UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Antti Männikkö

AGILE DEVELOPMENT MODEL IN MULTI-PROJECT ENVIRONMENT

Master's Thesis
Degree Programme in Computer Science and Engineering
May 2020

Männikkö A. (2020) Agile Development model in Multi-project Environment. University of Oulu, Degree Programme in Computer Science and Engineering. Master's thesis, 53 p.

ABSTRACT

Agile software development has slowly but steadily become a part of modern software industry. Older development models, such as waterfall, are seen obsolete and less flexible in today's software development environment. But how do current agile methods fare in place where there are multiple ongoing projects and sprint goals can suddenly change? This thesis observes current agile models in multi-project software development environment and tries to improve current model the evaluated team has.

The target team is a small, remote branch of medium sized organization consisting of nine members. The team utilized Scrumban – Scrum with Kanban - but it faced some problems. Rather than having one or two big projects, the branch did many smaller scale projects, and current model did not fit in such environment as good as it was thought. The team had some difficulties on communication between different projects, there was a noticeable amount of "ad hoc work" (issues that were not logged anywhere and were done independently) and metering the progress of the project was hard sometimes. A better model for the team was in need.

First, the evaluation team is briefly introduced and some more info on Scrumban, the Sprint cycle and what kind of projects the team usually has is given. Data gathering plans - surveys and interviews - is also described. After examining the survey and interview results, quality measurement, work cycle, resource allocation and learning were the topics that were widely discussed. Four goals were made: streamlining the work cycle, a way to measure progress, making learning easier during the work cycle and way to monitor the ad hoc tasks. After some analysis, the decision was to take values and principles from Kanban, such as workflow visualization and limiting Work In Progress (WIP).

The evaluation period lasted through two Sprints. The evaluation period was during COVID-19 pandemic but fortunately that did not affect that much to it. After the evaluation, feedback session and discussion about the new model was had. The overall impressions on better Jira board visualisation (personal swimlanes) were overwhelmingly positive and team decided to keep them after the evaluation ended. On other features, the team felt that they were good idea but needed some more time on execution. Overall, the team felt like the new model improved their ability to work agile but there was still some field on improvement, particularly on the learning field.

To sum up the thesis, the four principles that might help the team to choose and agile method are introduced: *find what team is good at and what is lacking, do not be afraid of experimenting, think what parts you could automate and reflect on your changes*.

Keywords: Agile, Scrum, Kanban, continuous delivery, workflow

Männikkö A. (2020) Ketterän kehittämisen malli moniprojektiympäristössä. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 53 s.

TIIVISTELMÄ

Ketterä ohjelmistokehitys on hiljalleen mutta tasaisisesti tullut osaksi nykyistä ohjelmistoteollisuutta. Vanhemmat kehitysmallit, kuten vesiputous, nähdään vanhentuneina ja vähemmän joustavina nykypäivän ohjelmistokehitysympäristössä. Mutta miten nykyiset agile-menetelmät pärjäävät paikassa, missä on useampi projekti meneillänsä ja sprintin tavoitteet voivat muuttua yhtäkkiä? Tämä tutkielma havainnoi nykyisiä agile-malleja moniprojektisessa ohjelmistokehitysympäristössä ja yrittää parantaa arvioidun tiimin nykyistä mallia.

Kohdetiimi on pieni yhdeksän ihmisen etäinen haara keskisuuressa yrityksessä. Tiimi käytti Scrumbania (Scrum Kanbanilla), mutta heillä on ollut ongelmia. Yhden tai kahden suuren projektin sijaan haaralla on paljon pieniä projekteja, ja nykyinen malli ei toiminut kyseisessä ympäristössä niin hyvin, kuin ajateltiin. Tiimillä on ollut vaikeuksia tiedonvälityksen kanssa projektien välillä, huomattava osa työstä tehtiin ”ad hoc työnä” (työ, jota ei kirjattu minnekään ja tehty itsenäisesti) ja projektin kehityksen mittaaminen on vaikeaa välillä. On aika paremmalle mallille.

Ensiksi kohdetiimi on esitelty lyhyesti ja enemmän tietoa Scrumbanista, Sprintjaksosta ja minkälaisia projekteja tiimillä yleensä on annettu. Tiedon keruu menetelmät (kyselyt ja haastattelut) on myös kuvattu. Kyselyn ja haastattelun tulosten tutkimisen jälkeen kävi ilmi, että laadunmittaus, työjakso, resurssien allokoiminen ja oppiminen olivat eniten keskustelua herättäviä aiheita. Tehtiin neljä tavoitetta: työjakson virtaviivaistus, tapa mitata kehitystä, oppiminen työjaksolla helpommaksi ja ad hoc tekemisen valvonta. Pienen analyysin jälkeen päätös on ottaa arvoja ja periaatteita Kanbanista, kuten esimerkiksi työnkulun visualisointi ja keskeneräisen työn (Work in Progress, WIP) rajoitteet.

Arvioinnin ajanjakso kesti kahden Sprintin verran. Arvioiminen oli COVID-19 pandemian aikaan, mutta se ei onneksi vaikuttanut juuri paljoakaan siihen.

Arvioinnin jälkeen oli palautesessio sekä keskustelua uudesta mallista. Vaikutelmat Jira-taulun visualisointiin (henkilökohtaiset työjakaumat) olivat yleisesti ylivoimaisen positiivisia ja tiimi päätti pitää ne arvioinnin loppumisen jälkeen. Tiimin mielestä muut toiminnot olivat hyvä lisä, mutta tarvitsivat lisää aikaa toteutukseen. Yleisesti tiimistä tuntui, että uusi malli paransi heidän kykyänsä tehdä ketterästi, mutta olisi vielä muutamassa kohtaa parantamista, etenkin oppimisen kannalta.

Lopputyön summaamiseksi, neljä periaatetta, jotka saattavat auttaa tiimiä valitsemaan ketterän menetelmän esitellään: *löydä se, missä tiimisi on hyvä ja mitä puuttuu; älä pelkää kokeilla; mieti, mitä kohtia voit automatisoida; ja pohdiskele muutoksiasi*.

Avainsanat: Agile, Scrum, Kanban, jatkuva toimitus, työnkulku

TABLE OF CONTENTS

FOREWORD

When the first meeting about possible topics of my master thesis was had, I was quite overwhelmed by the possibilities. Mikko had created a wide spectrum of ideas ranging from renovating old code libraries that were almost deprecated at this point to designing a wireless mobile testing platform. One suggestion particularly stood out from the rest - expanding the used agile model to fit more into team's environment. The model used then was inherited from the headquarter teams, which most of them were larger and focused on one or two big projects at a time. At the time, I was quite unfamiliar with the values and principles of agile development - the only experience I had was a couple of months Scrum. As I further delved on the subject, I found that agile meant way more than just how the development team does the work cycle. Mindsets, learning processes, waste elimination, and so on - agile software development covers a wide array of things! The road to a finalised model was quite demanding, and a lot of things could have been gone better, but the payoff and learning experience was quite worth it.

First, I would like to thank my supervisor Ulrico Celentano. His input at the start helped tremendously. Second, a huge thank you for my technical supervisor Mikko Holappa, who provided me learning material, as well as corrections. Lastly, the huge thanks to software architect Ville Mattila, who kind of acted as a second technical supervisor for the thesis, and Scrum Master Jesse Kirjavainen, whose help during the evaluation period was huge help, and who provided and gave permission to to use the Jira script.

Oulu, Finland December 13, 2020

Antti Männikkö

ABBREVIATIONS

AAIM	Agile Adoption and Improvement Model
ASD	Adaptive Software Development
ART	Agile Release Train
CDP	Continuous Delivery Pipeline
CE	Continuous Exploration
CI	Continuous Integration
CD	Continuous Deployment
DSDM	Dynamic Systems Development Method
FDD	Feature-Driven Development
IID	Iterative and Incremental Development
JAD	Joint Application Design
JRP	Joint Requirement Planning
LPM	Lean Portfolio Management
OO	Object Oriented
PDCA	Plan-Do-Check-Act
PO	Product Owner
PPM	Project Portfolio Management
QA	Quality Assurance
RAD	Rapid Application Development
RoD	Release on Demand
SAFe	Scaled Agile Framework
TDD	Test Driven Development
TPS	Toyota Production System
WIP	Work In Progress
XP	Extreme Programming

1. INTRODUCTION

Compared to the rapid advancement that has happened to computer technology and software design, the models and paradigms of how software developers produce software as a group have not really changed that much. Half a century old methods such as waterfall still work as fine as they did long ago, though in modern development environment – that takes advantage of technology such as continuous integration tools and where customer is more involved in project – they seem lacklustre.

The methods for speeding up the production while avoiding unnecessary work, stem from manufacturing business, where after Second World War different industries began producing for bigger masses [1]. Similarly in the software industry, customized software for different users meant that waterfall's model way of gathering requirements for months and then starting design and development process, left very little space to change anything in the middle of the process. Old models seemed obsolete for many developers, especially since tools like Git and frameworks such as .net made development process smoother. Agile methods and their predecessors helped software development to become more flexible at the cost of comprehensive documentation.

After the *Manifesto for Agile Software Development* was published [2], many organizations began to invest more on the ways they develop software and how to get the communication with customer as part of the development cycle. Scrum, Extreme Programming (XP) and Kanban have become a part of modern software development.

One of the hardest parts of becoming an agile team is fully adopting agile software methods. Not all teams are equally sized, do only one project at a time, or have a clear management structure. Many teams struggle to actually become an agile team; they adopt some practices, they do daily stand up meetings, they create user stories, they improve their software quality. But implementing only some of the practices may lead to only minor improvements, as in making the team not to focus on areas they lack and only improving on where they are already good at [3].

One magnificent part about agile methods is that they are very flexible. If one was to interview two teams that both are using Scrum, they would find that while both use the principles found in every Scrum guide, the way they follow and execute those principles differ. This is also a double-edged sword - every organization technically can become an agile one by adapting agile method and tailor it around the current work environment. On the other hand, some strategies that have been found successful in one environment may not translate into another and new agile teams that have not yet fully understood the principles may need some time to get the basics working.

This thesis examines and improves the current model for small teams that, instead of having one or two big projects, have multiple small ones going on. Which models are as applicable to this kind of team and are the some principles that need more attention? The evaluated team utilizes Scrumban (Scrum with elements from Kanban). Although the old approach has proven effective, many team members feel that some aspects, such as communication between projects, need improvement.

First, the Related Work chapter goes through history of agile and some models. Second, the Analysis and Design goes through survey and interview process and results, and the new method is introduced. The Evaluation chapter goes through the evaluation period, and examines the results. Finally, in Discussion the overall success of the tested method, as well as some possible further work, is discussed.

During the evaluation, the global COVID-19 pandemic made that instead of team members being face-to-face in office, everyone was forced to work remotely. While this affected the original evaluation plan significantly, it also made an opportunity to look the topic from a different perspective. Some of the effects are discussed on Evaluation and Discussion chapters.

2. RELATED WORK

2.1. Sequential models

Although this study focuses on agile development methods, acknowledging older, sequential models could potentially lead to some findings that help refining the chosen model. In some cases discussed below, the old but well tested models might even be better than modern but highly conceptual level agile models.This section focuses on two known and used models: waterfall and V-model.

2.1.1. Waterfall

There are various waterfall models but Royce's original implementation shown in Figure 1 is still widely used. Interestingly, Royce presented an iterative version where processes can also go backwards if needed but stated that it might be a risky approach because of testing occurring so late that going back to redo software requirements is risky [4].

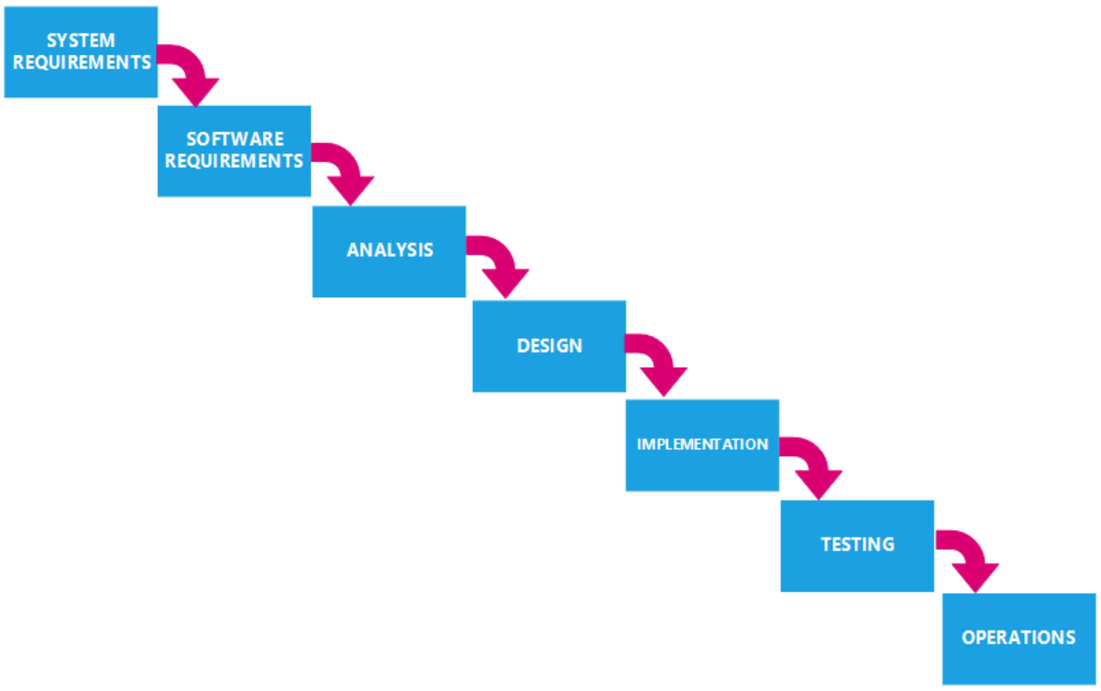


Figure 1. Standard waterfall model.

The big plus of waterfall is that due its linearity it is fairly easy to implement and follow; phases always go in certain order and there is little to none departure from the original plan. Good documentation of system requirements also helps during development phases. In projects where requirements are defined from the start (and it is known that very little changes) waterfall works great [5].

Unfortunately, the linearity is also waterfall's weak part. In project where requirements are partially unknown or they change during development, this means there is considerable amount of rework being done [5] [6]. Other issues are time spent on writing and approving documentation and poor customer collaboration [6].

Waterfall could be considered if projects are long (at least year or more), requirements are known and they will not change, development team wants robust architecture and documentation and client communication is not needed. In the context of this study, waterfall's disadvantages outweighs its advantages.

2.1.2. V-model

The V-model is an extension of waterfall. As seen in Figure 2, the distinction to its predecessor is that a left, descending part is reserved for defining the project. After that, comes the middle part, implementation. Finally, the right, ascending part is reserved for testing and integration. V-model has more weight on testing and acceptance [7][8].

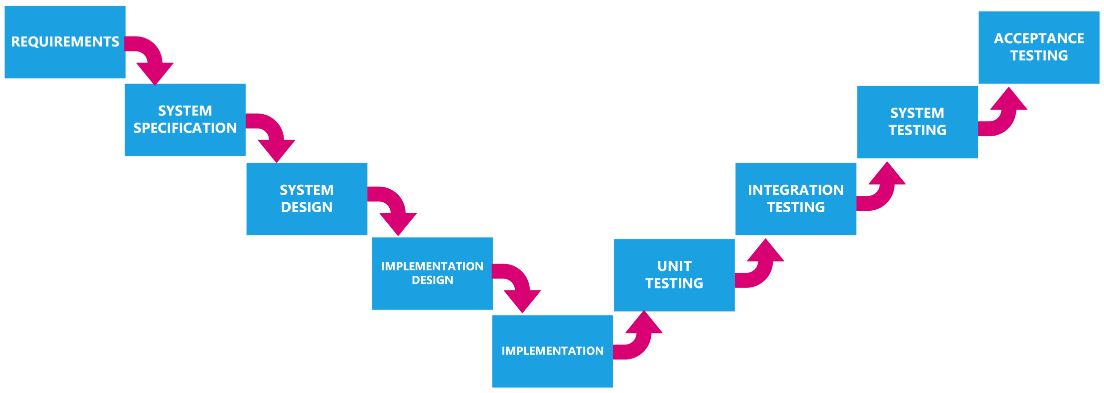


Figure 2. Standard V-model.

Just like waterfall, the V-model is easy to follow. However, in case of something on requirements changes, current work can be stopped and new v-cycle can be started, essentially redoing the project [7]. Test plans improve the quality of software and makes sure the requirements are met [9].

While there is possibility to adapt to upcoming changes, this also means that documentation has to be modified to fit current requirements [5]. This also means that V-model is hardly effective in terms of cost. In the end, V-model is not that more flexible than waterfall [9]. V-model is suitable for teams that want to ensure the quality of the product. Otherwise, it might be a little too inflexible and expensive to use in software development.

2.2. Rapid Application Development (RAD)

Rapid Application Development (RAD) is a general term to describe pre-agile methods that differed from waterfall model [10]. RAD methods generally share many similar principles with agile and methods, such as Adaptive Software Development (ASD), are direct derivatives from them. The main difference is how RAD and agile do development process; agile generally is more focused on iterative cycles than RAD. [11].

RAD was designed to "give faster development and higher-quality results than those achieved with the traditional cycle" [12]. The complexity of code was constantly rising, and developing software with previous software development methods were deemed too time consuming. RAD's cycle of test, modify and code was meant for speeding up the development process. RAD also emphasizes on metrics, for example measuring cycle-time [12].

While there are differences among definitions, the typical RAD cycle consists of four phases:

- Requirements planning - determine requirements.
- User design - workshop with users and planning the design.
- Construction - development and validation.
- Cutover - acceptance testing and training [12].

During the early phases of project, RAD engages with customer through Joint Requirement Planning (JRP) and Joint Application Design (JAD). The former is usually done in planning to map requirements and to do a research on customer, while the latter is used in user design phase to ensure quality and user experience [12].

Changing from traditional methods to RAD was not painless transition. People with little or no experience on Object Oriented (OO) paradigms had hard time adapt. Management would focus only on the speed aspect of RAD and got unrealistic expectations. Time for design processes was short or, in some cases, they were completely ignored [10]. RAD needed a team that could collaborate with customer - adding social skill in mix [13].

Despite of rough start, utilizing RAD was a success for many teams. Bringing customer to development process made it easier to narrow requirements and more automated tools were utilized [13]. The new technology such as visual programming and client/server architecture helped to ease in new development model [10].

2.3. History of Agile

The Manifesto for Agile Software Development was created on February 11-13, 2001 at the first Agile software development meeting [2], though the term "agile" has been used to describe fast paced manufacturing methods before that [14]. While the term Agile is relatively young, software development methods that are adaptive and incremental can be traced back to the mid of last century.

Iterative and Incremental Development (IID) methods can be traced back to 1930's at Bell Labs where Walter Shewhart proposed a cycle of "plan-do-study-act" to improve quality. However, more important for software development, was NASA's Project Mercury in 1950's where the team developed software in small increments. After that, more projects began to use IID approach during 70's [15].

Gladden wrote in his 1982 article how the waterfall approach seemed to conduct to lengthened schedules with the final product not satisfying the new requirements. Gladden proposed a model called "The Non-Cyclical (Hollywood) Model" that not only focused more on short period objectives, it also improved communication between the developer and customer by physical demonstrations [16].

Toyota Motor Corporation began heavily investing on working environments and methods during 1980's. The now famous Toyota Production System (TPS) allowed Toyota to become one of the biggest car manufacturers, just behind Ford and General Motors GM in terms of numbers. Where the two aforementioned methods could rely on mass production, Toyota had to rely on smaller assembly lines. Toyota's change on both management and worker level helped the relatively small manufacturer to maximize its output [17]. TPS introduced many concepts that are essential to agile thinking and serve as a foundation to many agile methods [17]:

- Long term philosophy - decision making based long term planning but also factoring shorter, financial objectives.
- Right process makes right results - continuous work process planning that visualizes the whole and ties up the processes. Customizing production process based on customers needs and wants. Avoiding overproduction and balancing the workload. Introducing culture that when encountering a problem, the current process is stopped to solve the it.
- Production of added value through development of people and partners - growing leaders that understand the work process, obey the philosophy and teach it to others. Making teams and competent people that embrace the philosophy. Respecting collaboration partners and suppliers and helping them to improve.
- Continuous solving of background problems enhances learning - going to place to gain further understanding of the situation (genchi genbutsu). Making decisions slowly by considering different choices.
- Constant reflection - the Kaizen (constant small steps of improvement) principle.

After Toyota's success, many notable corporations began to duplicate the system, which eventually paved way to Lean mindset [17].

In the 90's many agile methods were developed to fit more in constantly evolving software development landscape. During this era, widely popular methods such as Scrum and XP were created [15].

Agile is not only a set of instructions - one needs to have a mindset for agile development. The reason why Toyota succeeded was not only streamlining the car making process, but applying a way of thinking to making the product. So-called Lean (covered in Section 2.4.4) thinking helped the workers to easier understand the values of Lean and maximize the output. Eventually, in 90's, Lean thinking stranded to software development and helped companies like eBay and Microsoft to grow exponentially [18].

2.3.1. Kaizen

Kaizen (continuous improvement) is a concept that is usually linked with Lean mindset. While the concept is somewhat ambiguous, it could be summarized as a "repeated small steps of improvement from everyone involved in process" [19]. Continuous improvement is a value that every agile method strive for. For example, in Lean this can be seen in waste management and learning in individual and team level.

Kaizen shares its origin with TPS. Masaaki Imai introduced Kaizen in 1986 in Toyota and it quickly became part of the learning system [20]. The Plan-Do-Check-Act (PDCA) cycle gave a good framework to problem finding and solving.

Kaizen programs were inspiration to agile's improvisation programs; the idea that everyone - developers, managers, etc. - involved in process should join in learning session and share their knowledge stems from it. However, in some organizations participating in Kaizen is voluntary [19], where in agile they are core part of process.

2.3.2. Manifesto for Agile Software

For two days, a group of like-minded people gathered around and discussed on current trends of developing software. Together, they defined the term 'agile' and created the manifesto, in which they proclaimed that they valued [2]:

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

When looking at different agile methods in-depth, it is clear that each of these four values is taken into consideration. However, the way these are achieved differ from method to method; how to respond to alternating requirements, how to get customer as a part of development process, etc.

Agile manifesto also determined 12 different principles [17]:

- Satisfying customer through continuous delivery is highest priority.
- Being able to adapt to change.
- Constant software delivery.
- Daily collaboration with business people and developers.
- Supporting motivated people and building project around them.
- Exchanging important information through face-to-face conversation.
- Metering progress with working software.
- Promoting sustainable development.
- Constant consideration on technicals detail and design.
- Emphasis on simplicity.

- Self-organizing teams create best values.
- Constant self-reflection and improvement.

As with the values, agile methods value these principles.

2.4. Agile Methods

2.4.1. Adaptive Software Development (ASD)

Adaptive Software Development (ASD) is an early derivation of RAD (Section 2.2). ASD was planned to work on larger size teams and gave more thought on management and adapting on current changes [21].

ASD introduced the concepts of adaptive cycles and learning loops to software development. With adaptive cycle planning and reviews, the team can validate current version and evaluate development efficiency. ASD highlights that with adaptive management and learning as a team, processes become less rigid and workflow becomes easier to manage [21].

The adaptive cycle is made of five parts (seen on Figure 3): project initiation (where requirements are gathered and the project begins); adaptive cycle planning; concurrent component engineering (development); quality engineering and final quality assurance (QA) and release [21]. Parts from cycle planning to quality review form the adaptive cycles, also known as learning loops. At the start of the cycle project time-box (a determined time when the project is completed) and mission is determined a highest risk items are selected. At the end, learning activities are practised. These differ team by team, but some examples are customer focus group review and software inspections [22].

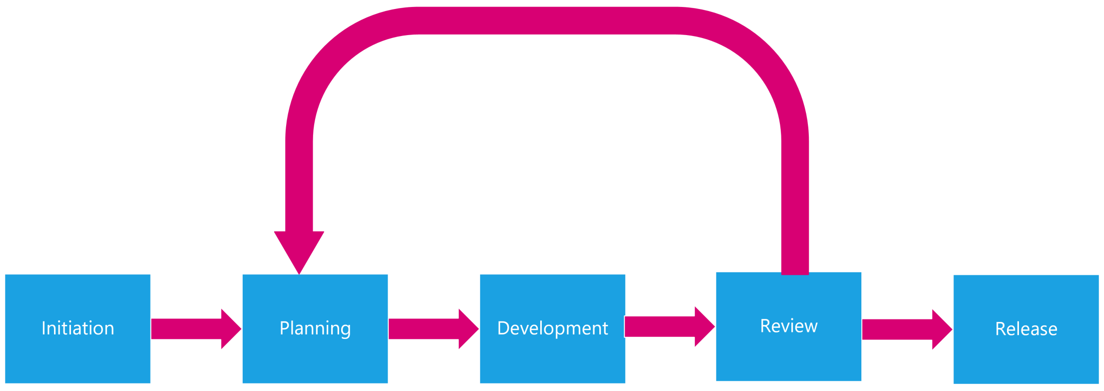


Figure 3. Adaptive Life Cycle.

While ASD identified that it is important to be more adaptive and that learning during the process is what everyone should do, it is hardly a used method. The underlying problem is that it might be too abstract - hardly any practices are defined [23].

2.4.2. Dynamic Systems Development Method (DSDM)

Dynamic Systems Development Method (DSDM) is an agile method that factors in whole project life cycle. It can be seen as a successor to RAD [24]. DSDM has eight principles [24]:

- Focus on the business need - reflect what are the projects goals and make decisions based on them
- Deliver on time - "quite often the single most important success factor."
- Collaborate - encourage stakeholders and team members to work as one.
- Never compromise quality - the level of quality should be agreed upon.
- Build incrementally from firm foundations - plan the scope and wireframes for the project but leave room for upcoming requirements and improvisations.
- Develop iteratively - the loop of design, develop, feedback.
- Communicate continuously and clearly - prefer face-to-face communication, keep everyone involved in project at same wavelength through daily stand-ups, workshops etc.
- Demonstrate control - agree on business objectives, measure progress.

DSDM development process is more abstract than others. In a nutshell, the project is divided into three phases: functional model iteration, design and build iteration and implementation. Each one of these phases are cyclical, and in deterministic situations the process would go from feasibility and business study to functional to design to implementation after maybe a couple of iterations per phase but as stated before, projects constantly change. Rather, DSDM wants the iteration process to be specific to different projects [22]. The basic process is illustrated in Figure 4.

2.4.3. Feature-Driven Development (FDD)

Most agile methods emphasise the importance of management and workflow. Feature-Driven Development (FDD) is a distinct method in that its only focus is on design and building phases [25]. FDD consists of five processes [25]:

- Develop an overall model - choose an appropriate model based on the scope, context and requirements.
- Build a features list - pick major feature sets and divide them into smaller sets, then finally let users review the list.
- Plan by feature - prioritize features and schedule milestones.
- Design by feature - select features and team(s).

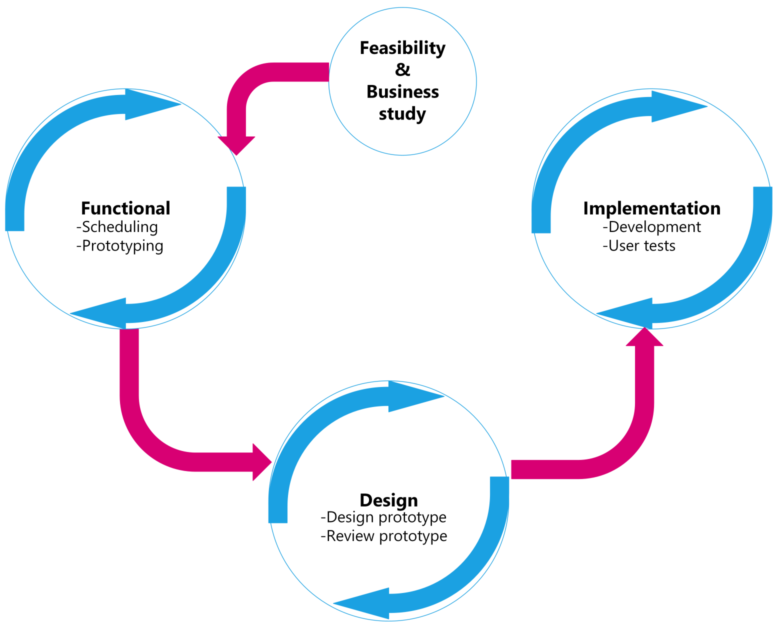


Figure 4. DSDM process in optimal situation.

- Build by feature - start developing the selected features.

FDD also has eight practices: domain object modelling, developing by feature, code ownership, feature teams, inspection, regular build schedule, configuration management and reporting of results. The last one is noteworthy as it is exclusive to FDD. The project completion p is calculated by following equation:

$$p = \frac{\sum f}{t}, \tag{1}$$

where t is total number of features in project and f is a feature value that is calculated as:

$$f = \sum w \tag{2}$$

where w is weight of completed features. Each feature has different weight, dependent on previous data. Milestones are: domain walk through, design, design inspection, code, code inspection and promote to build, assuming values 1%, 40%, 3%, 45%, 10% and 1% respectively [22].

2.4.4. Lean

In manufacturing world, Lean is one of the oldest methods that can be considered agile - its roots lie in 80's Japanese automotive industry.

In contrast to younger methods like Scrum, Lean is more of a mindset than lines [3]. It is based on seven principles [26][3]:

- Eliminate waste - anything that does not help to create or slows down the development of product can be considered waste and must be removed.

- Amplify learning - learn from results and feedback of the project to improve on developing software.
- Decide as late as possible - important decisions are better made later, when there are more facts than speculations.
- Deliver as fast as possible - eliminate delays and gather feedback earlier
- Empower the team - with effective work environment and expertise work flow becomes better.
- Build integrity in - a good product feels coherent and useful even when time has passed.
- See the whole - do not only focus on your own expertise field, see what are the interests and adjust properly.

Even though Lean is way older than many other agile methods, Lean in software development is fairly new. Poppendiecks's book "Lean Software Development: An Agile Toolkit" was the first to describe Lean in a context of software development [26]. While many concepts in the original Lean method, such as the seven principles, can be applied to software manufacturing, some had to be changed to fit better. As an example, overproduction is hardly a concern in a web application development but feature creep is [26].

Lean's way to make processes less time consuming through eliminating waste has inspired other agile methods to streamline the workflow. The seven wastes in software development are [26][3]:

- Partially done work - the kind of software that is not part of the current environment or in production and is only partly complete.
- Extra processes - documentation that is only for the sake of keeping things documented brings almost no value.
- Extra features - anything that is out specifications makes the source code more complex and can lead to unexpected errors that without the extra code would not be there.
- Task switching -the fastest way of doing multiple projects is to do them one at a time, as switching between them takes time.
- Waiting - delay in every stage of software development, be it either short or long, is waste.
- Motion - if big part of work day is spent between moving through office to exchange information, it generates waste.
- Defects - the sooner a defect is detected, the less waste is causes.

2.4.5. Crystal Family

Crystal family contains several methods, named after different colours. Commonly used ones are clear, yellow, orange and red. The darker the colour, the more people are involved in the process; clear has eight or less people, whereas red can have anywhere between 50 to 100 [27].

To choose a fitting Crystal method, one should consider team size first. After that, criticality - damage caused by defect - should be considered. Criticality categories are Comfort (C), Discretionary money (D), Essential money (E) and Life (L) [27].

Crystal has seven properties, the first three being the most important and found in all projects:

- Frequent delivery - iteratively producing software that is delivered to users.
- Reflective improvement - discussing on what works and what does not at least once per month.
- Osmotic communication - keeping team members up to date with relevant information and hearing their questions ans suggestions, prefer on being in same room together when discussing on things.
- Personal safety - being able to speak of issues without of fear of being ridiculed.
- Focus - Prioritizing tasks and making sure the developers have time to develop the project without interruptions.
- Easy access to expert users - using experienced users to conduct feedback on current state of development, at least having a semi-weekly meeting with them.
- Technical environment with automated tests, configuration management and frequent integration - continuous integration with testing to prevent and catch possible errors [27].

Crystal's cycles are a group of small iterations, followed by integration. Depending on the method, cycles can last less than a work day to multiple days [27].

2.4.6. Scrum

Scrum right now is the most used agile method with about half of the software development organizations using it [28]. Its most distinct features are sprint and assigned roles: Product Owner (PO, one person that maintains product backlog - a board where tasks that are not in sprint are located), development team (multiple people that work based on Scrum principles) and Scrum Master (helps team to understand Scrum and maximizes the usage) [29].

Scrum has five values that are essential for every Scrum team [3]:

- Commitment - doing project as a team, giving or listening feedback and taking responsibility.

- Respect - trusting other team members, scheduling enough work time for project, letting everyone choose their tasks.
- Focus - doing tasks that are planned for the sprint, agreeing on what are the priorities.
- Openness - listening to other people opinions, synchronizing team members (on what are the technical details, goals and plans).
- Courage - not blaming other people when planning or requirements are poor, understanding users, not striving for perfect product but one that is good enough.

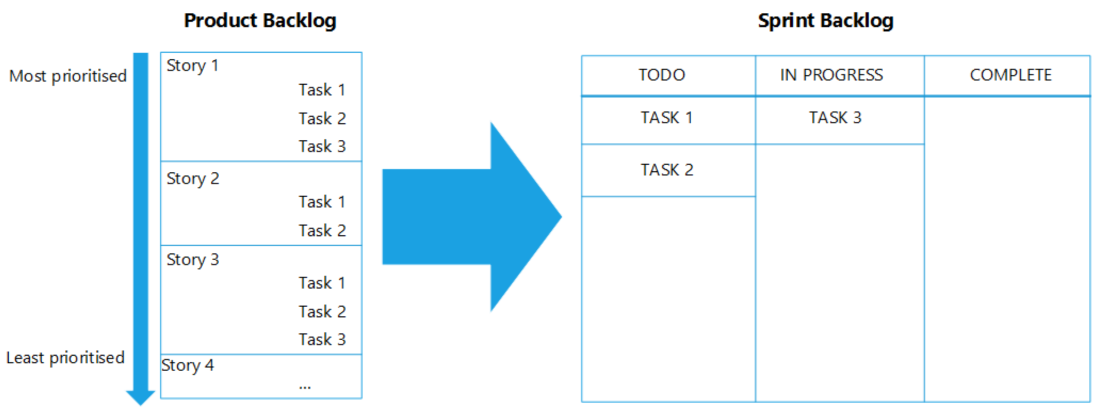


Figure 5. Scrum backlogs.

Development cycles are done in sprints (Figure 6), a maximum of one month long period where the Sprint Goal is set and worked on. A typical sprints starts on Sprint Planning where entire team plans on contents of starting sprint and selects tasks from the product backlog and moves them to sprint backlog. After the Sprint planning, the development starts. Daily Scrum is held every work day to check that development team is working towards the Sprint Goal, to determine what each member is working on and to keep each synchronized on what is going on right now. Daily Scrum is usually 15 minutes long. Sprint Review is done at the end of current sprint where clients and stakeholders are also invited. Sprint Reviews purpose is to show done tasks and discuss further procedures regarding the project. Lastly, Sprint Retrospective (or Retro for short) is held within the Scrum team to self reflect on how the sprint went, what could have been done better, what has worked and what should be improved upon next sprint [29].

Empiricism, the theory which states that experience on things is the primary way to gain knowledge, is foundation to Scrum's three pillars [29]:

- Transparency - everyone involved in the process has understanding on what is going on and what is the definition of done (Daily, Planning).
- Inspection - frequent going through of the progress and trimming undesirable variances (Daily, Retro).

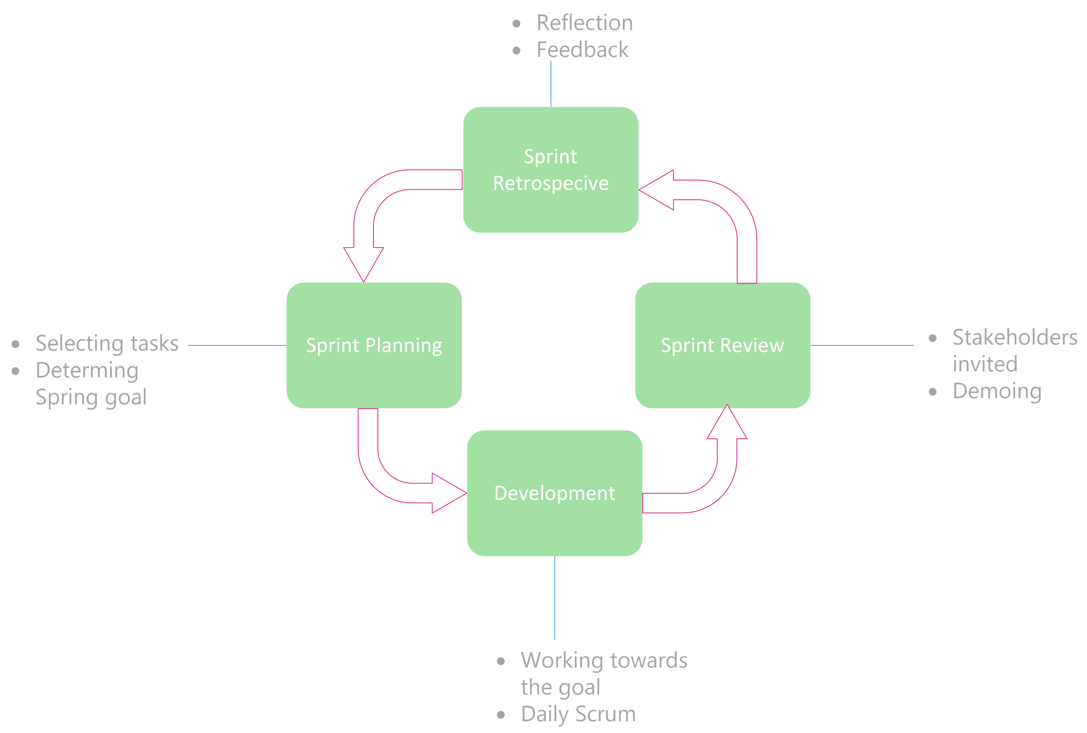


Figure 6. Scrum sprint.

- Adaption - adjusting items based on current context on the project (Planning, Review).

Scrum is suitable for small teams because it allows people to work independently, while not sacrificing visibility or communication. Sprint events make sure that every team member is up to date with current development and that there is no miscommunication error between the customer and developers [30].

2.4.7. Extreme Programming (XP)

Extreme Programming (XP) model was created during the late 90's. Its core values are feedback (rigorous unit and acceptance testing and pair programming), simplicity (splitting features into smaller chunks called stories and "keeping it simple"), communication (continuous cooperation with customer) and courage (fast code integration) [3]. XP's focus is more on programming and building quality code rather than team management like in Scrum. Teams using XP are usually small to mid sized teams, a maximum of ten people, and the main mindset is to "embrace change". Project requirements are constantly changing and some stories (depicted features) become more urgent; the way to address these changes are to get feedback as often as possible and responding to the change requests [31].

XP has 13 practices that are centred around making better software [31]:

- Planning game - make plan based on customer's parameters.

- Small releases - chop the "complete" product into small pieces and release them after a couple of months in development.
- Metaphor - create metaphor(s) with customer that create foundation to product.
- Simple design - "say everything once and only once".
- Tests - every new code component and story should have tests and they must run and pass.
- Refactoring - design constantly evolves through the development and implementation should reflect that.
- Pair programming - each developer has a pair, and they work with one device.
- Continuous integration - new functionality is integrated with current system as soon as possible.
- Collective ownership - every developer involved in project is responsible for the source code.
- On-site customer - customer is full-time involved in product development.
- 40-hour weeks - no overtimes.
- Open workspace - everyone works in one room.
- Just rules - rules have to be abided but rules can also be changed to fit.

As seen in the practices, XP is more concrete on how to develop software and work as a team. Concept like pair programming and collective ownership are mostly just used when discussing about XP. Teams that do XP usually also adopt Test-Driven Development (TDD) practices. In TDD, developers implement automated unit tests before writing any piece of code. After that, they make the implementation based on the premise that all test must pass [32].

Scrum and XP share some similarities with each other such as stories and working cycles. The major differences are sprints, daily Scrums and Product Owner/Scrum Master roles from Scrum and importance on testing, continuous integration, simplicity, improvement and pair programming from XP. Because of the similarities, some Scrum teams have taken some practices and principles from XP, such as TDD and simple system design, to improve software quality [33].

2.4.8. Kanban

Like Lean, Kanban was created in Japan to optimize car manufacturing, though its roots lie in 40's, when Toyota Motor Company began investigating methods to improve process flow for manufacturing [1]. Software development began utilizing Kanban methods during early 2000's to minimize Work In Progress (WIP), visualise the workflow and improve cooperation between team members [34].

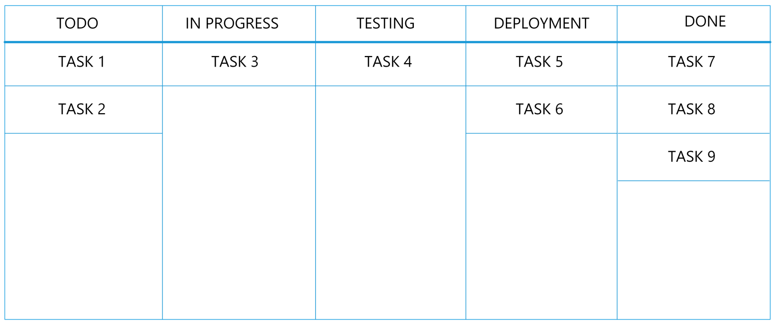
Most Kanban groups follow the foundational principles [3]:

- Start with what you do now.
- Agree to pursue incremental, evolutionary change.
- Respect current roles, responsibilities and titles.

Additionally, Kanban has six core practices [3]:

- Visualize the workflow - what are the steps from ongoing project to finished one?
- Limit WIP - avoid accumulating work.
- Manage flow - put emphasis on how well the different work cycles go together and how smoothly they operate.
- Make process policies explicit - try to make it so that the workers do right thing and they do it right.
- Implement feedback loops - continuously review your work and process.
- Improve collaboratively, evolve experimentally - by using metrics and models, your team can improve.

One distinct feature to Kanban is the Kanban board (Figure 7). Its function is to show what each developer is on doing, what is prioritized and what items are in focus [34]. Many non-Kanban organizations have begun to integrate the Kanban board to their work process; for example, Scrum with Kanban where the former gives tools for development and latter visualizes the workflow [35].



TODO	IN PROGRESS	TESTING	DEPLOYMENT	DONE
TASK 1	TASK 3	TASK 4	TASK 5	TASK 7
TASK 2			TASK 6	TASK 8
				TASK 9

Figure 7. A basic Kanban board.

2.4.9. Scaled Agile Framework (SAFe)

Even popular agile methods can sometimes struggle in larger scale organisations and doing agile between multiple, often global teams takes a lot of hard work and communication [36]. Organizations began on investigating scalable solutions to fit to modern needs [37]. While there are several methods that are scaling, Scaled Agile Framework

(SAFe) is the most used one [28]. Although it is meant for big organization use, there are some interesting attributes that could be utilized in a smaller team.

SAFe is relatively young method in field. Since it is designed with bigger organizations in mind, the expectation is that there are multiple agile teams, five to seven people, within the organization and they need to work cohesively [38]. Teams can use other methods like Scrum as a base - SAFe mostly works on higher management level and is compatible with other popular methods [39].

SAFe brings Lean Portfolio Management (LPM) as a part of the business solution. LPM combines traditional Project Portfolio Management (PPM) and Lean to bring agility to managing the projects [40].

SAFe has four core values that display how leaders should act [40]:

- Alignment - portfolio strategizing, business value planning and adjusting scope.
- Built-in quality - making an environment where quality is the norm.
- Transparency - visualisation of current work.
- Program execution - planning and execution at the perspective of business owner and removing anything that hinders the workflow.

Additionally, there are 10 principles that are considered mandatory [40]:

- Take an economic view.
- Apply systems thinking.
- Assume variability; preserve options.
- Build incrementally with fast, integrated learning cycles.
- Base milestones on objective evaluation of working systems.
- Visualize and limit WIP, reduce batch sizes, and manage queue lengths.
- Apply cadence, synchronize with cross-domain planning.
- Unlock the intrinsic motivation of knowledge workers.
- Decentralize decision-making.
- Organize around value.

Because multiple teams are working together, there needs to be a way to synchronize and collaborate between teams. Agile Release Train (ART) - depicted in Figure 8 - consists of teams (business, software, quality assurance, etc.) working together with stakeholders. Continuous exploration (CE), continuous integration (CI) and continuous deployment (CD) are the three carriages and release on demand (RoD) is the last stop [40]. The ART process takes somewhere between 60 to 120 days [38]. Depending on scale, there can be multiple ARTs going same time. Together with DevOps, ART is part of the continuous delivery pipeline (CDP) [40].

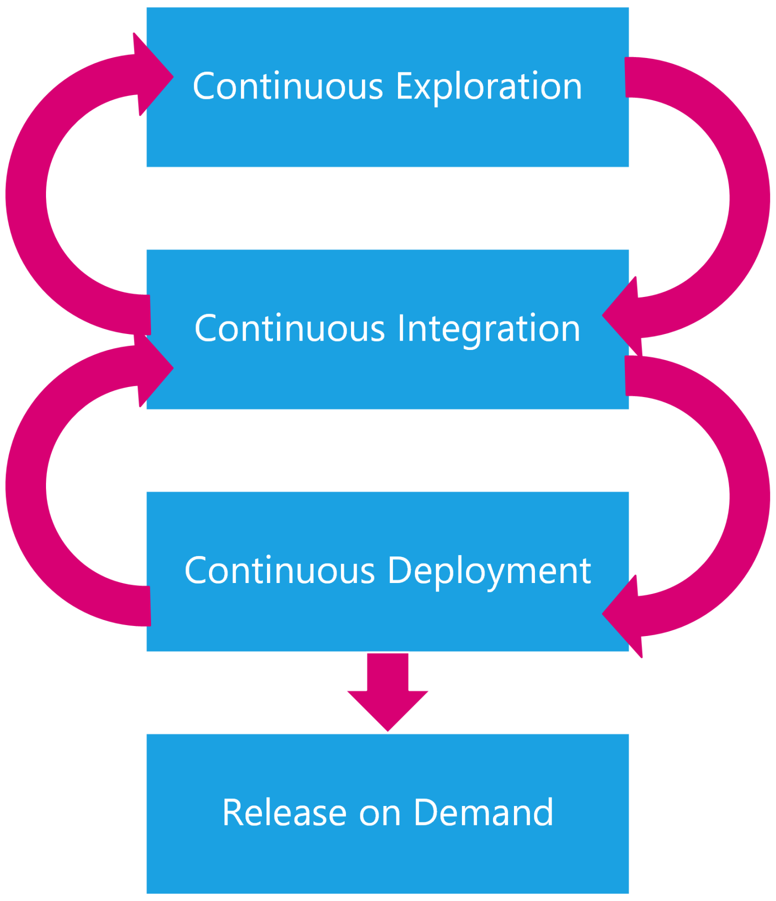


Figure 8. Agile Release Train (ART), a part of continuous delivery pipeline (CDP).

The official manual presents four different configurations, bringing some scalability to it. Essential SAFe is the bare bones structure, containing everything needed to work agile. Large Solution offers tools to manage multiple ARTs. Portfolio SAFe adds business agility as part of the framework. Finally, Full SAFe is everything in one big package [40].

2.4.10. Comparison

The Tables 1 and 2 show that some methods are more defined than others. Usually the more delimited methods focus on particular fields rather than the whole process; for example Lean concentrates on eliminating waste and empowering the team rather than defining a working development cycle.

Every method adapts to changing requirements by developing feature by feature. The main difference among them is how they prioritize items - some do feature prioritization planning, others visualise them and constantly re-evaluate, and so on.

Cycles commonly have at least short development phase and deployment. This is expected, as Agile Manifesto principles urge teams to deliver frequently. Some methods have a planning and feedback as part of cycle, while some methods do not define cycles. In the latter case, the structure of work phases is up to the context of the current project and team.

The measurement of the progress is very different across the methods. Some specify tools that are used (FDD's project completion formula, for example), others are more passive and just meter the done functionalities.

Some methods lean on improving by practices (XP and Scrum) and some on principles (Lean and SAFe). The former tends to be concrete on their implementation: phases are strictly determined, they are usually easy to implement and the focus is more on continuous delivery and software quality. The latter are more flexible and have focus on the work process. According to Annual State of Agile Report, the methods that are practice-focused are far more popular than those based on principles [28].

The Manifesto and its principles valued customer collaboration and every method has them involved to some degree. The least customer-dependant methods report current progress and gather feedback from them. On the other side of scale, with methods like XP, customer is constantly involved in development process.

Learning and self-reflection are important aspects of agile mindset. Most methods have reserved time for dedicated learning activities. A minor part common to all methods is embracing a mindset for improvement and have a constant feedback.

Each method has its feature that bring something unique to the table. As an example, Lean's list of the types of waste gives the team idea on what to avoid and give easier time to deliver on time.

2.5. Choosing and Adapting Agile Method

There are many factors that can influence on what is the most suitable method for given team. Company culture, customer collaboration, projects, processes, tools and team members all affect the decision. Organizations may even need to change their current approach on these dimensions to take the most of selected method [22].

Unfortunately, adapting a new method is not always painless. Possible obstacles can be organisational changes (changing mindsets, moving from documentation, preferring face-to-face communication), people-related changes (putting more trust to team members, promoting learning and collaboration), process-related challenges (less planning and more ad hoc type of working, adapting to iterative cycles) and technology and tools related challenges (preferring technologies and tools that support iterative development). To combat these, a good amount of time and training needs to be had [41].

Some adoption models have been created to help the organizations accustom to agile. Qumer et al. presented Agile Adoption and Improvement Model (AAIM) that consists of three blocks with their respective levels. The first block, called Agile-prompt, introduces the team to basics of agile. Its only level - agile infancy - aims to familiarize everyone with the core values of agile development without applying the method right away. The next block is Agile-crux and it focuses on agile practices. The three levels are agile initial (communication and collaboration), agile realization (constant delivery and minimal documentation) and agile value (people-oriented values). Agile-apex is the final block that concentrates on learning and quality product environment. In agile smart level, learning environments are build. Agile progress is the last level and it focuses on production environments and keeping the process agile [42].

Agile Adoption Framework looks at helping agile coaches to adopt agile to a single project by having index for estimating agile potential and a four-stage practices adoption process. The former evaluates the teams potential to use agile in their work. It is measured through four components: levels; principles; practices and concepts; and indicators. The different levels are collaborative, evolutionary, effective, adaptive and ambient and each has its own set of practices. Principles measure the teams ability to follow the important agile practices. Practices check if there is potential to have any practices that are commonly found in agile. Lastly, set indicators (a bunch of questions related to agile adoption) are used to estimate if team is ready to adopt agile practices and concepts. Once the measurement index (done by evaluating the set indicator results) has been done and evaluated, the four-stage process can begin. The four stages are: identifying discontinuing factors (determining what could prevent successfully adopting agile practices), project level assessment (estimating what is the target agile level), organizational readiness assessment (determining on what level the organization is ready to operate) and reconciliation (weights the differences found in project level and organizational readiness assessments) [43].

Table 1. Comparison of agile methods 1/2

Agile Methods					
Category	**ASD**	**DSDM**	**FDD**	**Lean**	**Crystal**
Adaptability	Adaptive cycles and management	Phases can move back and forth depending on changes	Building by feature and prioritizing them	See the whole and adjust according to situation	Levels of criticality, team adjusting according to personalities, working environment and current assignment.
Iteration cycles	Project initiation ⇒ cycle planning ⇒ concurrent component engineering ⇒quality engineering ⇒ final quality assurance ⇒ release	Specific to different projects. In optimal situation feasibility and business study ⇒ functional model iteration ⇒ design and build iteration ⇒ implementation	Overall model development ⇒ features list building ⇒ feature planning ⇒ feature design ⇒ feature building	Not defined, team dependent	Short iteration(s) ⇒Delivery
Progress measurement	Time-box and mission progress monitoring	Face-to-face communication, daily stand-ups, other progress measurement tools	Configuration management and reporting of results, calculating project completion	Lean thinking, core values	Focus on individual task and where project is heading
User/client interaction	At the beginning of project (requirements gathering), some learning practices	Encouraging collaboration between stakeholders and team members, agreeing on business objectives	Reporting of results	Used feedback to amplify learning	Easy access to expert users
Learning	Learning activities after adaptive cycle, team dependant	Feedback as part of iterative loop, keeping everyone up to date with current knowledge	Milestone completion measurement	Amplifying learning and empowering the team.	Reflective improvement and close communication
Unique features	Adaptive cycle	Project dependant cycles	Reporting of results	Seven types of waste	Crystal methods based on team size and criticality

Table 2. Comparison of agile methods 2/2

	Agile Methods			
Category	**Scrum**	**XP**	**Kanban**	**SAFe**
Adaptability	Sprint planning, prioritizing tasks	Planning game, refactoring	Visualizing the workflow, picking important tasks.	Agile Release Train, continuous delivery pipeline
Iteration cycles	Sprint planning ⇒ development ⇒ sprint review ⇒ retrospective	Iteration planning ⇒ development ⇒ deployement	Not defined	Continuous exploration ⇒ continuous integration ⇒ continuous deployment⇒ release on demand
Progress measurement	Done tasks, product backlog	TDD passing tests, done functionalities	Done tasks, Kanban board, metrics	Milestones, complete ARTs
User/client interaction	Stakeholders present during sprint review	On-site customer, feedback as often as possible	Not defined	Release on demand, customer centricity
Learning	Sprint retrospective	Refactoring, feedback loop, "embracing change"	Feedback loops	Team learning, shared vision
Unique features	Sprints, roles	Refactoring, pair programming, on-site customer	Kanban board	Agile Release Train, Lean Portfolio Management

3. ANALYSIS AND DESIGN

This chapter introduces the evaluation team, gathers and analyses data through surveys and interviews and makes modifications to model based on the results of analysis. The model is focused on improving on way developers work, though some changes benefit management.

3.1. The Team

The team selected as a case study for this thesis is a small, remote branch of a mid sized IT-organization, that has nine people in total - eight in development team and one IT-support person. The development team is further divided into development (three developers that includes Scrum Master, one tester and two software architects) and management (one project manager and one account manager). At their headquarters in Vantaa, the organization has invested in scaling agile method to manage their teams. Meanwhile, the remote branch utilizes Scrum and Kanban (Scrumban).

The projects that the remote branch do are fairly small ones; they last from two weeks to half year and projects usually have a maximum of four people. Depending on current projects and role, development team member usually is involved in more than one concurrent project. Depending on the size of the project, some amount of planning and story writing is done beforehand. However, some customers, usually ones that are long time partners, prefer more of an ad hoc way of adding new or fixing features. In case of these kinds of requests, the task is small enough that one person can handle it.

Scrum is used for sprint cycles and roles. While the team has task board, due to nature of some tasks, it is used more of a Kanban way - tasks can be added in a middle of a sprint. This is in contrast to Scrum sprints, which are more project- than team-focused. This is a conscious choice by team, since there is no other way to log tasks that do not contribute to sprint goals.

The following sections evaluate the currently used principles and practices. Details of the projects, irrelevant to the discussion, are left out of the scope of the analysis due to non-disclosure agreements.

3.2. Data Collection

The collection was done by following the GDPR guidelines [44]. All surveys were done anonymously and had same themes as the interview. The interviews were not recorded and specific questions (e.g "How motivated are you?") that could have been linked to particular survey result or individual-based interview notes were avoided. Participants were constantly reminded that attendance was voluntary and they could stop participating any time. Before starting the surveying and interviewing, a small presentation about the purpose of the thesis was held.

First, an anonymous survey was sent to evaluated team members through Microsoft forms. The form (found on Tables 3 and 8) contained statements regarding the agile principles and values found in [2] [22]. Statements 1-3 are about the general state of working environment, 4-15 are for individuals and interactions, 16-23 tackle on working software (over comprehensive documentation) and 24-29 are for customer collaboration. The survey taker had to mark the statements as fully agreeing, slightly agreeing, slightly disagreeing and fully disagreeing. The typical neutral statement ("I am not sure") was left out to force participant to take position.

After the survey was taken, a semi-structured interview for each participant was had. The purpose of the interview was to additional insight of the general questionnaire answers.

The same survey was sent to different development teams within the organization. However, due to partial disruption of the work habits caused by the COVID-19 (coronavirus disease originated at the end of 2019) pandemic, attendance at Vantaa office was low. As surveying other teams was intended for comparison purposes and main focus being on improving the model for remote team, it was decided to not involve other teams - though comparing the evaluated and Vantaa's model and improving headquarters teams is not out of the question in future.

Table 3. Remote branch teams answers from the survey.

Questionnaire - study team answers				
Statement	**Fully agree**	**Slightly agree**	**Slightly disagree**	**Fully disagree**
Current working environment feels agile (1)	1	5	0	0
Working environment utilizes agile practices (2)	4	2	0	0
I am satisfied with current working methods (3)	1	3	2	0
My team prefers individuals and interactions over processes and tools (4)	2	3	1	0
Communication between project team members is efficient (5)	2	3	1	0
In case of having a problem, I can express it some way and get a help to fix it (6)	5	1	0	0
Current project(s) have a lot of boring tasks (7)	0	2	3	1
I feel motivated (8)	4	2	0	0
I want to motivate other team members (9)	4	2	0	0
My team motivates its members through the environment and activities (10)	1	2	3	0
My team is self-organizing (11)	0	5	1	0
My team needs time to define the roles of a project (12)	1	1	4	0
I would want more face-to-face discussion between team members about the topics of the project (13)	0	2	4	0
Projects are completed in time (14)	4	2	0	0
My work is exhausting (15)	0	3	3	0
Projects can react to upcoming changes (16)	4	2	0	0
Projects need more documentation (17)	1	4	1	0
Projects can continuously deliver software (18)	1	3	2	0
Projects deliver working software (19)	2	4	0	0
The amount and quality of software is metered some way (20)	0	2	4	0
Current work cycle is practical (21)	2	2	2	0
I wish change to current work cycle (22)	0	2	2	2
Projects follow the principles of continuous integration (23)	2	3	1	0
Customers understand our working methods (24)	0	6	0	0
Customer is accompanied in our projects (25)	2	4	0	0
The customer takes part on deciding on the important issues of the project (26)	3	3	0	0
Communication with the customer is inefficient (27)	0	1	4	1
Collaboration with the customer is continuous (28)	3	3	0	0
I would like to have more meetings between the team members and customer (29)	1	0	5	0

3.2.1. *Survey*

Everyone who participated at interviews also answered the survey. Overall, with the exception of some statements, there were many similarities between the interview answers and anonymous survey results.

Everyone agreed that the working environment is agile, with majority of participants slightly agreeing (83%). Likewise, everyone agreed agile practices are utilized right now; four agreed fully (67%) and two somewhat agreed (33%). This points that the team thinks they are doing agile, but some parts do not fully feel like it. This is further confirmed by some people being slightly unsatisfied with currently used methods (33%). The dissatisfaction with some things might have come from those workers who are more experienced in agile development - most of participants are still relatively new to agile so they are likely less to find problems in the current model.

Communication among project members seems to be at least at a satisfying level. Only one person (17%) slightly disagreed. Majority fully agreed that they get help when they asked (83%). No one could pinpoint any clear weaknesses in communication during the interviews and while some hoped that it could be at least somewhat improved, many thought that it is not the most prioritizing aspect to improve on.

The motivation level seems high in the team. Four person (67%) fully agreed that they are motivated and want to motivate others. However, exactly half (50%) slightly disagreed on team motivating with its environment and activities. Half of the participants also sightly agreed (50%) that the current work is exhausting and the other half (50%) disagreed.

Many team members believed that their team is partially self-organizing, with five (83%) slightly agreeing and one (17%) slightly disagreeing. Interestingly, 33% agreed either fully or slightly at statement about defining roles - the rest slightly disagreed (67%). During the interview, when asked whether the team is self-organizing, the answers were mixed. Some thought that team being self-organized was one of the strengths, and some thought that people need to be more independent and daring when working on project.

All agreed to some extent that projects team can adapt to changes. Everyone agreed that projects complete on time almost always. Team members were indifferent on whether their project team does continuous delivery - 33% slightly disagreed, 50% slightly agreed and 17% fully agreed. All thought that projects deliver working software. Many mentioned the team ability to estimate the time to get a task done as the biggest reason why project timelines extend. CI/CD tools and robot tests were mentioned as why the person thinks that the team produces quality software, although many wanted testing to start a bit earlier during the project.

Majority (67%) slightly agreed that more documentation would be needed. When asked what kind of documentation they want, almost everyone meant documentation supporting project teams - how to setup a development environment, test cases, etc.

In the same way, 67% believed that software quality metering is inefficient as now. One of the participants claimed "...we have technologies such as SonarQube at hand, but they are neither used nor utilised properly", while other wanted more visible statistics.

Most divisive topics were the statements about work cycles. Those (33%) that thought that the current work cycle is functional were probably the same that do not want changes to it. Slightly agree and slightly disagree got two (33%) votes each. The controversial subject regarding this was the Scrum sprint cycle. Wildly different answers were given on how effective it is, were the artefacts left from the scrum rituals taken used in a meaningful way and if Scrum with Kanban is the correct model. Some liked the structure and wanted maybe a slight modification to suit a little better the needs, while other thought that work cycle needs big improvement and current one slows development and generates too much waste. Overall, majority still wanted to use the Scrum sprint cycle framework but also to refine it little.

Customer collaboration was seen as a strength, and the results from the survey reflect that. A couple participants described the customer collaboration mostly non-bureaucratic and as projects proceed, the client becomes more understanding on the way that the work is done.

According to interviews, the most controversial topics were quality measurement, work cycle, (human) resource allocation, learning/training possibilities and work outside of the sprint scope (ad hoc). Work environment and communication also got some amount of discussion, though most were satisfied with current situation.

Table 4. Rough estimation of agile level based on Agile Adoption Framework

Agile Adoptation					
Level	Embrace change to Deliver Customer Value	Plan and Deliver Software Frequently	Human Centric	Technical Excellence	Customer Collaboration
5					**X**
4	**X**				
3		**X**	**X**		
2				**X**	
1					

Generally, the team seems to find the general state of working environment tolerable, but still in its early stage. Attitude towards team members seems to be good and team clearly is motivated. Team is very indifferent about the state of working software and continuous integration but majority agrees that improvement is in need. The team thinks that customer collaboration is at satisfying level, and is one of the strengths. Table 4 has a rough estimate of team's agile adoption done by the author [43].

In a nutshell, the upgraded model should focus on the following things:

1. Keep the current work cycle roughly the same but streamline it.
2. Have a way to measure progress.
3. Make learning process during cycles more natural.
4. Options to monitor and manage ad hoc tasks.

3.2.2. Principles and Practices

After a discussion with some of the members, it was decided that a complete revamp of current model is unnecessary. Everyone at this point were accustomed to Scrum's work cycle and adapting to completely new one would take time. Rather, applying practices from the underutilized Kanban was considered.

Recalling the principles of Kanban, they could help in the following way:

- Visualising the workflow - monitoring and managing ad hoc, measuring progress.
- Limit WIP - less overburden, streamline work cycle.
- Manage flow - streamline work cycle.

- Improve collaboratively, evolve experimentally - help learning process.

Additionally, some practices from XP, such as test-first coding and continuous integration, could be added to improve software quality [33].

3.3. Improving Model

Updating a method is not as easy as telling everyone "starting now, we are going to follow new practices." Most practices are dependent on mindset and trying to change human behaviour is hard and time consuming. Luckily, different tools are offered by various organizations.

Jira, a task management software Atlassian, and Microsoft Azure were in heavy use. Because of that, different plug-ins and tools to them were searched first.

3.3.1. Visualizing and Managing Workflow

One element that was found lacking was the use of visualization tools. Project team members had to share their information either through daily stand ups or Slack messages. While somewhat working, the amount of information on both makes it hard to focus on relevant matters.

Tasks were managed through Jira board. The board itself is a useful tool to prioritize and assign tasks. The downside is that having multiple projects clog up the board fast and assessing each others workload is sometimes hard.

To improve on workflow visualization, Jira board was divided into per persons instead of ongoing and undone tasks split in order to help team members to see what tasks are assigned to whom and which are not currently assigned. This change also aims to help team to better understand project progression and what are being developed on as well as lessen time to get what other team member is doing. Moreover, an additional column "In testing" was added to separate tasks between "still in development" and "ready, but needs validation".

A pre commit hook script was added to projects that had Git. When a user makes a commit, if the commit message contains Jira task name at the start, the commit message is forwarded to task's comment section (see Appendix 2). This will help the developer to update the task without manual work or extra tools, as well as helps other team members to know the current situation.

3.3.2. Limiting WIP

Previous studies have noted that limiting WIP is a challenging task and, if done poorly, can result into even more waste [34]. Regardless, a proper implementation would help focusing on important tasks.

For the first time in its history, team had to work under WIP limits - each member had a limit of 3 ongoing tasks. This was monitored on Jira's own WIP limit setting. The aim was to force developer to focus on one task at a time and maintain a smoother

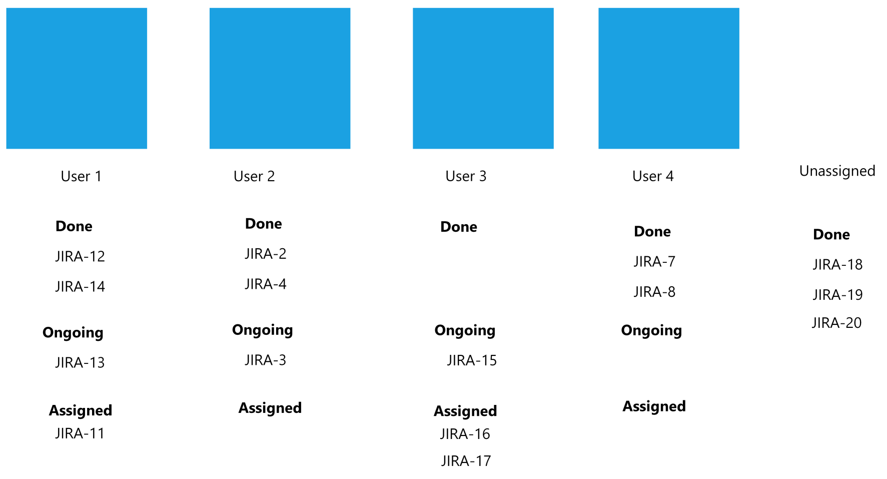
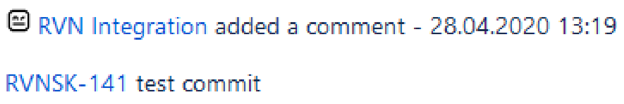


Figure 9. Simplified illustration of Jira board sorted by person.



RVN Integration added a comment - 28.04.2020 13:19

RVNSK-141 test commit

Figure 10. Example of automated message generated pre commit hook script.

workflow. On top of that, a simple Jira script was created to warn user about having too much items on "In Progress" (see Appendix 1).

3.3.3. Self-improvement and Experimentation

Amplifying learning is also an assignment that is not quite easy. The willingness to learn new skills and courage to take task that are out of comfort zone comes from person itself, not from any process or principle. Agile methods try to motivate people to learn with different activities but cannot generally force them to learn.

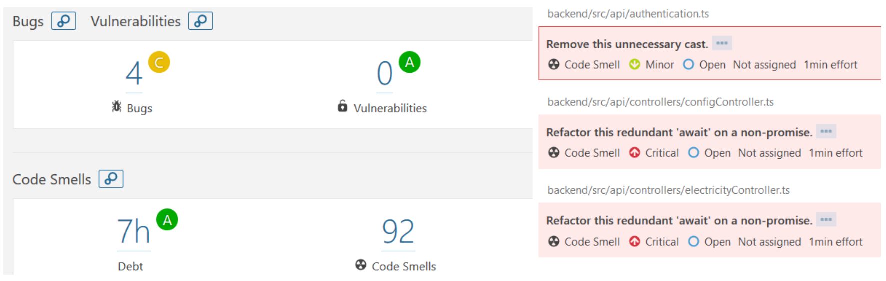
The team was instructed to increase courage to learn by assigning tasks themselves and more experienced ones on the subject to mentor the task takers. This is not the most concrete solution, as it needs person to be willing to take task themselves and no other tools or plugins are used. On the other hand, it also serves as a way to improve team's willingness to learn.

3.3.4. Assuring Quality

In agile development, software quality assurance can be done in either static or dynamic way; the former comes in forms of no code execution such as code review and the latter covers things like test runs. Agile methods favour dynamic approaches as they are efficient in terms of time and can be used at the very start of development [45]. Acceptance testing was practised in previous project so it was familiar to team.

While some members had knowledge on CI/CD tools, these were less familiar so more focus was put on them.

Various tools and practices for CI are found for different purposes and needs. After exploring the options, the decision came down to how much said tool would cost and how long does the setup take. SonarQube - an open-source continuous code inspection platform [46] - was chosen because some members had previously experimented with it, it being free to use, good support on programming languages and how easy to setup it was. Example on SonarQube reporting bugs and vulnerabilities can be seen in Figure 11.



backend/src/api/authentication.ts
Remove this unnecessary cast.
Code Smell Minor Open Not assigned 1min effort

backend/src/api/controllers/configController.ts
Refactor this redundant 'await' on a non-promise.
Code Smell Critical Open Not assigned 1min effort

backend/src/api/controllers/electricityController.ts
Refactor this redundant 'await' on a non-promise.
Code Smell Critical Open Not assigned 1min effort

Figure 11. Example on code coverage and issues found by SonarQube. Here, the tool has found four bugs from the source code.

Some projects had automated robot tests implemented and test results from them are now used as part of code coverage. Also, pipeline structure was slightly modified (Figure 12) in one project: regression pipeline that runs once a day was added to give developers insight on current production build.

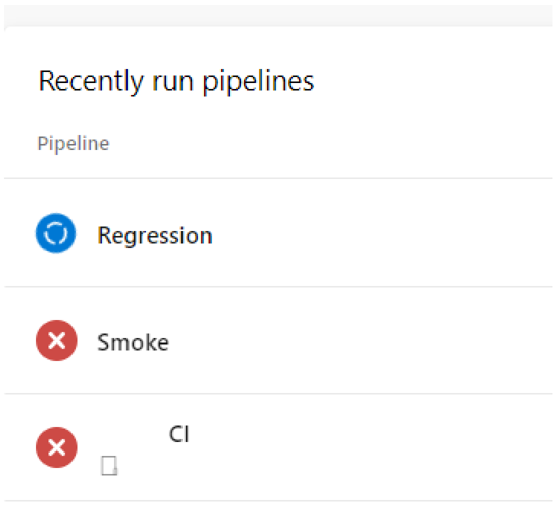


Figure 12. Project pipeline structure: CI runs the "Continuous Integration" tagged tests, Smoke (tests that inspect the stability of current build) runs after CI is complete and Regression is ran every night.

3.3.5. Proposed Model

The new model adds more Kanban principles and practices to already established Scrumban. In addition, some XP software quality practices are adopted. In the end, the following was added:

- Improve the visualization and readability of task management tool (Jira).
- Introduce WIP limits.
- Encourage team members to more autonomic and take tasks from backlog, even though they have less experience on the subject.
- Use tools to meter software quality.

4. EVALUATION

Evaluation of the proposed model was done between the sprints in April. Everyone on development team participated in the evaluation. The first sprint was reserved for learning and familiarizing new practices and principles and if needed, develop the method further. In second sprint, the focus was more on collecting data. At the very end, final survey regarding the new features was sent to team members.

Setting up the code coverage and quality tools took some extra time. Because of that, tools were absent during the first sprint.

Jira's generated charts were used to track on how issues move and time spent on them. Figure 13 shows some statistics of sprint previous to improved model - how much tasks (or issues, as Jira calls them) has been moved from ToDo to Done, which days had more tasks than others and average, median, minimum and maximum time that task took. These results will be used as a reference for the comparison with the results after the introduction of the improvements.

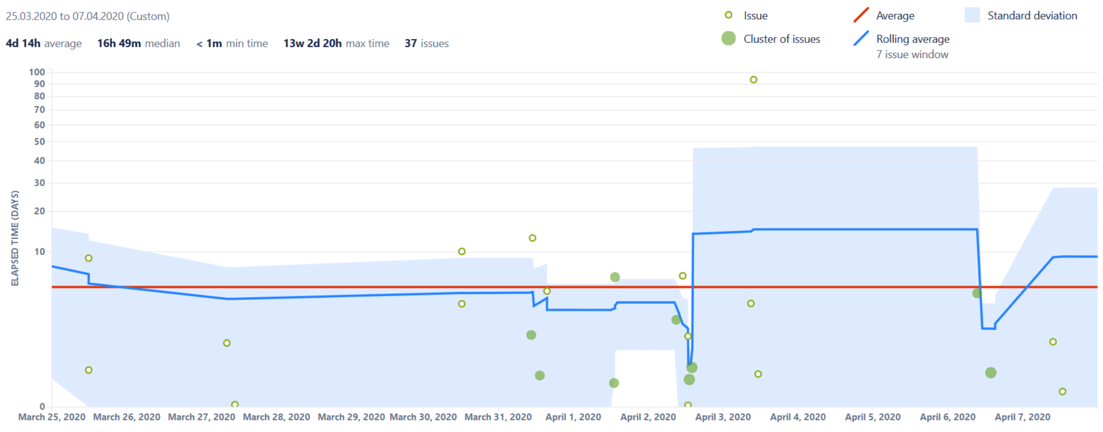


Figure 13. Jira control chart from sprint before the evaluation.

Because of COVID-19 pandemic, some of the previous plans had to be readjusted. Anything requiring being at the office or needing face-to-face communication had to be either reimagined to fit to remote work or cut entirely. Luckily, most of planned features were eligible for remote, even though it was not the modus operandi before the evaluation.

The global pandemic also affected the number of ongoing projects. The smaller projects were scarcer than normally, but bigger projects were not affected.

Regarding to code coverage tools, many projects on the evaluated spring had a structure that made utilizing tools like SonarQube hard. In these cases, code quality evaluation was less important than other implementations.

Data were mostly gathered from Jira provided charts and code coverage tools. Conversation about the implementation among the team members was had through the two sprints in Slack. Interview and surveys were had at the end of each sprint.

4.1. Results

The team tested the proposed model for four weeks. During the end of first sprint, a discussion about the model was held. The second discussion was couple of days before the second sprint ending. Both discussion had similar structure; changes to workflow were praised and whether WIP-limitation is seen as necessary were the most discussed topics.

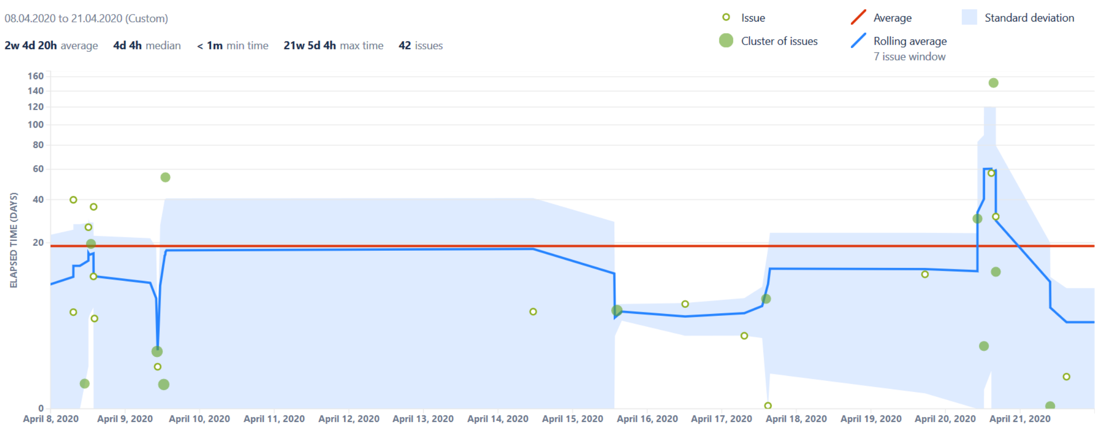


Figure 14. Jira control chart from first evaluation sprint.

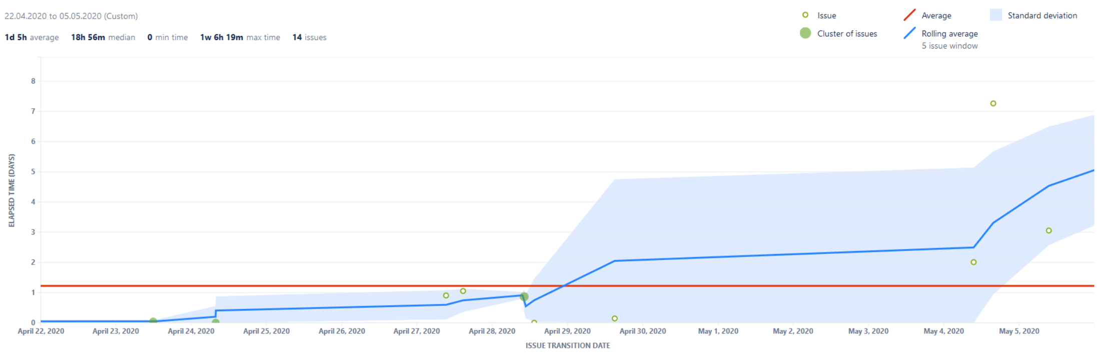


Figure 15. Jira control chart from second evaluation sprint.

Unfortunately, the statistics from control charts, Figures 14 and 15, do not give any good data about the efficiency of the model. During the first evaluation, Scrum Master did some backlog cleaning - removing old tasks that had not moved for a long time - and it wildly fluctuated the control chart as seen on between 19th and 20th of April. The second sprint had even less tasks than the first one - less than half compared to pre-evaluation sprint. Nevertheless, one field that could be used to make some sort of conclusion is median time. Comparing the pre-evaluation and second sprint median, the time to issue shift from ToDo to Resolved has not changed significantly.

As Figure 16 shows, issues were quickly resolved and team was not overwhelmed by new issues. Because some unfinished issues were taken from previous sprint, the amount of resolved issues exceeds created ones. During the second interview that was

Table 5. Statistics from Jira control charts

Jira Control Chart Statistics			
Name	Pre-evaluation sprint	Evaluation sprint 1	Evaluation sprint 2
Average time per issue	4d 14h	2w 4d 20h	1d 5h
Median time per issue	16h 49min	4d 4h	18h 56min
Min time	<1min	<1min	0
Max time	13w 2d 20h	21w 5d 4h	1w 6h 19min
Issues	37	42	14

held at the end of the evaluation period, many developers thought they used the board more actively than before, and judging from the figure and statistics from Table 5, team responds to and solves issues fast. The oddities on chart are explained national holidays between end of April and May and a new project starting at the start of May.

Table 6. Thoughts on tested features

Evaluation Survey Results				
Name	I think the feature was good	I think the feature needs some refining	Can not say	I think the feature was not good
Changes to Jira board	4	0	0	0
WIP limits	1	2	0	1
Taking unassigned tasks, mentoring	1	2	1	0
Code quality and coverage tools	0	3	1	0

A small survey and general feedback meeting was made on how the participants felt about the features. As seen on Table 6, the changes to board (personal swim lanes and column changes) was the ultimate winner. Everyone was very positive about the changes, including those that were sceptical first. Feedback included mentions of easier to read than previous board, improvement of workflow, increased usage of board and easier estimation of resources. No one found anything negative about it and team decided to keep the new board structure after the evaluation.

Big part of participants felt little discontent about the WIP limits. No one really was affected by it during the sprint and it did not affect anyone - the feature was invisible throughout the evaluation. Participants were eager to see it in action, and the amount of work during the sprint did not allow it. Regardless, most still felt that the WIP limit idea was good; many pointed in feedback meeting that it forces person to focus one task at a time. One mentioned that maybe instead of being limit per person, it should be limit per project, as in traditional Kanban. Team wanted to keep WIP limits for further evaluation and stress testing but it will likely be adjusted a little.

The alterations to learning were not as helpful as wished. One subject that was brought up several times during the interviews was the threshold of taking an unknown task. This suggests that further research on how to attract team members to try dif-

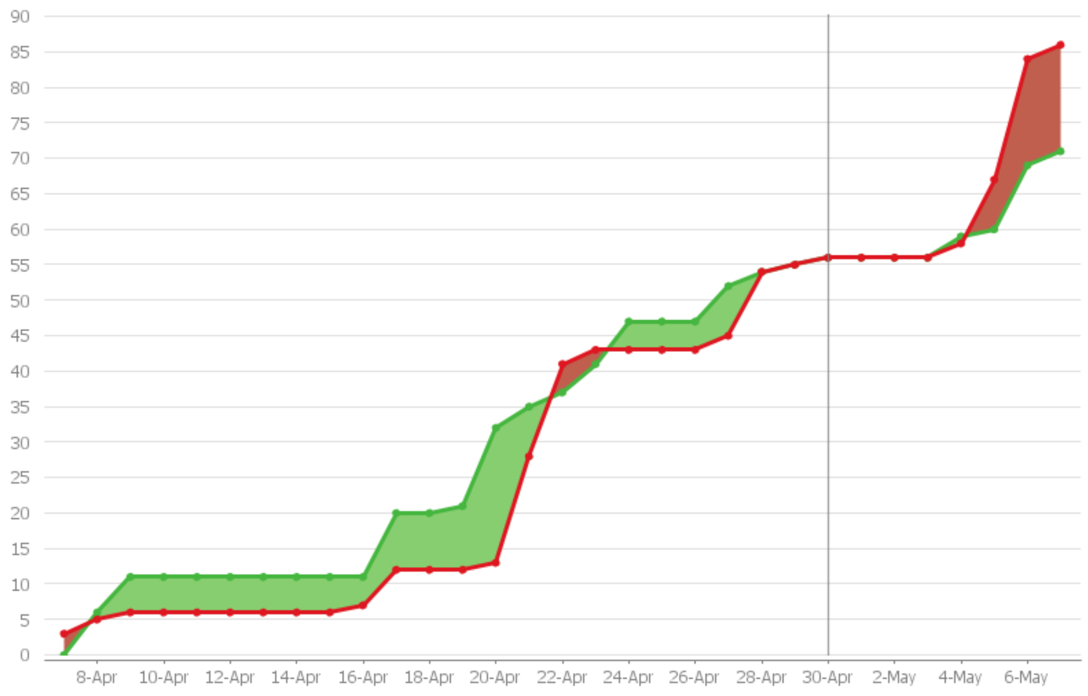


Figure 16. Number of created (red) versus resolved (green) issues.

ferent tasks is in need. Some ideas that participants brought were hyperlinks to documentation and other material to tasks description field, more detailed user stories and mentions of who might help in case of having a problem. Some in-house tasks, that were more descriptive in details, were added to backlog during the second sprint and it seemed to help developers picking a task themselves.

Code coverage and quality tools were under not utilized due to difficulty to implement them on current projects. Still, majority think they are a good idea.

Table 7. General feelings on features

Overall impressions on the new features	
Feature	**Impression**
Personal swimlanes and column changes	Great change, keep it this way.
WIP-limit	Invisible through evaluation. good idea but needs further evaluation.
Taking tasks from backlog and mentoring	Too feew tasks to experiment. Threshold of taking unknown task needs to be lowered
Code coverage and quality tools	Difficuilt to implement on some projects. Still in under progress.

As the Table 7 might suggest, the evaluation left positive mark to team, even though the features, with the exception of changes to the board, were not so successful. Still, it opened for more discussion about current model and potential future work.

Overall, the participants felt that the testing environment felt more agile than the default one. Almost everyone mentioned the improved workflow as a reason why. The first field - "Current working environment feels agile" - in the second survey seems to coincide with the statement, with half fully agreeing and other half slightly agreeing. Same can be said about the second statement.

Table 8. Remote branch teams answers from the second survey

Questionnaire - study team answers				
Statement	**Fully agree**	**Slightly agree**	**Slightly disagree**	**Fully disagree**
Current working environment feels agile (1)	3	3	0	0
Working environment utilizes agile practices (2)	3	3	0	0
I am satisfied with current working methods (3)	2	3	1	0
My team prefers inviduals and interactions over processes and tools (4)	4	2	0	0
Communication between project team members is efficient (5)	1	5	0	0
In case of having a problem, I can express it some way and get a help to fix it (6)	6	0	0	0
Current project(s) have a lot of boring tasks (7)	0	2	4	0
I feel motivated (8)	2	4	0	0
I want to motivate other team members (9)	4	2	0	0
My team motivates its members throughthe environment and activies (10)	1	4	1	0
My team is self-organizing (11)	0	4	2	0
My team needs time to define the roles of a project (12)	0	0	4	2
I would want more face-to-face discussion between team members about the topics of the project (13)	1	3	2	0
Projects are completed in time (14)	2	3	1	0
My work is exhausting (15)	0	3	3	0
Projects can react to upcoming changes (16)	3	3	0	0
Projects need more documentation (17)	0	4	2	0
Projects can continuously deliver software (18)	1	3	2	0
Projects deliver working software (19)	3	1	2	0
The amount and quality of software is metered some way (20)	1	4	0	1
Current work cycle is practical (21)	5	1	0	0
I wish change to current work cycle (22)	0	0	3	3
Projects follow the principles of continuous integration (23)	2	3	1	0
Customers understand our working methods (24)	1	2	3	0
Customer is accompanied in our projects (25)	0	6	0	0
The customer takes part on deciding on the important issues of the project (26)	6	0	0	0
Communication with the customer is inefficient (27)	0	3	3	0
Collaboration with the customer is continuous (28)	1	5	0	0
I would like to have more meetings between the team members and customer (29)	1	0	5	0

Regardless, some people are not still fully satisfied with the model. Based on data found in Table 8, two (33%) thought that the working methods are good right now, three (50%) think some improvement is still needed to be made and one (17%) thinks that more revision to the model is still needed.

Big part of the interviewed participants claimed that remote work did not affect the way they work that much. That said, some mentioned that communication seemed to be slightly disjointed compared to office environment. Everyone admitted that time will probably fix this issue as team gets more conformable to new working environment. Second survey does also not seem to support the problem in communication, with majority (83%) slightly agreeing that communication between team members is efficient and everyone fully agreeing than when asking for help, they will get it.

During the interview, some participants wondered if the team is fully functioning without vast guidance. In survey, two members (33%) slightly disagreed that the team is self-organizing. One person mentioned that the team is still very young and inexperienced and, while changes to model might help members to be more independent, more experience in field will probably adjust it.

The desire to have face-to-face discussions had increased since the first survey, with one fully agreeing (17%) and three (50%) slightly agreeing. This might be product of current remote working environment, rather than outcome of evaluated model. Still, the importance of face-to-face communication cannot be underestimated - after all, it is one the twelve principles of Agile Manifesto [2]. Some alternatives to face-to-face communication should be thought of, in case of meetings through internet do not turn out as effective.

Project documentation is still fairly incomplete in most projects, according to the interviews. Four people (67%) slightly agreed that more documentation is still in need.

While the evaluated model did not take stance on documentation, the changes to better visualisation aimed to display information better. While everyone agreed that getting pieces of information on the Jira board is easier now, this generally does not help at troubleshooting and setup phases. Search for easy to use documentation tools is clearly still in need.

Even though the implementation of the code coverage and quality tools were not as successful as wanted, it seemed to leave good impression to most team members. Only one (17%) thought that metering is still not done at all. The introduction of tools seemed spring an effect, where developers understood the potential of them and began to further refine them. During the evaluation period, some meetings about the testing methods were had, implying that the experimentation left some sort of impression.

A couple of participants mentioned that communication between client and team has been slow. While worrying, the results can be explained as people shifting to remote work and it still being in early stages.

Table 9. Remote branch teams answers from both surveys

Survey comparison								
Statement	**Fully agree**		**Sligthly agree**		**Sligthly disagree**		**Fully disagree**	
	First	**Second**	**First**	**Second**	**First**	**Second**	**First**	**Second**
Current working environment feels agile (1)	1	3	5	3	0	0	0	0
Working environment utilizes agile practices (2)	4	3	2	3	0	0	0	0
I am satisfied with current working methods (3)	1	2	3	3	2	1	0	0
My team prefers inviduals and interactions over processes and tools (4)	2	4	3	2	1	0	0	0
Communication between project team members is efficient (5)	2	1	3	5	1	0	0	0
In case of having a problem, I can express it some way and get a help to fix it (6)	5	6	1	0	0	0	0	0
Current project(s) have a lot of boring tasks (7)	0	0	2	2	3	4	1	0
I feel motivated (8)	4	2	2	4	0	0	0	0
I want to motivate other team members (9)	4	4	2	2	0	0	0	0
My team motivates its members with the environment and activies (10)	1	1	2	4	3	1	0	0
My team is self-organizing (11)	0	0	5	4	1	2	0	0
My team needs time to define the roles of a project (12)	1	0	1	0	4	4	0	2
I would want more face-to-face discussion between team members about the topics of the project (13)	0	1	2	3	4	2	0	0
Projects are completed in time (14)	4	2	2	3	0	1	0	0
My work is exhausting (15)	0	0	3	3	3	3	0	0
Projects can react to upcoming changes (16)	4	3	2	3	0	0	0	0
Projects need more documentation (17)	1	0	4	4	1	2	0	0
Projects can continuously deliver software (18)	1	1	3	3	2	2	0	0
Projects deliver working software (19)	2	3	4	1	0	2	0	0
The amount and quality of software is metered some way (20)	0	1	2	4	4	0	0	1
Current work cycle is practical (21)	2	5	2	1	2	0	0	0
I wish change to current work cycle (22)	0	0	2	0	2	3	2	3
Projects follow the principles of continuous integration (23)	2	2	3	3	1	1	0	0
Customers understand our working methods (24)	0	1	6	2	0	3	0	0
Customer is accompanied in our projects (25)	2	0	4	6	0	0	0	0
The customer takes part on deciding on the important issues of the project (26)	3	6	3	0	0	0	0	0
Communication between the customer is inefficient (27)	0	0	1	3	4	3	1	0
Collaboration between the customer is continuous (28)	3	1	3	5	0	0	0	0
I would like to have more meetings between the team members and customer (29)	1	1	0	0	5	5	0	0

It can be summarised that team members think the proposed method feels more agile than previous one; first statement had three people fully agreeing that the environment feels agile on the second survey as opposed to first survey only having one. One person was less confident that agile practices are utilized but also one person more satisfied with current working methods. Comparison between the results of two surveys is gathered at Table 9.

Even though some mentioned having communication problems in second interview, it does not seem to reflect on survey results. In fact, statement number four about team preferring individuals and interactions over processes and tools had two more fully agreeing than previously.

Motivation seems to be lower than previously. This is likely due to external factors. Moving from office to remote and overall quiet sprint probably affected to the outcome more than the new model.

The new model did not seem to have any effect on self-organisation. While encouraging members to take tasks from the backlog independently tried to increase it, as stated before, the implementation itself and the amount of tasks were lacking. Having said that, at the second survey, statement 12 about roles had no one fully or slightly agreeing, whereas the first had one vote on both.

Interestingly, the team seems to be less disagreeing about the projects being completed on time. First survey has four people agreeing, while second only has two, and one person slightly disagree on the second. Neither the interviews or projects during the evaluated period seem give a clear reason explaining the drop. When comparing the statements about continuous delivery (16 to 19), the amount of people slightly disagreeing than previously. More time is needed to determine whether this is caused by the new model, the sudden shift from office to remote or even combination of both.

Even though the code coverage and quality tools were active only the latter half of evaluation, more people slightly agreed than disagreed about the statement of software quality measurements. This is not so surprising, as the statement did not mention anything about "how many measurements" are made, just if there is any. In hindsight, a statement should have been rephrased to "The amount and quality of software is metered enough" to give data whether the participants think if the tools and statics that are used now are enough.

The most surprising variation between the two surveys is number 21 - the statement about practical work cycle. The evaluated model's sprint cycle did not differ from previous one - so what caused the sudden just from two to five people fully agreeing? One theory is that better workflow correlates to better work cycle. In optimal situation, better workflow means more work is done and there is less air between the sprint rituals. Statement 22 about wishing the change to work cycle seems to agree with this, as participants did not agree with the statement any more.

The team is absolutely confident that the customer takes part on decision making. However, team thinks that the communication is not as continuous as before. The absence of face-to-face meeting probably have affected the statement slightly.

Table 10. Rough estimation of agile level at then end of evaluation period

Agile Adoption					
Level	Embrace change to Deliver Customer Value	Plan and Deliver Software Frequently	Human Centric	Technical Excellence	Customer Collaboration
5					**X**
4	**X**		**X**		
3		**X**		**X**	
2					
1					

Comparing to concepts and practices found in [43], the team has been getting better on human centric and technical excellence field. Planning and delivering software frequently and technical excellence still would need some work to be able to be called adaptive team. Table 10 shows new estimation of team's agile adoption level, evaluated by the author [43].

5. DISCUSSION

Researching, implementing and evaluating a new model is quite demanding job. It is important to remember that best practices, principles and tools for each agile team differentiate - the size of a team, the type of software they develop, the structure of organization and many more variables dictate what can be applied to which team. Also, the time to teach and get used to them can take quite a while and forcing them to a working environment does not guarantee efficient results.

Luckily, the target team was respondent to new changes. The new features were adopted very fast, and very little training had to be done. Because the features did not change the way how work is done and most newly introduced tools were automated (Section 3.3, Appendices 1 and 2), the team member had to do only small changes to their workflow to adapt to them. If, for example, work cycle would have been changed, it probably would have taken more time to accustom.

The evaluation period was quite problematic. First, the COVID-19 pandemic forced target team members from office to remote work. Second, the smaller projects were scarce, and getting any kind of code coverage or quality data from them was hard.

Even though there was a deviation from the original plan due to unforeseen events, the plan was successfully implemented to a different working environment. Still, more future research is needed. The sample size for projects was small, the target environment changed during the study and code quality and coverage data was hard to get. Despite the unfortunate evaluation period, some conclusions can be made.

The team felt like it became more agile. Despite the changes being rather minimal, they seemed to have a positive effect on team. Even the features that did not work as well as intended, seemed to spring up new discussions on current model. While the Jira Control Chart data was not as optimal as hoped, Figure 16 showed that issues are resolved rapidly. Finally, some further improvement on quality assurance was started at then end of the evaluation period.

While the team described themselves as a Scrumban team, the truth was that the old model was heavily Scrum with very little Kanban elements. Switching to more Kanban focused model helped to take care of tasks that were not originally in sprint focus, as well as made the team caring about workflow management.

During the evaluation period, one thing that was brought up was how the multi-project environment could be factored in new model. While it is true that almost everything in evaluation can be applied to an average agile team, the multi-project environment creates some obstacles that might not be problematic in others. One concern is the resource allocation: what task is everyone doing, how big is their backlog, etc. This was mitigated by better visualisation; board was divided by personal swim lanes, which made it easier to see everyone's backlog as well as current work. The other matter to think about was how to make a person focus on one project at a time. Although the WIP-limits do not fully ensure that the developer works on one project before shifting attention to another, they make sure that developers at least have to think what items need to be done first.

5.1. Principles

Principles are the foundation of agile software development. They guide teams towards better quality and give justification why some practices are held.

If one was given a task to make team more agile, the following "becoming-more-agile" principles might help:

- *Find what team is good at and what is lacking.* It could be tempting to focus on parts that the team is already doing well. Knowing the weaknesses of the team helps to narrow what practices and principle could make team better in agile.
- *Do not be afraid of experimenting.* Even if the new model is deemed a failure, it might motivate people to further revamp the model or find new ways to improve on.
- *Think what parts you could automate.* The Jira WIP-limit and Git pre-commit scripts were quite easy to implement and are very scalable. The less the model affects on the developer work process, usually the better. Be careful not to spend too much time on automating processes that take seconds to be resolved manually, though.
- *Reflect on your changes.* What worked and what did not? Did the new features add new value to work environment? Just like the team should reflect on their current progress on projects, the current model could also be thought on.

5.2. Future Work

As the evaluation working environment differed from previous one, some further validation whether agile model is as applicable not only to remote working but also to office environment is needed. As the previously used tools adopted by target team did not change when shifting environments, the possible change would probably be very minimal. However, there is possibility that different concerns may rise up in face-to-face than in remote situations.

The amount of projects during the two test sprints was low. Because of this, sprint backlog was smaller than usual and managing WIP-limits was easier. More testing in more hectic sprints is needed to determine how effective the WIP limitations are.

More metering would give better info on how the team has evolved over. Data found on Jira's own charts was not adequate enough to warrant any bigger conclusions. While some Jira statistics plug-ins exist, they are fairly pricey. Some tinkering with Jira scripting might give better data with small effort.

Some further investigation to other multi-project environments may point out other problems found out in these circumstances. On the other hand, agile models are flexible and as long as the team knows their strengths and weaknesses, adapting the current model should not be that big of a headache.

Code quality and coverage is a significant part of modern software development nowadays; this study only scratched the surface. More analysis on the effects on them in multi-project environment is needed. One point to consider is what principles and tools would work best in this kind of setting.

One feature that was bypassed was the use of burn-down charts. The idea of trying to use them to track issues came at the very end second evaluation sprint. The chart will also help on the aforementioned metering problem so it is very certain that it will be tried at some point on the future.

This thesis focused on improving the development environment. Further evaluation on project management and (in case of Scrum teams) Scrum Masters viewpoint could further improve the model.

At the second interview, project manager wished that maybe the model could be further developed, this time for helping the management. For example, resource estimation still needs a bit guessing. In possible further development, the model would be re-evaluated at scope management and possible solutions to enhance their work would be made and tested.

6. CONCLUSION

This thesis explored agile models that could be applicable to multi-project environment. The motivation was to improve current methods as well as to find out what values, principles and practices are peculiar in this type of environment. The test team was a small remote branch that had many small projects and couple of bigger ones going on. Rather than completely choosing a new model, improvements to the current one were designed. The proposed new model was Scrum with heavy implementations from Kanban and with some code quality principles found in XP.

The strengths and weaknesses of the team were gathered through surveys and interviews. The weaknesses then were further analysed and principles and practices from other methods were sought to help improve on weaknesses.

The evaluation period was quite problematic. Unfortunately, the COVID-19 pandemic caused everyone to move from office to remote work; the target environment changed just in the middle of thesis. This also affected how team members communicated, what visualisation tools could be used and the amounts of projects going on. The two sprints during evaluation were slower than usual and the amount of tasks in backlog was smaller than previous sprints. Despite that, the evaluation period can be considered successful, and very little changes had to be done in order to fit the model to remote work.

The new features that were introduced were not all as effective as it was initially thought. WIP-limits were basically invisible throughout the evaluation period, encouraging members to take unassigned tasked was hard because of small backlog on threshold being high and big part of projects had architecture that made the use of code coverage and quality tools hard. However, the changes to visualization were a great success, so much that the team wanted to keep the personal swim lanes and In Testing column without any further changes.

It is recommended to experiment with new principles and practices. Even in case of failure, the results might help team to better understand agile methods and improve them. Automatisation, finding on what teams strengths and weaknesses are and reflection on changes further help on improving the model.

Future evaluation of the results would be preferred. Testing in office environments might bring up some information that was not available during remote work. A sprint where more projects are currently going is needed to further analyse the added principles. Code quality and coverage tools needs further evaluation and studying as the thesis barely touched the subject. Regardless, the tools and work cycles remained roughly the same as before so it is highly possibility that the new model is as applicable to old environment.

The viewpoint of the project managers needs to be explored more. Management values some tools and practices differently than developer does. A re-evaluation of the model from the perspective of a project manager could bring some improvement ideas that were absent in this thesis.

7. REFERENCES

[1] Ohno T. (1988) Toyota production system: beyond large-scale production. CRC Press.

[2] Beck K., Beedle M., Bennekum A.V., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R., Mellor S., Schwaber K., Sutherland J. & Thomas D. (2001) Manifesto for agile software development , pp. 4–5.

[3] Stellman A. & Greene J. (2014) Learning agile: Understanding scrum, XP, lean, and kanban. O'Reilly Media, Inc.

[4] Royce W. (1970) Managing the development of large software systems: concepts and techniques. In: Proceedings, IEEE WESCON, The Institute of Electrical and Electronic Engineers, pp. 1–9.

[5] Balaji S. & Murugaiyan M.S. (2012) Waterfall vs. v-model vs. agile: A comparative study on SDLC. International Journal of Information Technology and Business Management 2, pp. 26–30.

[6] Petersen K., Wohlin C. & Baca D. (2009) The waterfall model in large-scale development. In: International Conference on Product-Focused Software Process Improvement, Springer, pp. 386–400.

[7] Forsberg K. & Mooz H. (1991) The relationship of system engineering to the project cycle. In: INCOSE International Symposium, vol. 1, Wiley Online Library, vol. 1, pp. 57–65.

[8] Mathur S. & Malik S. (2010) Advancements in the V-model. International Journal of Computer Applications 1, pp. 29–34.

[9] Bhuvaneswari T. & Prabaharan S. (2013) A survey on software development life cycle models. International Journal of Computer Science and Mobile Computing 2, pp. 262–267.

[10] Agarwal R., Prasad J., Tanniru M. & Lynch J. (2000) Risks of rapid application development. Communications of the ACM 43, p. 1.

[11] Abbas N., Gravell A.M. & Wills G.B. (2008) Historical roots of agile methods: Where did "agile thinking" come from? In: International conference on agile processes and extreme programming in software engineering, Springer, pp. 94–103.

[12] Martin J. (1991) Rapid application development. Macmillan Publishing Co., Inc.

[13] Reilly J.P. & Carmel E. (1995) Does rad live up to the hype? IEEE Software 12, pp. 24–26.

[14] Youssef M.A. (1992) Agile manufacturing: a necessary condition for competing in global markets. Industrial Engineering 24, pp. 18–20.

[15] Larman C. & Basili V.R. (2003) Iterative and incremental developments. a brief history. Computer 36, pp. 47–56.

[16] Gladden G. (1982) Stop the life-cycle, i want to get off. ACM SIGSOFT Software Engineering Notes 7, pp. 35–39.

[17] Liker J.K. (2004) The Toyota Way: 14 Management Principles From the World's Greatest Manufacturer. New York: McGraw-Hill.

[18] Poppendieck M. (2011) Principles of lean thinking. IT Management Select 18, pp. 1–7.

[19] Paul Brunet A. & New S. (2003) Kaizen in japan: an empirical study. International Journal of Operations & Production Management 23, pp. 1426–1446.

[20] Imai M. (1986) Kaizen: The key to JAPAN'S competitive success. New York, ltd: McGraw-Hill .

[21] Highsmith J.A. (2000) Adaptive software development: a collaborative approach to managing complex systems .

[22] Koch A. (2005) Agile software development : evaluating the methods for your organization. Artech House.

[23] Chowdhury A.F. & Huda M.N. (2011) Comparison between adaptive software development and feature driven development. In: Proceedings of 2011 International Conference on Computer Science and Network Technology, vol. 1, IEEE, vol. 1, pp. 363–367.

[24] Consortium A.B., Dsdm agile project framework handbook. URL: https://www.agilebusiness.org/page/ProjectFramework_00_welcome. Accessed 22.10.2019.

[25] Abrahamsson P., Salo O., Ronkainen J. & Warsta J. (2017) Agile software development methods: Review and analysis. arXiv preprint arXiv:1709.08439 .

[26] Poppendieck M. & Poppendieck T. (2003) Lean Software Development: An Agile Toolkit. Addison-Wesley.

[27] Cockburn A. (2004) Crystal clear: A human-powered methodology for small teams: A human-powered methodology for small teams. Pearson Education.

[28] The 13th Annual State of Agile Report. URL: https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508.

[29] Schwaber K. & Sutherland J. (2017) The scrum guide .

[30] Rising L. & Janoff N. (2000) The scrum software development process for small teams. IEEE software 17, pp. 26–32.

[31] Beck K. (1999) Embracing change with extreme programming. Computer , pp. 70–77.

[32] Williams L. (2003) The XP programmer: the few-minutes programmer. IEEE software 20, p. 16.

[33] Mar K. & Schwaber K. (2002) Scrum with XP. InformIT .

[34] Ahmad M.O., Markkula J. & Oivo M. (2013) Kanban in software development: A systematic literature review. In: 2013 39th Euromicro conference on software engineering and advanced applications, IEEE, pp. 9–16.

[35] Mahnic V. (2014) Improving software development through combination of scrum and kanban. Recent Advances in Computer Engineering, Communications and Information Technology, Espanha , pp. 281–288.

[36] Dingsøyr T., Fægri T.E. & Itkonen J. (2014) What is large in large-scale? a taxonomy of scale for agile software development. In: International Conference on Product-Focused Software Process Improvement, Springer, pp. 273–276.

[37] Ebert C. & Paasivaara M. (2017) Scaling agile. IEEE Software 34, pp. 98–103.

[38] Turetken O., Stojanov I. & Trienekens J.J. (2017) Assessing the adoption level of scaled agile development: a maturity model for scaled agile framework. Journal of Software: Evolution and Process 29, p. 1796.

[39] Laanti M. (2014) Characteristics and principles of scaled agile. In: International Conference on Agile Software Development, Springer, pp. 9–20.

[40] Scaled Agile I., Achieving business agility with safe® 5.0. URL: `https://www.scaledagile.com/resources/safe-whitepaper/`. Accessed 17.01.2020.

[41] Gandomani T.J., Zulzalil H., Ghani A.A.A., Sultan A.B.M. & Nafchi M.Z. (2013) Obstacles in moving to agile software development methods; at a glance. Journal of Computer Science 9, p. 620.

[42] Qumer A., Henderson-sellers B. & Mcbride T. (2007) Agile adoption and improvement model. Proceedings European and Conference on Information Systems (EMCIS) .

[43] Sidky A. & Arthur J. (2007) A disciplined approach to adopting agile practices: the agile adoption framework. Innovations in systems and software engineering 3, pp. 203–216.

[44] European Union, Data protection under GDPR. URL: `https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr/index_en.htm`. Accessed 05.10.2020.

[45] Huo M., Verner J., Zhu L. & Babar M.A. (2004) Software quality and agile methods. In: Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004., IEEE, pp. 520–525.

[46] SonarSource, Sonarqube. URL: `https://www.sonarqube.org/`. Accessed 03.04.2020.

8. APPENDICES

Appendix 1 Source code for limiting "In Progress" tasks in Jira.

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.bc.issue.search.SearchService
import com.atlassian.jira.jql.parser.JqlQueryParser
import com.atlassian.jira.web.bean.PagerFilter

//set transition condition to be passed by default
passesCondition = true

def jqlQueryParser = ComponentAccessor.getComponent(JqlQueryParser)
def searchService = ComponentAccessor.getComponent(SearchService.class)
def issueManager = ComponentAccessor.getIssueManager()
def user = ComponentAccessor.getJiraAuthenticationContext()
        .getLoggedInUser()
def assignee = issue.assignee

if(assignee != null){
  def assigneeName = assignee.getName()
  def query = jqlQueryParser.parseQuery("status = 'In Progress' and
        assignee = " + assigneeName)
  def resultCount = searchService.searchCount(user, query)

  if (resultCount >= 3){
  //set transition to be blocked
  passesCondition = false
  }
}
```

Appendix 2 Git pre-commit hook automated Jira message script

```
#!/bin/sh

update_jira_task () {
        IFS=' '
        SUB='-'
        INPUT=$(<$1)
        read -ra ADDR <<< ${INPUT}
        if grep -q "$SUB" <<< "$ADDR"
        then
                TASK=${ADDR%% *}
                URL="http://jiraurl.com/rest/api/2/issue/${TASK}/comment"
                curl -D- -u usernamer:password \
                -X POST --data '{"body":"'"$INPUT"'"}' \
                -H "Content-Type: application/json" ${URL}
        fi
}

update_jira_task $1
```