools offered by the other options were not needed for a project of only two people.

The Kanban board within Trello quickly became the center of the project and the center of attention at every meeting. All problems with the physical Kanban board were gone. Figure 5.3 shows how the board looked after about three months into the project. As explained in the picture, only a few of the standard columns in Kanban were used. The reason being to limit overhead when only one developer was working on the board. For example the backlog and todo columns, which are a part of the standard Kanban board visible at Figure 3.2, are combined. Testing is also assumed to be completed before moving to column Done and once again tested at a meeting before moving to Deployed and Live, instead of having an explicit testing step/column.

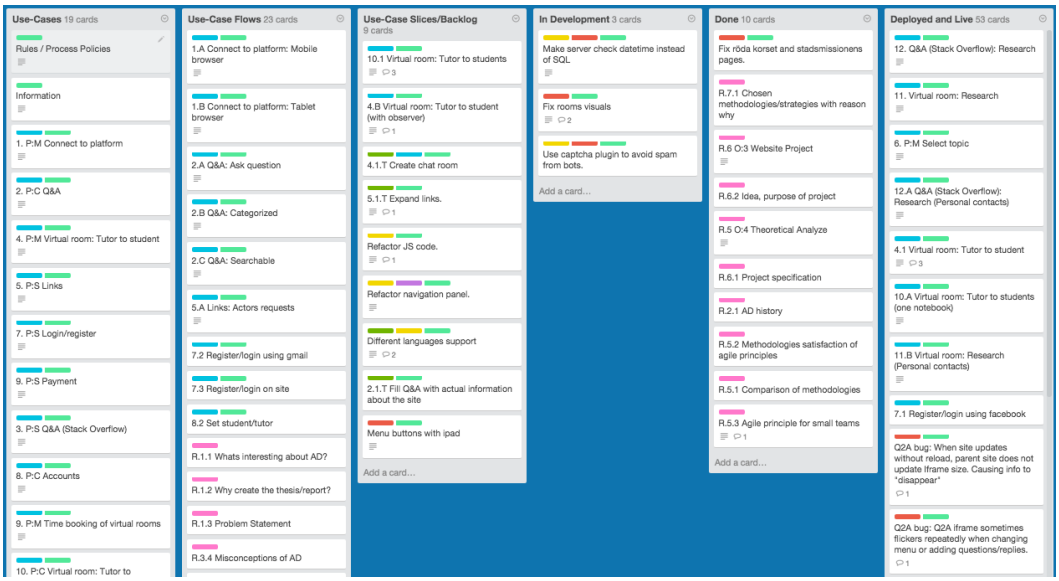


Figure 5.3 – A part of the Kanban board used with the online software tool Trello: https://trello.com. The three most left columns are part of the Use-Case 2.0 strategies. The four right columns are basic steps used within Kanban that somewhat corresponds to the columns seen in Figure 3.2. The third column is both a part of the Use-Case 2.0 strategies and a standard Kanban step.

Kanban Cards

The card types used were Use-Case Cards, Use-Case Flow Cards, Use-Case Slices Cards, Task Cards and Report Cards. To understand how these exactly work it's best to look at Appendix A Project Process Policies. An example of the structure policies, or rules, for the cards can be seen in Figure 5.1.

Close to everything about the project was written inside the cards. Every task or idea had a card connected to it. The cards basically has three movement types after creation: It can progress to the right on the board until it ends up in column Deployed and Live. It can at some point in some category be removed, or archived as it's called in Trello, if it's found unwanted. These cards are not permanently deleted and can always be found again after removal. Finally a card can find itself to be frozen in some category, except in In Development and in Done. The cards in Deployed and Live are for instance normally frozen in place, unless some event leads to reopening of the card. Other exam-

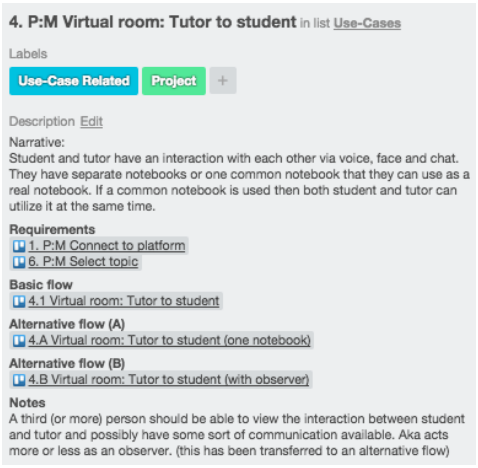


Figure 5.4 – Example of a Use-Case Flow Card.

ples are low priority cards in the Backlog, or earlier steps, that have a small value but high complexity leading to postponing.

At each meeting, new cards could be created or existing cards could be modified. It was during the meetings that it was decided which cards should be kept, removed or worked on for the coming week. Also the cards in the Done column were tested and discussed together with the cards in the In Development column. Another important part of the meetings were organizing the Backlog column so that the cards were kept arranged in priority order from top to bottom.

The Report cards are special. They were added early since the author of report and the developer are the same person. Therefor it made sense including these cards since sometimes a significant amount of time had to be spend on the report or investigating agile development. This gave the product owner a better overview and comprehension of what was going on at any point in time. After all, one of the practices in Kanban is about having transparency in a project, see section 3.4.1 Visualize.

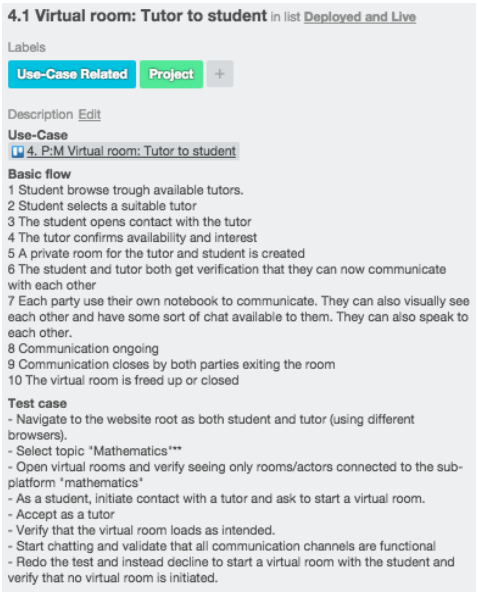


Figure 5.5 – Example of a Use-Case Flow Card.

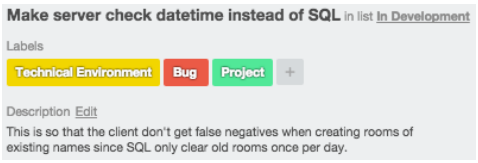


Figure 5.6 – Example of a Task Card.

5.3 Project Results

Drawing any conclusions regarding how much Kanban assisted the project in terms of quality and outcome is complicated, which is discussed in chapter 6 Discussion. However a seconday goal was to was to develop a working prototype of the interactive website, as seen in section 1.1.1 Goals, and this chapter shows where the project landed in that regard.

The goals for the prototype were initially set after the projects use-cases were identified, as described in section 5.2.3 Kanban Board. However since the use-cases contents changed, as the project developed, the definition of the goals also changed. As The Agile Manifesto states, *Responding to Change over Following a Plan*. Excluding the most basic features, such as having a functional navigation, the final required functionality for the website to be counted as a working prototype became these three use-cases:

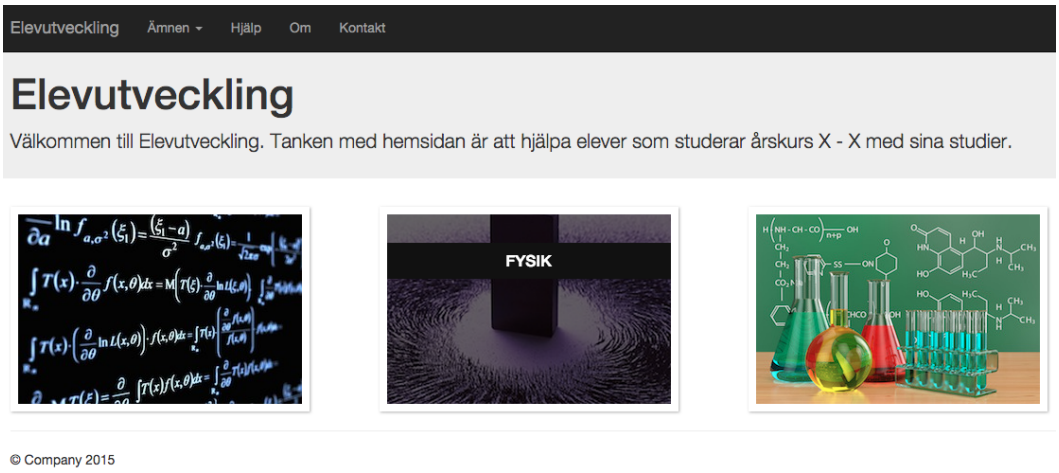


Figure 5.7 – The home page of the website prototype showing three selectable images for school subjects. Mathematics, physics and chemistry. The mouse is hovering over physics (FYSIK). At the time of this reports writing the website can be found at: http://elevutveckling.com

1. **Virtual room: Tutor to student**
   The most important use-case. For each subject, a student and a teacher must be able to enter a virtual room together containing a virtual whiteboard. The whiteboard should function as a regular whiteboard where the student and teacher can draw equations and do calculations. All three communication channels, video, voice and chat, must be available.

2. **Q&A (Stack Overflow)**
   Each subject must have a questions and answers section, similar in functionality as the well known website called Stack Overflow. The students must be able to ask questions, find questions and answer other students questions related to the subject they are currently navigating in.

3. **Links**
   The least complicated of the three required use-cases. Each subject must have a range of interesting external links connected to the studying of that subject.

Note that these descriptions are the most basic requirements for the prototype and the actual use-cases contains more information as seen in Figure 5.4 and 5.5.

The programming language used was JavaScript on the client side together with the common Document Object Model (DOM)-manipulating library jQuery. On the server side PHP: Hypertext Preprocessor (PHP) was used together with MySQL. The human-readable text format JavaScript Object Notation (JSON) was used extensively both for client/server communication and for the textual representation on the website, as well as for some other content such as image paths and external links.

This was done partly because the website was developed in Swedish. Having all representable text structured in JSON files means having an easier time changing language in the future. Another reason was to make the changing of content easy for anyone, such as the product owner, by only looking at and modifying the JSON files, thus avoiding having to understand the actual code and the database.

More technical details are available at Appendix B Project Technical Details. That appendix is also meant as the documentation for the product owner in order to transfer technical knowledge regarding the project.

5.3.1 Project Components & Tools

This section will show the more abstract components and tools used in the project to create the website as for example what frameworks were used.

Git

Of course the some sort of code management and version handling system were to be used in the project, as every project should have. Git was chosen as it's a popular and well established code management system that the developer had prior experience working with. The online Git framework Github was used to offer a better visual experience. `https://github.com`

Misshosting

The web hotel used for the website. Used simply for having the servers in Sweden together with the fact that it uses cPanel which is a popular user friendly graphical web hosting control panel, which makes the website easier to administrate for people inexperienced in web development. MySQL was used for the database. `https://misshosting.se`

Bootstrap

Bootstrap is a well established comprehensive design framework using Cascading Style Sheets (CSS) and JavaScript. It was used to let the developer, who has little experience in design, focus more on functionality. Besides giving a sleek design it's responsive in such a way that the website looks and functions well on different resolutions such as those found in phones. Because of its popularity it also has many useful plugins such as the form validation plugin called Bootstrap Validator that was used in the project. Thus utilizing Bootstrap, or similar frameworks, means getting assistance in creating a dynamic website for multiple platforms and avoiding reinventing the wheel. `http://getbootstrap.com`

Groupworld

One of the earliest use-cases processed in the project was about conducting a research in order to find suitable frameworks, or platforms, to satisfy the virtual room use-case, as described in section 5.3 Project Results. Besides the developer doing a research, a third party company was used to help in the

investigation which led to the finding of Groupworld, see Appendix C Virtual Room Research Results.

Groupworld is a highly customizable cross-platform web conference platform for online meetings using video, voice and chat. The most important feature of the platform is that it integrates smoothly to existing websites. The result of the integration can be seen in Figure 5.8. http://www.groupworld.net

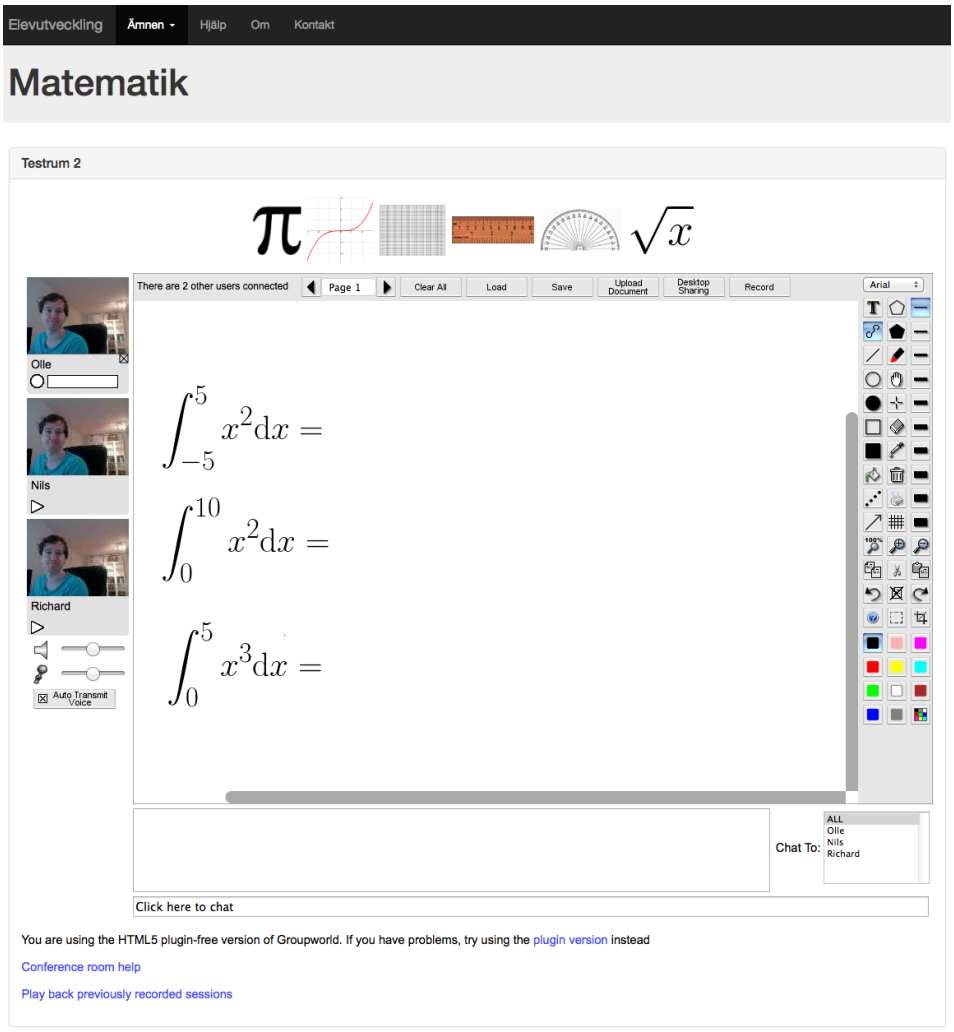


Figure 5.8 – Image showing the groupworld platform integrated with the website. The whiteboard in the middle can be used with features shown to the right, similar to Microsoft Paint. The plugins above the whiteboard offers mathematical support such as writing the sample integral equations visible in the image. The connected users can see each other, talk to each other and also chat using the chat window at the bottom. All three users in this example is the developer himself, which is the reason for the duplicate video channels to the left.

Question2Answer

A framework used to satisfy the second most important use-case requirement for the website prototype, explained briefly at section 5.3 Project Results. A initial research was conducted to find suitable frameworks, like with the virtual rooms, but without using a third party company. Question2Answer was the picked solution. Besides satisfying the basic requirements, it also offers support for voting, structuring, ranking system and more. It's popular and open source which both were favorable qualities for the project. Since the framework is meant to function as a stand alone website it was stripped down to the most essential components and integrated with both Bootstrap and the project website. The result of the integration can be seen in Figure 5.9. http://www.question2answer.org

iHover

A image framework used for styling and effects. Used for the websites navigation images, seen in Figure 5.7. Integrates well with Bootstrap. http://gudh.github.io/ihover/dist

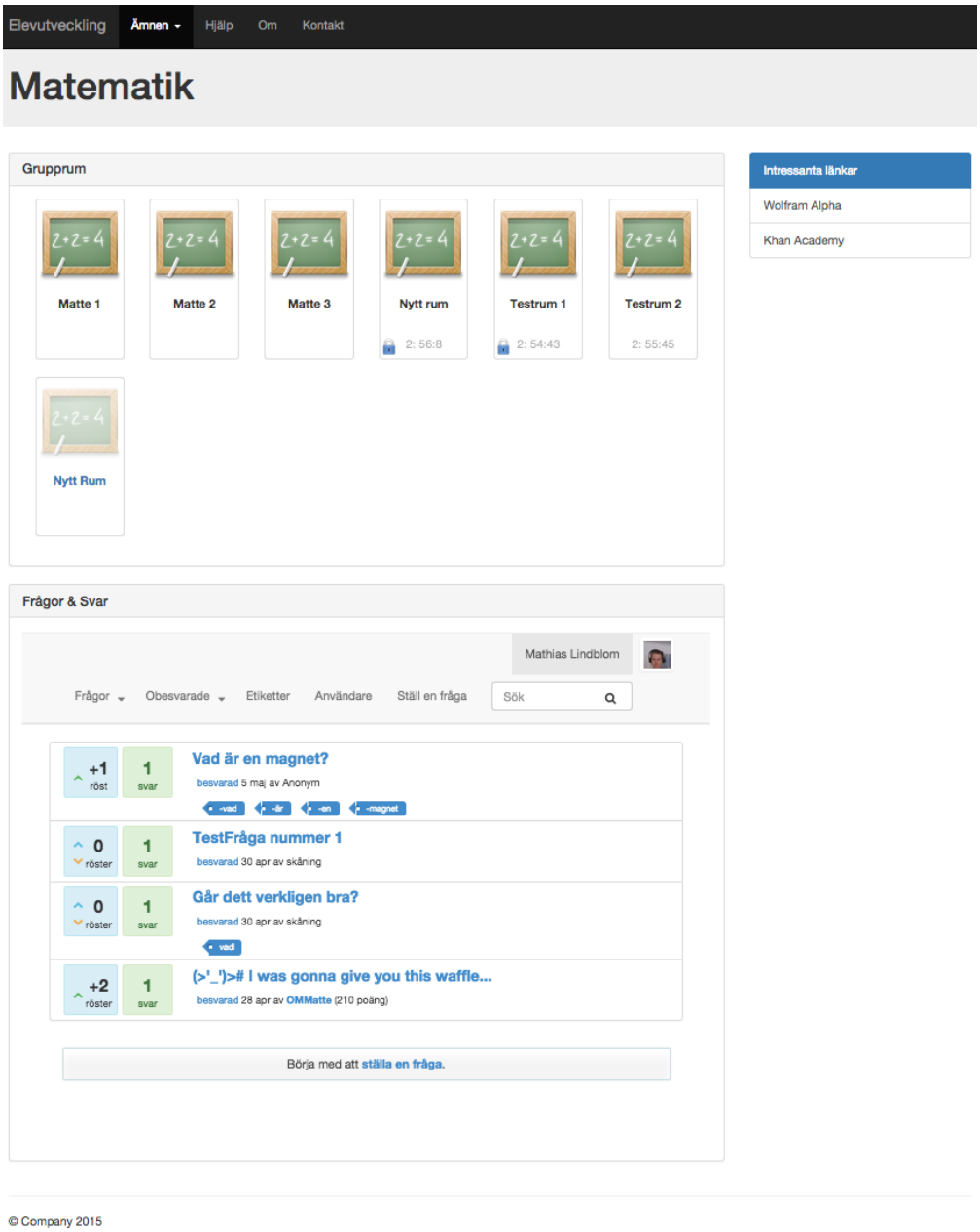


Figure 5.9 – Image showing the page of a subject, mathematics to be precise, in the website prototype. Below the page title the virtual group rooms can be seen. Some are password protected while others are publicly open. Some also have a timer which shows how long the rooms stay up before they disappear. Anyone can create a new room using the "Nytt Rum" (English: New Room) icon. At the bottom the Question2Answer framework can be seen integrated with the Bootstrap layout. To the right are links to external helpful sites regarding studying mathematics. In other words, the image shows all three required components described in section 5.3 Project Results.

5.4 Product Owner Experience

At the end of the project a short structured Interview was held with the product owner, Per-Arne Forsberg. As briefly described in section 5.1 Project Description, Per-Arne has years of experience working with different agile methodologies and had a prominent role in the project. Therefor his input regarding the outcome and experience of using the modified Kanban methodology for this project was of great value. The interview was held in Swedish. Both the questions and answers were translated into English with the interviewee present. The following are the questions and answers:

What worked well using Kanban?
Getting a solid structure. I could clearly follow how the project developed over time. It was easy keeping track of the status of the development process.

What did not work well using Kanban?
I thought it worked really well actually, can not think of anything directly bad or not working. Maybe that without the timeboxed meetings, that were not a part of Kanban, Kanban might have felt a bit thin.

Did timeboxing bring any value by using it together with Kanban?
Yes, it's a good way to follow up on the progress made and naturally leads to reprioritizing and restructuring the work. It also helps the developer(s) motivation and to keep their focus with new short term goals. Very good that you get a regular connection between the developer(s) and product owner, or some customer etctera. Timeboxing, in a distributed project, when people are sitting in different location is a good way to keep the communication up and going. It also helps keep everyone in the project focused on the prioritized project parts and goals.

Did Use-Case 2.0 bring any value by using it together with Kanban?
Yes, it worked well to use it together with Kanban, especially since it was hard for us to know how long time the different components would take to develop. Use-Case 2.0 made it easy to break down the components to more manageable pieces. After the break down it was easier to analyze and adapt the components to the Kanban board.

Do you think Kanban was advantageous to the project?
Yes. I don't know any other agile methodology that could replace Kanban for this kind of work. Not any other type of methodology either for that matter. Nothing I know of would offer the same kind of transparency to a project, all the way from the early stages to delivery. Moreover, the methodology offered great flexibility which was important in this project.

What are your thoughts on the collaboration between you and the developer in regards to the methodology used? Both positive and negative thoughts.
As I said, the transparency and structure was great and I could easily follow the work flow. We were open and clear to each other and the board helped in that regard. I could easily follow the developers progress. This made it easy to reprioritize and restructure regularly. Even the individual components progress were easy to track where they were in their development phase. Don't really have any negative thoughts.

What would you like to have done differently, given the opportunity to redo the project?
Nothing, or wait. Remove all report writing. Every sentence in the report was one functionality lost in the project.

What is your final verdict to using Kanban in the way it was used in the project?
It worked very well. As stated in the other questions, it gives transparency, structure and flexibility.

Would you recommend this project's methodology approach for similar projects?
I recommend to use this methodology when designing new projects with a small group of participating people. Especially when the initial requirements leaves many unknown parameters.

Any final thoughts?
No, just that I think the project went very well overall.

Chapter 6

Discussion

The first thing to mention is that it was initially planned to be three developers on the project. However before work even got started it was clear that it was only going be one developer, the author of this report. Three developers, or at least two, would most likely have made the report much more interesting. More agile practices and strategies, such as Pair Programming or standup meetings, could have been tried out since a lot of them are about communication and collaboration between the developers. The author of this report could have had a stronger focus on agile development and leave most of the coding to the other developers. More developers would mean more input and it would have been interesting to hear the other developers thoughts and feelings about the project. For example interviews could have been held with each of the other developers. A interview with the developer could have still been held in this case, however since the developer is the report writer it would be biased and a bit excessive. Instead, this chapter indirectly includes the developers thoughts on the project.

The reader might have noticed that several of the Kanban practices were not strictly enforced in the project, which is discussed more in detail in the following section, section 6.1 Final Methodology Used in the Project. This is not simply the result of laziness. If the project members see no reason to include something for their project or even sees it as disadvantageous, it should not be included. It would actually go against the reasoning of agile development and Kanban to include something simply because it's a part of the methodology used. This might feel a bit counterintuitive: Follow the methodology, but only take the bits you like. However the very core of agile development is to be able to adapt and that includes the methodology used. Of course adapting and modifying the methodology extensively could lead to finding yourself not developing according to the agile principles any longer. Weather or not this is advantageous is hard to say and depends highly on the project and its members. As stated in the report agile development can be seen as a ideology, so as food for thought compare it to democracy. The core of democracy is letting everyone vote to change the society. Take this to its extreme and you can vote to longer have a democracy.

6.1 Final Methodology Used in the Project

This section is a summary of techniques used and highlights and briefly discusses the most apparent changes to Kanban, while having the practices in focus. It's important to understand that when using some agile methodology such as Kanban, there will always be some level of modifications to suit a project.

A WIP limit, according to the Kanban practice section 3.4.2 Limit Work-in-Progress was never strictly set because it felt excessive when using only one developer. It was simply assumed that the developer's current WIP always were the cards within In Development step/column and the developer alone chose when to pull cards to that column.

One of the more notable changes was that the complexity parameters and more thorough analyzing of the cards were skipped in favor of timeboxing, as described in section 5.2.1 Timeboxing. The practice section 3.4.3 Manage Flow talks about using the more advanced parameters of the cards and analyze the flow of the cards. This practice was almost completely ignored. The reason is partly since the usage of the more advanced card parameters were skipped for described reasons. Another reason is that measuring flow with only one developer, that simultaneously writes a report, would give quite a notable error margin for obvious reasons. The progress was instead discussed and estimated vocally at the weekly meetings.

The practice described in section 3.4.4 Make Process Policies Explicit was clearly utilized. Most Process Policies were written down and structured, as seen in Figure 5.1, while others were verbally agreed upon such as timeboxing or seen as to obvious to be needed to exist in written form. All though it's questionable if verbally agreed upon process policies can be called explicit.

Both section 5.2.3 Kanban Board and section 5.2.3 Kanban Cards should be enough to deem the practice Visualize to be satisfied to a high degree. As Kanban advocates the Kanban board was always in the heart of the development process and the content was regularly updated and discussed at every meeting.

It's hard to come to a conclusion regarding to what degree the practice Improve Collaboratively was satisfied. However it was certainly not ignored. At each weekly meeting for example, it was discussed what was working and what was not, both in terms of the methodology and the project. This is brought up in section 5.2.1 Timeboxing. Estimates regarding what would be done for the following week became better as the project took form and the developer gained experience.

The usage of Use-Case 2.0 was not necessarily a modification to Kanban. It was more of a complementation. The Kanban cards needed some basic structure and layout. Use-Case 2.0 had a solid and easy to understand yet somewhat advanced configuration. To get the most advanced user story or use-case system was not important, however it was convenient using a system that was meant to be used with Kanban and took the card structuring a long way. Most importantly was that it was well documented which was not the case for many of the user story and use-case systems available.

To summarize the Kanban practices are considered satisfied except Limit WIP that was found excessive and Manage Flow that was considered to give little value and cause overhead for the small-scale project. Timeboxing was added to achieve an iteration based workflow similar to Sprints in Scrum. Use-Case 2.0 was used to structure and complement the Kanban cards on the board.

6.2 Report Critique

There are several reasons to be critical for the type of research conducted in this report. Some have already been mentioned briefly throughout the report. The first thing to note is that agile development by itself is quite vague in its definition and the work regarding agile development and its methodologies are mostly based on expert experience. It's the result of industry experts that have seen problems, come up with ideas to fix them, had companies or themselves try them out, and finally evaluated the outcome. The problem is that there are so many parameters and influences when evaluating software development methodologies that it's really hard to make sense of the result. Too be clear, the data is very noisy. For instance, you can not make some software using one methodology and then have the same development team create the same software again using another approach as an evaluation method for methodologies. The results of the new approach will most certainly be influenced by their first attempt. What is really needed is several development teams consisting of developers with somewhat equal experience and background. However, depending on parameters such as if the software type is a website, game or security system the results might still vary a lot. Thus most work regarding agile development and its methodologies, including this report, falls into the category of social science or soft science which is important to have in mind.

Each agile development methodology has vast amount of information, interpretations and alternative versions going for them. To make a full theoretical analysis and comparison is out of scope for this report. The first and easy limitation was to not include every possible agile methodology in order to have room for more depth into the chosen methodologies as well as to keep the complexity down. The reasoning behind the four chosen methodologies can be read at section 3.5 Other Mentionable Methodologies. However, even with this limitation it would not be viable going through every aspect of each methodology since each methodology could fill a report by itself. So instead the practices of each methodology was chosen as a solid and common ground to be analyzed. What this means is that some elements that defines the methodologies have potentially been "cut off" and therefore given an unfair assessment to some degree.

6.3 Conclusions

The consensus of the developer and product owner, who has years of experience working with agile methodologies, was that it was a success using Kanban the way it was used, as described in section 6.1 Final Methodology Used in the Project. For the small-scale project developed in this report, using Kanban this way was certainly a suitable agile methodology. Given the practical results and experience of the participants it's reasonably safe to say that using the same approach for other similar project configurations would lead to the same positive conclusion.

As shown in Figure 5.9 and talked about in section 5.3 Project Results, the project itself was also a success. Most importantly, the owner of the project sees the project as successful, see section 5.4 Product Owner Experience. It's impossible to say where the project would have ended without the use of Kanban but the methodology helped making sure that the project goals were kept up-to-date and always in focus for the developer.

6.4 Future Work

The actual project will continue to be developed and it's up to the project owner to decide where he wants to take it.

Regarding agile development, in future research this report can be used as an expert review when investigating agile development methodologies and strategies for small-scale projects. This report not only talks about how agile methodologies can be applicable to small-scale projects but also offers a well documented example of how to practically do it using Kanban. This is something rarely seen in literature and research papers about agile development. Since Kanban is a lightweight methodology, that have several alternative expanded versions, this report might also offer some assistance in further developing alternative Kanban methodologies. Moreover, it could be used in order to further develop a standard for small-scale projects using Kanban.

Glossary

Early Victory A strategy commonly referenced in agile development that refers to the sociological benefits of showing early and frequent results, such as increasing morale and trust. 27

Executive Sponsor Also referred to as the project sponsor. The executive sponsor is a role in project management that is responsible for the success of a project, within a business/company. 29

Git Git is a popular distributed revision control system mainly used for software development. 23, 31

JIT Just-in-time is a production strategy that seeks to improve the ROI for a business by reducing in-process inventory and associated carrying costs. In simpler terms, reducing the stock up of inventory items in a process chain. 32

Osmosis The movement of a liquid, sometimes gas, through a plasma membrane (which is a sort of filter) which over time results in a balancing from high concentration areas to low for the molecules that are able to pass through the plasma membrane. 28

ROI Return on investment means the benefits, or return, for the investor resulting from an investment of some resource. 18

Runaway Process Inflation Runaway inflation is a very rapid inflation (decrease of money value). Runaway process inflation is a term sometimes used within agile development to describe rapidly growing development processes. 5

Stakeholder Has several meanings but this report refers to corporate stakeholders. A corporate stakeholder is a broad term meaning a person or party who has an interest in a certain business or project for a company. It can for example be investors, customers and suppliers. 19, 20, 33

Technical Dept Technical dept (also called design dept or code dept), is a metaphor primarily used in software development referring to the consequences of poor

system design. The dept builds up as you for example introduce bugs and so called spaghetti code that has to be fixed in the future, as in repaying your dept. 5

User Story Within software development and product management, user stories are short descriptions, or stories, that tries to envision the users interaction for some requirement. The layout of user stories varies for different projects but It should capture the "who", "what", and "why" in a short but descriptive way. Usually the stories are highly abstract but before implementation on a story begins they are broken down into more fine graned sub-stories. The term is commonly used within agile development and originates from the methodology XP. 22, 24, 25, 45, 46, 60

Acronyms

CSS Cascading Style Sheets. 52

DOM Document Object Model. 51

GUI Graphical User Interface. 25

JIT Just-In-Time. 32, *Glossary:* JIT

JSON JavaScript Object Notation. 51, 52

LPD Lean Product Development. 12

LSD Lean Software Development. 12, 13

PHP PHP: Hypertext Preprocessor. 51

ROI Return-On-Investment. 18, 63, *Glossary:* ROI

TTD Test-Driven Development. 21–23, 39

UML Unified Modeling Language. 46

WIP Work-In-Progress. 32, 33, 60, 61

XP Extreme Programming. 21, 23–27, 35, 39–41, 64

Bibliography

[1] Martin, C.R. and Martin, M. 2006. Agile principles, patterns, and practices in C#. Prentice Hall, Massachusetts. 768 pp.

[2] Agile Alliance. 2015. The alliance. [http://www.agilealliance.org/the-alliance/]. Accessed Mars 5, 2015.

[3] Crockford, C. 2007. Quality. [https://www.youtube.com/watch?v=t9YLtDJZtPY&t=175]. Accessed Mars 12, 2015.

[4] Knuth, D.E. 1984. Literate Programming. Comput J. 27: 97-111.

[5] Cockburn, A. 2004. Crystal clear: a human-powered methodology for small teams. Addison-Wesley Professional, New Jersey. 312 pp.

[6] Sinek, S. 2014. Why good leaders make you feel safe. [http://www.ted.com/talks/simon_sinek_why_good_leaders_make_you_feel_safe]. Accessed Mars 13, 2015.

[7] Keil, M. and Carmel, E. 1995. Customer developer links in software development. Commun ACM. 38: 33-44.

[8] Williams, L., Kessler, R.R., Cunningham, W. and Jeffries R. 2000. Strengthening the case for pair programming. IEEE Softw. 17: 19-25.

[9] Beck, K. and Andres, C. 2004. Extreme programming explained: embrace change, 2nd edition. Addison-Wesley Professional, New Jersey. 224 pp.

[10] Scrum Guides. 2013. The scrum guide. [http://www.scrumguides.org/scrum-guide.html]. Accessed Mars 23, 2015.

[11] Anderson, D.J. 2010. Kanban: successful evolutionary change for technology organizations. Blue Hole Press. 278 pp.

[12] Klipp, P. 2014. Getting started with kanban. Kanbanery. 37 pp.

[13] Everyday Kanban. 2015. What is kanban. [http://www.everydaykanban.com/what-is-kanban/]. Accessed Mars 25, 2015.

[14] Schwaber, K. 2004. Agile project management with scrum. Microsoft Press. 192 pp.

[15] Womack, J.P., Jones, D.T. and Roos, D. 2007. The machine that changed the world: the story of lean production. Free Press, New York. 352 pp.

[16] Poppendieck, M. and Poppendieck, T. 2003. Lean software development: an agile toolkit. Addison-Wesley Professional, New Jersey. 240 pp.

[17] Fowler, M. 2008. Agile versus lean. [http://martinfowler.com/bliki/AgileVersusLean.html]. Accessed April 1, 2015.

[18] Jacobson, I., Spence, I. and Bittner K. 2011. Use-case 2.0. Ivar Jacobson International. 55 pp.

Appendix A

Project Process Policies

The Process Policies for the project are written in the online organization tool called Trello. For the time being they can be accessed using the following link: https://trello.com/c/Xtm9s8LF/95-rules

However since the project is ongoing the information will change and the link might become invalid in the future. A sample of how to Process Policies are written in Trello can be seen in Figure 5.1. The Kanban board in Trello is visible at Figure 5.3. Everything below is an extraction of the Process Policies at the time of this reports writing, slightly modified to suit the layout of this report.

A.1 Project Process Policies

Strategies in Use-Case 2.0 are integrated with the Kanban board.

A.1.1 General Kanban Board Policies

- Normally the cards move from left to right on the board. Each card type has more information on this below.
- The categories Use-Cases and Use-Case Flows are only for the Use-Case Cards and Use-Case Flow Cards. However, new ideas should be new cards in Use-Cases and discussed in the next meeting.
- The Use-Case Slices/Backlog is the "To-Do" list that a developer can grab and move to In Development. The cards highest up are the most prioritized card.
- Everything in In Development should be something being worked on by a developer. If the developer pauses work it should move back to the Backlog and if the work is done it should move to Done before grabbing a new card.
- Before moving to Done, the developer should have run the tests described in a card. Even if no tests are written, testing should still be conducted.

- Cards in Done await inspection before being moved back to the Backlog or being decided to put the code live.
- At the next meeting. Each card in Done should be explained and tested once again before deciding if the card should be Deployed or not. The card must of course be put live by a developer before being moved to Deployed.
- Cards only move to Deployed and Live from Done. The cards must also of course be deployed and live.

A.1.2 Use-Cases Cards

For a good example, look at:
https://trello.com/c/JvD4oEyF/47-1-1-connect-to-platform-pc-browser

Meaning

To represent to most abstract part of the use-cases whose content could change greatly over time.

Heading Structure

1. Use-case ID, a unique number followed by a dot.
2. Priority following MoSCoW written "P:X" where 'X' is either M (Must), S, (Should), C (Could), W (Would).
3. Title name of the use-case.

Content Structure

Begin with the narrative and use everyday language. What does the actor want to achieve? What is the goal of the use-case? Make sure it's clear who the actor is.

If requirements exist, leave a heading and link to each Use-Case Card or any other card that is a requirement. No need to implement already implemented cards.

Leave a heading for the basic flow and all alternative flows, if they exist. Link each flow to the card corresponding to that particular flow.

Add any possible notes that are not related to the narrative in the end. Comments can also be used for notes.

Color Code / Labels

Always and only 'Use-Case Related' and 'Project'

Movement

Use-Cases <-> Done -> Deployed and Live

The card sit still in Use-Cases and only move to done when all use-case flows corresponding to the card are moved to Done or Deployed and Live. Once

there the decision is made weather or not it's truly satisfied and can either move back or to Deployed and Live. The card can also be Archived if decided to be unwanted.

A.1.3 Use-Case Flow Cards

For good examples, look at:
https://trello.com/c/JvD4oEyF/47-1-1-connect-to-platform-pc-browser
https://trello.com/c/4krVGIwA/48-1-a-connect-to-platform-mobile-browser

Meaning

To represent to basic and alternative flows of the use-cases. These are a part of the use-case but are split into different cards and has a different category for greater freedom of movement and thus keeping the most abstract part of the use-cases still under the category Use-Cases.

Heading Structure

1. The ID corresponding to its use-case with a dot and no space after.
2. The number 1 for basic flow. A letter A-Z for alternative flows.
3. The title of the corresponding use-case followed by a ':'.
4. The title of the flow.

Content Structure

Enumerate steps describing the navigation flow from a users perspective on the system.

Dot steps describing the test cases for how to check if the flow is satisfied. These should be more in detail than the flow steps and is for the developer to check against after implementation.

Color Code / Labels

Always and only 'Use-Case Related' and 'Project'

Movement

Use-Case Flows <-> Use-Case Slices/Backlog <-> In development <-> Done -> Deployed and Live

A.1.4 Use-Case Slices Cards

For a good example, look at:
https://trello.com/c/Dpd63Jy0/114-5-1-t-expand-links

Meaning

These cards are Use-Case Slices or Use-Case Tasks that represents a part of a Use-Case Flow. It can either be a few exact steps of the flow, a task needed to satisfy the flow or a small addition to a flow deserving of its own card.

Heading Structure

1. The entire Use-Case Flow ID followed by no space.
2. The letter 'T'.
3. Title of the slice, or task.

Content Structure

As you wish. The purpose of the card should be clear. However if the card is a strict part of some steps of a flow. Those steps should be written down.

Color Code / Labels

Always 'Project'. Always 'Use-Case Related' if it's a strict subset of a Use-Case Flow. Never 'Report'. The rest are optional.

Movement

Use-Case Slices/Backlog <-> In Development <-> Done <-> Deployed and Live

These cards can be directly put to 'In Development'.

A.1.5 Task Cards

A good example:
https://trello.com/c/5Mrp0crr/121-refactor-navigation-panel

Meaning

A task card does not directly correlate to a use-case and its flow(s). It can be a small feature, bug, design improvement etcetera. Basically anything that can add some form of value but does not fit logically to a use-case. If the complexity of card is large it should perhaps be converted to a Use-Case card.

Heading Structure

Just a fitting title of the task.

Content Structure

As you wish. Be as detailed as possible and needed.

Color Code / Labels

Always 'Project'. Never 'Report' or 'Use-Case Related'. The rest are optional.

Movement

Use-Case Slices/Backlog <-> In Development <-> Done <-> Deployed and Live.

These cards can be directly put to 'In Development'.

A.1.6 Report Cards

Meaning
These cards are for the report writer and only he/she knows its meaning.

Heading Structure
As the report writer pleases.

Content Structure
As the report writer pleases.

Color Code / Labels
Always and only 'Report'.

Movement
As the report writer pleases.

Appendix B

Project Technical Details

ToDo: Will contain technical information regarding the project in terms of how everything is built up and how to further develop it. It is meant primarily for the product owner and the interested reader. This documentation is at the moment still being written.

För att förtydliga så håller jag på att arbeta fram detta, ska försöka med hjälp av detta kunna överföra mitt arbeta till Per-Arne för fortsatt utveckling. Utifall examinatorn eller handledaren tyckte denna appendix såg lite tunn ut, hehe...

Appendix C

Virtual Room Research Results

The following is the results from using a third party company to conduct research regarding finding suitable virtual room frameworks, or platforms, for the project. A platform called Groupworld offered by Group Technologies Inc became the selected platform for the website prototype, see number 7 in the included pdf on the next page.

VIRTUAL INTERACTIVE CLASSROOM SOFTWARE

Summary

The customer requirement was to find software which serve as virtual classroom.

Summary of the Findings are as follows:

Section 1: Virtual Interactive Classroom Software				
No.	**Company**	**Software**	**Price (SEK)[1]**	**For Details**
1.	eLecta Live	The Ultimate Cross-Platform Online Teaching Software	Free	Please Click
2.	WizIQ	WizIQ Online Tutoring Software	30 Days Free Trial	Please Click
3.	TutorsClass	Online Tutoring Platform	Different Pricing Packages Available	Please Click
4.	BrainCert	Virtual Classroom for effective online live classes & collaboration	Different Pricing Packages Available	Please Click
5.	Tutors Box	Online Tutoring Whiteboard	Different Pricing Packages Available	Please Click
6.	ABC Draw	ABC Draw	Different Pricing Packages Available	Please Click
7.	Group Technologies Inc	Advanced Collaborative Online White Board Software	Different Pricing Packages Available	Please Click
8.	Vyew	Continuous Meeting Rooms for Real-Time & Anytime Visual Collaboration Teach & Tutor	Different Pricing Packages Available	Please Click
9.	Paradi Solutions	Virtual Classroom Platform	*N/A*	Please Click

Section 2: Virtual Interactive Classroom Software – Patents			
No.	**Patent**	**Date of Publication**	**For Details**
1.	Panoramic learning system platform based interactive teaching method between teachers and students	19 Mar 2014	Please Click
2.	Interactive Learning Network	29 Nov 2012	Please Click
3.	System and method for virtual content collaboration	13 Sep 2012	Please Click
4.	Teacher Assignment Based on Student/Teacher Ratios	1 May 2012	Please Click
5.	Educational System and Method Having Virtual Classrooms	13 Sep 2012	Please Click

[1] 1 USD = 7.55773 SEK
(http://www.xe.com/)

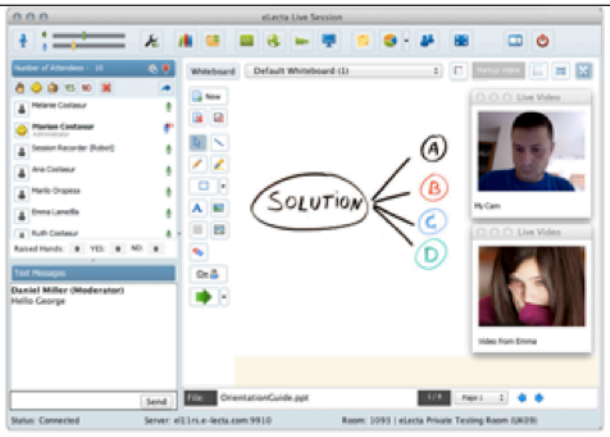
Contd. Summary

Section 3: Universities who do Virtual Classes - Universities		
1.	University of Leicaster	Please Click
2.	Victoria University	Please Click
3.	Tulane University	Please Click
4.	The University of Arizona	Please Click
5.	Federation University of Australia	Please Click
6.	University of California Riverside	Please Click
7.	The University of Queensland	Please Click

Table of Contents

Section 1: Virtual Interactive Classroom Software

Company 1. eLecta Live

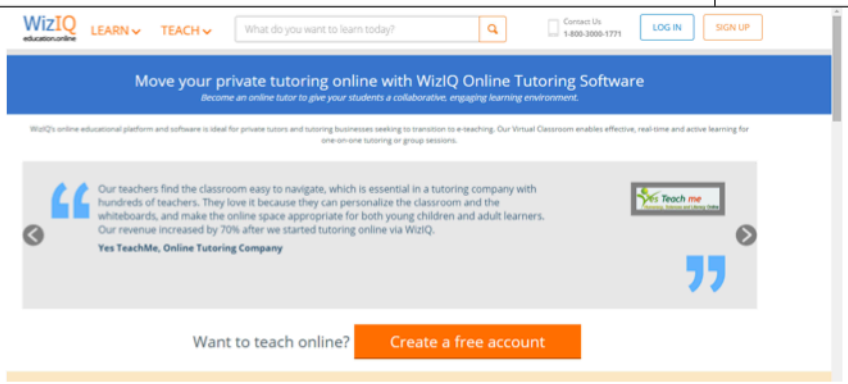
Company Details	**eLecta Live is a professional virtual classroom and a real-time online collaboration environment designed for teaching and training over the Internet. Arrange live online classes and lectures, online meetings, group sessions, individual one–on–one sessions and webinars – all taking place over the web. For your live online meetings you only need a computer with an internet connection.**	
Product 1. The Ultimate Cross-Platform Online Teaching Software		
Product Details	**eLecta Live supports live video and crystal clear audio, multiple interactive whiteboards, markup and annotation tools, file and document sharing, screen sharing and many other collaboration tools.** Teaching in the eLecta Live virtual class room is easy and provides ultimate experience for both the trainer and the trainees. In every classroom there are special built-in enhancements like audio catch-up and participant automatic content delivery feedback to ensure that all students are synchronized with the teacher and there is no one behind the others.	
Availability of Requirement	1 Tutor to 1 Student	✗
	1 Tutor to Many Students	✓
Screen Shots		

Pricing Details	**Item**	**Original price ($)**	**SEK Price**
	The Ultimate Cross-Platform Online Teaching Software	Free	Free

Contact Details		
	Address	1 Cinnamon Close, M22 4QF, GB
	Tel	(020) 8133 6276
	Fax	*N/A*
	Email	info@e-lecta.com
	URL	https://www.e-lecta.com/
Sources	https://www.e-lecta.com/Default.asp	

Contd. Section 1: Virtual Interactive Classroom Software

Company 2. WizIQ

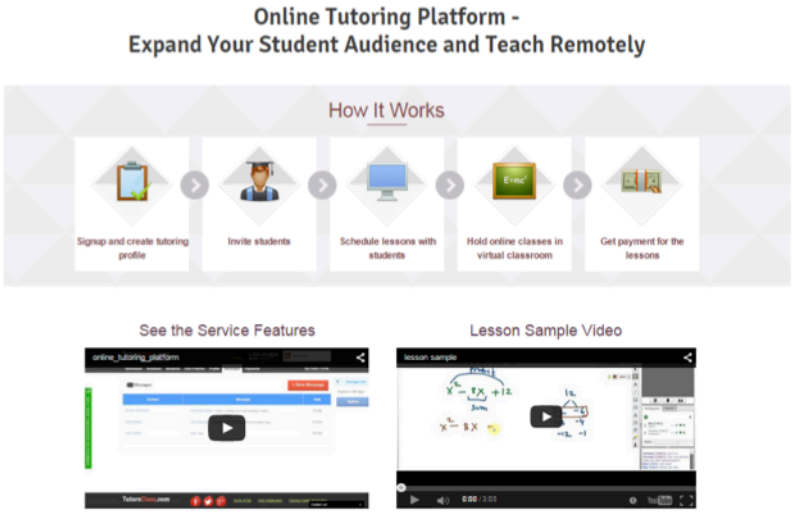
Company Details	WizIQ's online educational platform and software is ideal for private tutors and tutoring businesses seeking to transition to e-teaching.	
Product 1. WizIQ Online Tutoring Software		
Product Details	We offer 1-to-1-tutoring sessions via WizIQ for students who need individual support. The students` learning progress is guaranteed by the excellent, interactive classroom-experience and auditory-visual channels. Teaching via WizIQ and its easy-to-handle virtual whiteboard is inspiring and motivating for both teachers and students. • **Conduct one-on-one tutoring or group tutoring sessions in the Virtual Classroom** • Record and reuse your class sessions • Track student engagement and attendance • Collect student payments easily • Secure your data	
Availability of Requirement	1 Tutor to 1 Student	✓
	1 Tutor to Many Students	✓
Screen Shots		

Pricing Details	**Item**	**Original price ($)**	**SEK Price**
	WizIQ Online Tutoring Software	30 Days Free Trial https://www.wiziq.com/premium/?pageid=19	

Contact Details		
	Address	1101 Pemberton Hill Rd., Suite 101 Apex, N.C. 27502
	Tel	1-800-3000-1771
	Fax	*N/A*
	Email	support@wiziq.com
	URL	https://www.wiziq.com/
Sources	https://www.wiziq.com/online-tutoring-software/	

Contd. Section 1: Virtual Interactive Classroom Software

Company 3. TutorsClass

Company Details	An international company with almost 10-year experience in online educational business. The company has launched several projects aimed at help in education for English-speaking students from elementary school to graduate-level students.	
Product 1. Online Tutoring Platform		
Product Details	Tutorsclass offers you to get advantage of unique online tutoring product that is created exactly for online distance education. It includes: • Online whiteboard with typing and drawing tools • Live chat • File upload option • **Math formulas creation tool** • Students and lesson tracking system • Billing system • Secure payment from students directly to tutors via Paypal • Personal web-profile for each tutor Tutoring Groups and Organizations: • Tutor/student tracking, lesson scheduling • **Online classroom for one-on-one and group classes** • Payment tracking	
Availability of Requirement	1 Tutor to 1 Student	✓
	1 Tutor to Many Students	✓
Screen Shots		

Contd. Section 1: Virtual Interactive Classroom Software

Pricing Details	**Item**		**Original price ($)**	**SEK Price**
	Online Tutoring Platform		Different Pricing Packages Available	
Contact Details	**Address**	*N/A*		
	Tel	*N/A*		
	Fax	*N/A*		
	Email	support@tutorsclass.com		
	URL	http://www.tutorsclass.com/		
Sources	http://www.tutorsclass.com/howitworks/			

Contd. Section 1: Virtual Interactive Classroom Software

Company 4. BrainCert

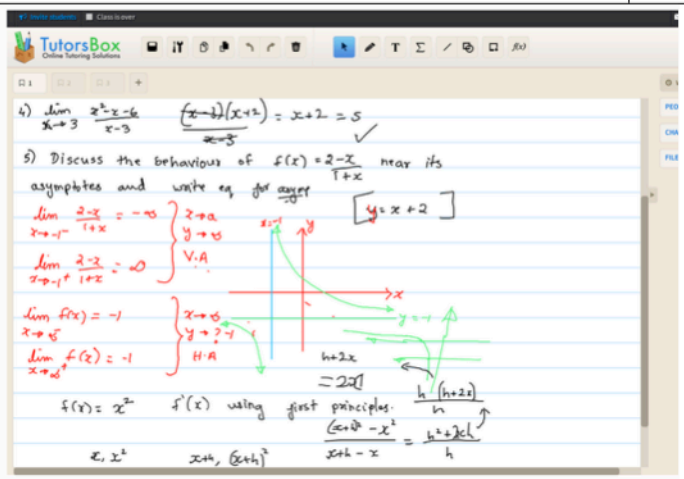
Company Details	BrainCert is a cloud-based all-in-one educational platform that comes integrated with 4 core platforms in one unified solution - courses platform, online testing platform, award-winning virtual classroom, and content management system. The result - significant cost savings, increasing productivity, and secure, seamless and enhanced user experience across all platforms	
Product 1. Virtual Classroom for effective online live classes & collaboration		
Product Details	BrainCert's "virtual classroom" functions in nearly the same way as a traditional physical classroom - providing students with a place to access live or recorded class sessions and participate face-to-face in high-quality audio and video conference along with multiple interactive whiteboards.	
Availability of Requirement	1 Tutor to 1 Student	✖
	1 Tutor to Many Students	✓
Screen Shots		

Pricing Details	**Item**	**Original price ($)**	**SEK Price**
	Virtual Classroom for effective online live classes & collaboration	Different Pricing Packages Available	

Contact Details	**Address**	1818 Library Street, Suite 500 Reston, VA 20190 USA
	Tel	*N/A*
	Fax	*N/A*
	Email	info@braincert.com
	URL	https://www.braincert.com/
Sources	https://www.braincert.com/online-virtual-classroom	

Contd. Section 1: Virtual Interactive Classroom Software

Company 5. Tutors Box

Company Details	TutorsBox is an online whiteboard designed specifically to give remote tutors the tools they need to help students succeed. It was first launched as EdoBoard in 2008.	
Product 1. Online Tutoring Whiteboard		
Product Details	• Real-time, Multi-user, Fast and easy. • **An online whiteboard application that lets you use your computer and tablet to give live tutoring classes to your students** **Features:** • Plan your classes • File sharing • **Science & Math friendly** • **Multi-boards** • Track progress • Live Audio/Video • **Multi-users**	
Availability of Requirement	1 Tutor to 1 Student	✗
	1 Tutor to Many Students	✓
Screen Shots		

Pricing Details	**Item**	**Original price ($)**	**SEK Price**
	Online Tutoring Whiteboard	Different Pricing Packages are available	

Contact Details		
	Address	*N/A*
	Tel	*N/A*
	Fax	*N/A*
	Email	contact@tutorsbox.com
	URL	https://tutorsbox.com/
Sources	https://tutorsbox.com/en/	

Contd. Section 1: Virtual Interactive Classroom Software

Company 6. ABC Draw

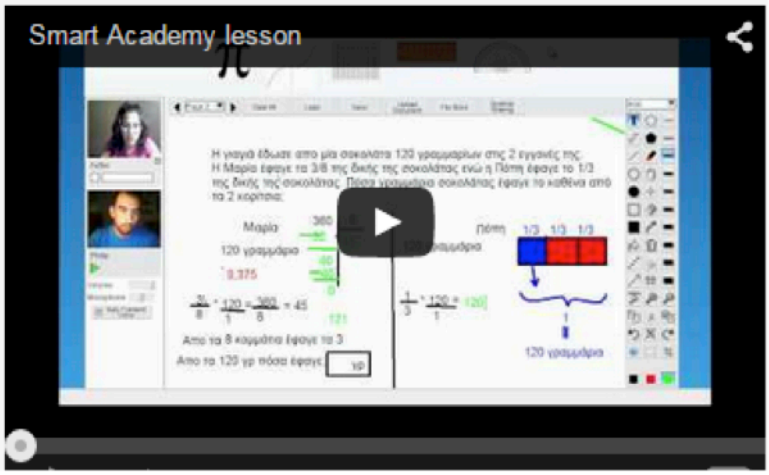
Company Details	ABCdraw transforms communication, learning, and collaboration through imagery.	
Product 1. ABC Draw		
Product Details	The ABCdraw Virtual Whiteboard connects tutors with their students to work on solving problems together. • **Teach up to 10 students at a time.** • Simultaneously write on the whiteboard. • Communicate with voice or text. • Draw lines, draw shapes, or use the text tool. • Hand-write with the ABCdraw Pad. • **Mathematical equation tools.** • Transfer files quickly. • Save images of your Virtual Whiteboard.	
Availability of Requirement	1 Tutor to 1 Student	✓
	1 Tutor to Many Students	✓
Screen Shots		

Pricing Details	**Item**	**Original price ($)**	**SEK Price**
	ABC Draw	Different Pricing Packages are available	

Contact Details		
	Address	USA
	Tel	1-800-980-9997
	Fax	*N/A*
	Email	abcinfo@abcdraw.com
	URL	http://abcdraw.com/
Sources	http://abcdraw.com/	

Contd. Section 1: Virtual Interactive Classroom Software

Company 7. Group Technologies Inc

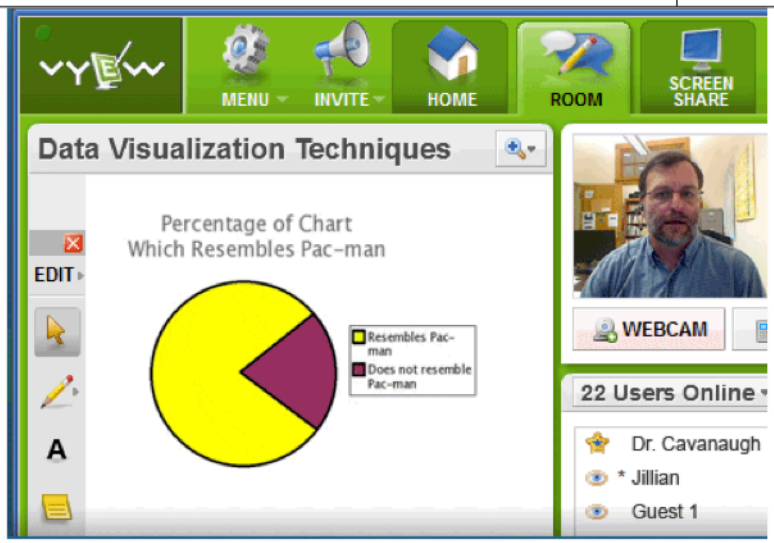
Company Details	Group Technologies Inc is a privately owned software development and consultancy company based in Ladysmith, BC, Canada. We have been developing the Groupboard range of products since 1997, and our focus is delivering high quality, reliable and stable online whiteboard and web conferencing solutions.	
Product 1. Advanced Collaborative Online White Board Software		
Product Details	Groupboard Designer is an advanced online whiteboard app which can be used from any web browser that supports HTML5 (including the iPad and iPhone) with no downloads required. **It can be operated either standalone, as a tool for marking up changes and saving them for other people to see, or it can be used in multi-user mode where muliple users can work on a design in real-time.**	
Availability of Requirement	1 Tutor to 1 Student	✓
	1 Tutor to Many Students	✓
Screen Shots		

Pricing Details	**Item**	**Original price ($)**	**SEK Price**
	Advanced Collaborative Online White Board Software	Different Pricing Packages are available	

Contact Details		
	Address	Ladysmith, BC, V9G 1A8, Canada
	Tel	(250) 667-0053
	Fax	*N/A*
	Email	info-contact@groupboard.com
	URL	http://www.groupboard.com/
Sources	http://www.groupboard.com/	

Contd. Section 1: Virtual Interactive Classroom Software

Company 8. Vyew

Company Details	California start-up. Founded by a team of UC Berkeley grads (Go Bears!), the company gained rich set of technical know-how in web applications while building product simulators for Microsoft, Nortel Networks, McAfee, and RSA Security	
Product 1. Continuous Meeting Rooms for Real-Time & Anytime Visual Collaboration Teach & Tutor		
Product Details	It's easy – no installations. It's compatible – PC, Mac, Linux, PowerPoints, documents, images, videos, mp3's, flash files. **Conferencing features – white boarding, video conferencing, screen sharing, Voice-over-IP.** Collaboration features – continuous rooms are always saved and always-on. Contextual discussion forums, voice-notes, track and log activity.	
Availability of Requirement	1 Tutor to 1 Student	✓
	1 Tutor to Many Students	✓
Screen Shots		

Pricing Details	**Item**	**Original price ($)**	**SEK Price**
	Continuous Meeting Rooms for Real-Time & Anytime Visual Collaboration Teach & Tutor	Different Pricing Packages are available	

Contact Details		
	Address	2180A Dwight Way, Berkeley, CA 94704
	Tel	(800) 594-4559
	Fax	*N/A*
	Email	Michael.North@vyew.com
	URL	https://vyew.com/
Sources	http://vyew.com/s/	

Contd. Section 1: Virtual Interactive Classroom Software

Company 9. Paradi Solutions

Company Details	Paradiso Solutions is a complete eLearning solutions company based in Silicon Valley, with a global presence not only in USA but around the world. We provide an easy to use and flexible eLearning platform as well as related services such as integrations, support and course creation.	
Product 2. Virtual Classroom Platform		
Product Details	Virtual Classroom Platform allows you to have the power of an LMS with Video Conferencing tools available for presentations, webinars, **1on1 Learning sessions, etc. Virtual Classroom Platform(VCP) provides you everything you need all in one platform.** • HD Video • Record all live events • Join and participate from your mobile device • Chat , polling and Quizzes Sharing Capabilities: • Desktop, **Whiteboard,** Documents, Applications, Remote Control Recording Add On	
Availability of Requirement	1 Tutor to 1 Student	✓
	1 Tutor to Many Students	✗
Screen Shots		

Pricing Details	**Item**	**Original price ($)**	**SEK Price**
	Virtual Classroom Platform	*N/A*	

Contact Details		
	Address	548 Market St, #13781, San Francisco, CA 94104
	Tel	1 800 513 5902
	Fax	*N/A*
	Email	sales@paradisosolutions.com
	URL	https://www.paradisosolutions.com/
Sources	https://www.paradisosolutions.com/solutions/virtual-classroom-platform	

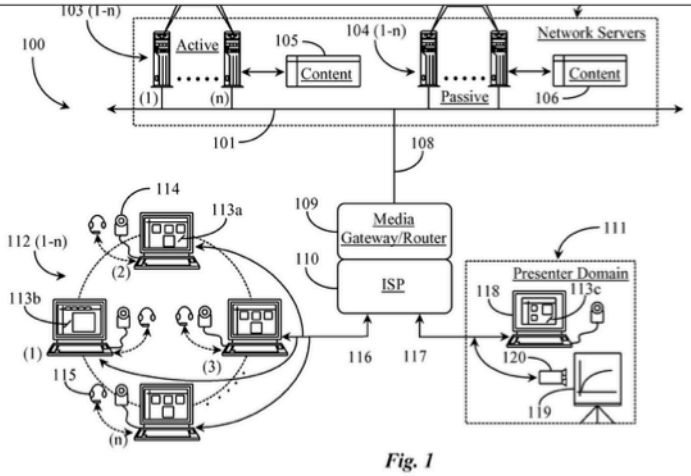
Section 2: Virtual Interactive Classroom Software – Patents

Patent 1. Panoramic learning system platform based interactive teaching method between teachers and students

Patent Number	CN103646574 A
Publication Date	19 Mar 2014
Inventor/s	Yan Jian
Brief Description	The invention relates to a panoramic learning system platform based interactive teaching method between teachers and students. **A network based virtual classroom is formed and accordingly the students in the same virtual classroom can see the teachers or the authorized students and explaining action on a virtual blackboard synchronously and accordingly the purpose of panoramic simulation of the interactive teaching of a real classroom is achieved.**
View Image	***N/A***
Sources	http://worldwide.espacenet.com/publicationDetails/biblio?CC=CN&NR=103646574A&KC=A&FT=D

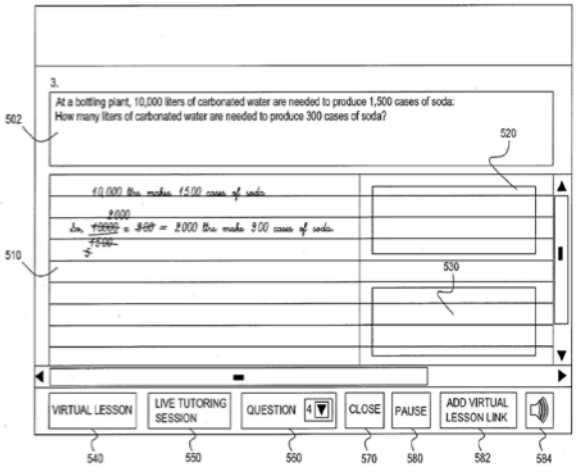
Contd. Section 1: Virtual Interactive Classroom Software - Patents

Patent 2. Interactive Learning Network

Patent Number	US20120301863 A1
Publication Date	29 Nov 2012
Inventor/s	Kamal Bijlani, P. Venkat Rangan
Brief Description	A system for providing network-based interactive learning including at least one network-connected passive server and at least one network-connected active server, each executing software from a non-transitory physical medium. The passive server provides a log-in function providing log-in for a student for a specific learning session, a provisioning function downloading reference materials for the particular learning session to the student's appliance after log-in, and a switch function connecting the student's appliance to the at least one active server after the reference material is downloaded. **The active server provides control of the specific learning session by managing communication between a presenter and individual logged-in students, and by transmitting control messages to the student's appliances,** the control messages initiated by activity of the presenter, and by causing display of individual portions of the reference materials downloaded to the student's appliances by the at least one passive server after log-in. **The system of claim 2, wherein the interactive interface further comprises a digital whiteboard responding to input from the presenter.**
View Image	 *Fig. 1*
Sources	https://www.google.lk/patents/US20120301863?dq=virtual+interactive+classroom+software+%2B+whiteboard+sharing&hl=en&sa=X&ei=SosbVY7iDozjuQTZh4KICQ&ved=0CDIQ6AEwAw

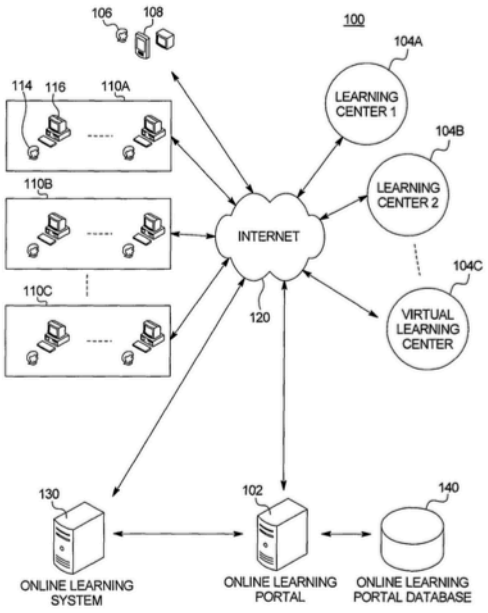
Contd. Section 1: Virtual Interactive Classroom Software - Patents

Patent 3. System and method for virtual content collaboration

Patent Number	US20120231441 A1
Publication Date	13 Sep 2012
Inventor/s	Sowmya Parthasarathy, Rajaraman Krishnan
Brief Description	**An information sharing system includes a database, an application module, a supervisor module, and a subordinate module.** The database stores user data associated with a supervisor and a subordinate and static data to be shared by them. The static data is associated with a topic for training. A collaboration system comprising: • **a first user module configured to display a virtual whiteboard comprising multiple layers of content to a first user among a plurality of users, each layer being associated with at least one of the plural users, wherein said first user is able to modify content only in said layer associated with said first user;** • a second user module configured to display the virtual whiteboard to a second user among the plurality of users; and • a sharing module configured to connect the first and second user modules over a network for real-time collaboration between the first and second users, responsive to a request by one of the first and second users, via the virtual whiteboard, • wherein the virtual whiteboard is at least initially oriented around a contextual topic determined by the first or second user that provides a subject for the collaboration.
View Image	
Sources	http://worldwide.espacenet.com/publicationDetails/biblio?CC=US&NR=20122 31441A1&KC=A1&FT=D

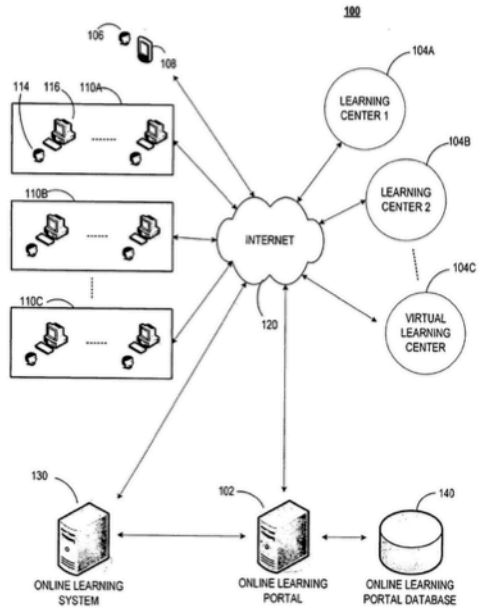
Contd. Section 1: Virtual Interactive Classroom Software - Patents

Patent 4. Teacher Assignment Based on Student/Teacher Ratios

Patent Number	US8170465 B2
Publication Date	1 May 2012
Inventor/s	Royia Griffin
Brief Description	**A system for assigning one or more teachers for teaching in one or more learning sessions comprises an online learning system having one or more nodes that creates one or more online learning sessions having corresponding student/teacher ratios.** A scheduling node schedules the one or more teachers for teaching the one or more online learning sessions. Database stores teacher qualification parameters associated with the student/teacher ratio of one or more learning sessions. A processing node adjusts a teacher qualification parameter associated with a teacher based on the number of scheduled learning sessions and corresponding student/teacher ratio of such learning sessions associated with the teacher. The scheduling node schedules the teacher for one or more online learning sessions having a student/teacher ratio that corresponds to the adjusted qualification parameter of the teacher.
View Image	
Sources	https://www.google.lk/patents/US8170465?dq=Virtual+room+tutor+to+students&hl=en&sa=X&ei=PdH2VIq6IMqLuATK9YLYDA&ved=0CBsQ6AEwAA

Contd. Section 1: Virtual Interactive Classroom Software - Patents

Patent 5. Educational System and Method Having Virtual Classrooms

Patent Number	US20080050715 A1
Publication Date	28 Feb 2008
Inventor/s	Mark Golczewski, Marc Corey
Brief Description	A system for instruction a student comprises an online learning system having one or more nodes that creates a learning session at a session start time. **The online learning node creates learning environments having a virtual classroom such that at the virtual class room, at least one student can view the activity associated with at least one other student with or without teacher involvement**. A database stores one or more matching parameters for matching on or more students with one or more teachers. A matching node selects a teacher for a plurality of students for the online learning session based on the one or more matching parameters. The online learning node places the plurality of students in the virtual class room after the matching node matches the teacher and one or more students and prior to the session start time. According to another aspect, the virtual classroom is created between a first online learning session and a second online learning session scheduled for a student consecutively.
View Image	
Sources	http://worldwide.espacenet.com/publicationDetails/biblio?CC=US&NR=20080050715A1&KC=A1&FT=D

Section 3: Virtual Interactive Classroom Software – Universities

University 1. University of Leicaster

Session Description	**Chat and the Virtual Classroom are called Collaboration tools in Blackboard. Blackboard provides two default collaboration sessions that you can use: Lecture Hall which is a Virtual Classroom session and Office Hours which is a Chat session.** You can either use these sessions or set up your own.	
Contact Details	**Address**	University Road, Leicester, LE1 7RH United Kingdom
	Telephone	0116 275 8025
	Email	csadmin@mcs.le.ac.uk.
URL	http://www2.le.ac.uk/offices/itservices/ithelp/services/blackboard/communication-and-collaboration/chat-and-virtual-classroom/create	

University 2. Victoria University

Session Description	**A Virtual Classroom is an online space designed for real-time collaboration and can be used to hold online classes**, webinars and meetings. **Virtual classrooms often include tools for text and audio chat, a shared interactive whiteboard**, sharing of files and the ability to record all interactions for future playback. The virtual classroom software used at Victoria University is called Blackboard Collaborate	
Contact Details	**Address**	PO Box 14428 Australia Melbourne
	Telephone	61 3 9919 4000
	Email	international@vu.edu.au
URL	http://learningandteaching.vu.edu.au/teaching_practice/blended_learning/elearning_environment/virtual_classroom/	

Contd. Section 1: Virtual Interactive Classroom Software - Universities

University 3. Tulane University

Session Description	• **Online course delivery** • **User-friendly interface to the virtual classroom** • "Concierge" type technical support. The distance learning (DL) IT staff focus is solely on DL students and supporting them and do not support other • IT or university issues. They are here for YOU! • Accessible from any location at any time via the Internet. We have had students complete their degree from an oil rig 100 miles off the African coast • **Using Blackboard you are able to: turn in assignments, view recorded lectures, participate in discussion boards and connect with faculty.** • **For classes requiring group work we can create virtual environment for your study group and meeting sites.** Even though on line your access to faculty and other students is virtually unlimited.	
Contact Details	**Address**	CAEPH, 1440 Canal Street, New Orleans, LA 70112,
	Telephone	504-988-1774
	Email	sphtmadmissions@**tulane.edu**
URL	http://www.sph.tulane.edu/publichealth/caeph/dl/virtual_classroom_convenience.cfm	

Contd. Section 1: Virtual Interactive Classroom Software - Universities

University 4. The University of Arizona

Session Description	The Online Rooms tool (powered by Blackboard Collaborate®) allows for you to create a virtual classroom within your D2L course site. Some of the options available within this tool include: • Live Video sessions: Using a webcam and microphone, you can hold live lectures with participants allowing them to see and hear you in real time. Moderators may also grant participants using a webcam and microphone the ability to speak to and be seen by the class. • **Whiteboard: The Virtual white board allows you to draw and type directly on a shared whiteboard as well as add clipart, screencaptures, and file uploads.** • **Desktop and Application Screen Sharing: Desktop screen sharing allows you to display everything visible on your computer screen to your participants, while Application sharing allows you to restrict visibility to a specified program.** • Breakout Rooms: Use breakout rooms to allow participants to engage in smaller group chat sessions, while retaining the ability to return to the main lecture at any time. • Polling: Ask live questions and have participants select an answer to help improve engagement in the session. Session Recording & Archive: Record your live session for later access by you and your participants. The recording will show up in your D2L course site within an hour of the session closure.	
Contact Details	**Address**	Tucson, AZ 85721, USA
	Telephone	(520) 626-6804
	Email	D2L@email.arizona.edu
URL	http://help.d2l.arizona.edu/content/online-rooms-overview	

Contd. Section 1: Virtual Interactive Classroom Software - Universities

University 5. Federation University of Australia

Session Description	Blackboard Collaborate (previously Elluminate) is the virtual classroom software used by FedUni. All references to Elluminate should be read as a reference to Blackboard Collaborate. Blackboard Collaborate allows you, the 'moderator,' to host online lectures and tutorials for up to 100 participants. Inside the room, you can: • **Conduct audio and video conferencing** • Present slideshows • **Collaborate on an interactive whiteboard** • Share desktops • **Send and receive files and instant messages** Record the session for later use.	
Contact Details	**Address**	Mount Helen VIC 3350, Australia
	Telephone	(03) 5327 6151
	Email	clipp@federation.edu.au
URL	http://federation.edu.au/staff/learning-and-teaching/clipp/elearning/virtual-classroom	

University 6. University of California Riverside

Session Description	The Collaboration Tools allow users to participate in real-time lessons and discussions. Examples of these sessions include real-time, online classroom discussions, guest speaker led sessions, Teaching Assistant sessions, and live question-and-answer sessions. **Sessions can be recorded and made available for review. There are two types of Collaboration tools available in Blackboard: Chat, a simple, text-based messaging tool; and Virtual Classroom, an interactive classroom session where users can use a virtual whiteboard, share a browser and ask questions.**	
Contact Details	**Address**	900 University Ave. Riverside, CA 92521
	Telephone	(951) 827-4741
	Email	helpdesk@ucr.edu
URL	http://cnc.ucr.edu/ilearn/fac_intermediate.html	

Contd. Section 1: Virtual Interactive Classroom Software - Universities

University 7. The University of Queensland

Session Description	**UQ's virtual classroom is a web conferencing tool that enables a group of people to work together in real time online. The virtual classroom provides the ability to interact and collaborate in real time with other participants in the course using voice, text chat, shared desktops, whiteboards and webcams while physically separate.** Virtual classrooms like physical classrooms can be used for an incredible range of activities, such as: • group discussions • break out group activities (in break out rooms) • practical demonstrations (of software/tools/procedures) • peer presentations • collaborative drawing • eliciting audience responses through polls (clickers style), surveys and comments • delivering lecture content • accessing multimedia (YouTube videos/audio clips) • playing games and using simulations • research or practicum student supervision • recording live sessions	
Contact Details	**Address**	Brisbane St Lucia, QLD 4072 Australia
	Telephone	61 7 3365 1111
	Email	marketing@its.uq.edu.au
URL	http://cnc.ucr.edu/ilearn/fac_intermediate.html http://www.uq.edu.au/	

Feedback

Any feedback on the answer provided would be greatly appreciated by the researcher and [redacted]

Query Details

Customer Name	[redacted]
Question	
Allocated Time Frame	
Answer Format	
Team Leader	
Researcher	