Tero Hakkarainen

Reform of Software Development Process in Metropolia IT

Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

30 November 2019

Metropolia

Author Title Number of Pages Date	Tero Hakkarainen Reform of Software Development Process in Metropolia IT 55 pages 30 November 2019
Degree	Master's Degree in Engineering
Degree Programme	Information Technology
Instructor(s)	Ville Jääskeläinen, Principal Lecturer Erja Nikunen, Principal Lecturer

This thesis is a report on a reform process that was done into the software development process of Metropolia IT department's software development team. The development task was done from the year 2016 to 2018.

The Metropolia IT software development team was a small development team and there were problems both with the working methods and the tools used in the development process. The starting point for the development task was a crisis with the personnel of the development team. The target was to develop a practical software development process for the Metropolia IT software development team in a situation where the development team was. A secondary target for the development task was to start using agile development methods as the main software development method and prepare the team to DevOps methods.

The development process was done in small steps and the members of the rebuilt development team were encouraged to give their input to the process. Reforms were done both to the processes and development support tools in the process and the steps taken were evaluated by the team before taking them into use. A big emphasis was addressed to develop communication in the development team and to obtain better software documentation methods both in source code and in technical documentation.

One key finding of the development task was that there exists no single correct universal software development process. The software development team must find what is the best method for them. Another key finding was that the process of re-developing software development methods in the development team is a continuous process. Software development tools and processes should be developed continuously. If the development of the software development team's working methods is stopped, the organization will collect knowledge debt which is similar to technological debt, when the software system is left unmaintained.

Keywords	Software Development, Agile methods, Teamwork, Development documentation, Continuous development

Metropolia

Contents

List of Tables

List of Figures

List of Abbreviations and Acronyms

API	Application Programming Interface is a method for software of software library to offer programmatical services to other applications.
BDD	Behavioural Driven Development is a software development process method that has an emphasis on combining both programmatical testing methods, like methods used in TDD, and other QA or nontechnical testing methods into one common software development testing process.
CD	Continuous Delivery is a software development method, where developed software source code is usually programmatically tested and after successful tests delivered to the target platform.
CI	Continuous Integration is a software development method, where developed source code is frequently shared with other developers in the project.
DevOps	DevOps is combined from words development and operations to describe combined methods and practices of both software development organization and technical infrastructure organization. A combined team of both developers and systems operators are usually called DevOps team.
EUR	EUR is an abbreviation for Euro, which is the official currency of 19 European Union member states.
Git	Git is not an abbreviation. Git is a common software source code version control system.
IDE	Integrated Development Environment. A common type of software development tool which combines common software development tools like code editing, debugging and compiling into on integrated toolset.
IT	Information Technology. Used in this thesis as an addendum to Metropolia in term Metropolia IT to describe Metropolia information technology organization.

Javadoc	Abbreviation from the words Java and documentation. Javadoc is a method to document Java source code in a way that is both human-readable and usable in IDEs.
JSON	JavaScript Object Notation is a method of describing data both as native data object for JavaScript programming language and as a human-readable text file.
Metropolia	Metropolia is not an abbreviation or an acronym, but it is used as a shortened version of the legal name of the Metropolia University of Applied Sciences.
Metropolia IT	Metropolia IT is a combination of the shortened version legal name of the organization to name Metropolia IT organization in the text and of Information Technology abbreviation. This combination does not specify any smaller part of Metropolia IT, like the development team.
PDCA	Plan-Do-Check-Act is a management method to control change in business management.
PHP	PHP is a programming language. PHP is an acronym for “PHP: Hypertext Preprocessor”. (The PHP Group)
PHPDoc	Abbreviation from the abbreviation PHP and word documentation. PHPDoc is a method to document PHP source code in a way that is both human-readable and usable in IDEs.
RDI	Research, Development and Innovation
REST	Representational state transfer is an IT architectural style to combine different data systems using stateless data interfaces. RESTful data interfaces commonly use JSON- and XML-files as data payload file formats.

SQL	Structured Query Language. A language to access, manipulate and maintain relational database systems and data stored into the databases.
TDD	Test Driven Development is a software development process method that has an emphasis on programmatical testing as an integral part of software development.
WIFI	WIFI or Wi-Fi is an acronym and originally a trademark created to promote wireless networking technologies. In its most common meaning, it stands for a wireless network.
XML	Extensible Markup Language is a method of describing documents in a format that can be read both by humans and computers. XML is commonly used in data interchanges between computer systems.

1 Introduction

According to Clark and Connor, “an optimal software development process is regarded as being dependant on the situational characteristics of individual software development settings”. They list different influencing factors, like attributes of the application being developed, attributes of the development team, such as size and experience and changing development requirements. (Clarke and O’Connor, 2012) While working as a team member in the small software development team, which was trying as hard as it can to be fast, responsive and agile in the constantly changing operation environment I noted that Clark and Connors's list is accurate. From my point of view, software projects fail if the development team’s working processes fail.

This thesis is a report of reform of software development processes in the software development team in Metropolia Information Technology department. It was triggered by immediate and sudden changes in the team and development started almost unplanned but mostly out of necessity. My role in this process was the activator and designer of the reform and I started this work independently: I noticed that there are difficult problems in the software development process and one must act on those immediately.

Therefore, my motivator for this process was first to get things done in a changing environment and secondly grow as a software development professional. However, I must state that the success of the process reform could only be achieved through teamwork. Therefore, I must begin this thesis text with a short thank you to my colleagues, past and present, who have inspired or influenced this thesis.

1.1 Business Problem and Development Task

As I stated previously the reform process started almost unplanned. Before the reform, we had noted as a team that there are problems in how we work, but that actual spark that ignited the reaction was an almost total loss of personnel resources. At the same time, we lost knowhow on how our in-house developed software was developed and how it was going to be supported or further developed.

Before December 2015 we had a situation where we had a development team of five developers and one trainee: My colleagues were a backend Java-developer, a full-stack PHP developer, a data warehouse developer and a backend Java- and PHP-developer trainee. I am most familiar with full-stack PHP development, but I also did specification and design work while working in the development team. The work of software development team was also supported by the trainee, who was at that time specialized in technical documentation.

At that point in time, it seemed that the year 2016 was going to be challenging because Metropolia IT had already scheduled a major software change: the student management system was going to be changed. That change meant that the development team had to update all in-house developed software that connects to the student management system.

In December 2015 the crisis happened, and challenges changed into acute problems: we knew beforehand that one of the developers was going to be unavailable for a long period starting from December. Then in the course of two weeks at Christmas holidays, two of the other developers decided to leave and gave their notices. When I came back from holidays, we were in a situation where only developers left was the data warehouse developer and a developer whose time and responsibilities were divided. So, in fact, the change was from about 3.5 software developer resource to about half software developer.

The situation was rapidly reacted by Metropolia IT managerial team and recruitment process for replacement developers was started immediately. However suitable candidates for permanent development positions were difficult to find in early 2016, so at first, we recruited developer trainee from our Metropolia ICT-studies faculties: a student whose software engineering studies were almost done. While we got some more resources, the first months of 2016 were pure survival: the situation was reacted on, and no new development was done. Then later we got two more trainees from our own talent pool, software engineering faculty of Metropolia, and we could move slowly from reacting to creating software in a more structural way. At that point, I had already started the reform of the software development process.

In the reactive stage of 2016, that is from January to February, I noticed two major problems, that I have stated previously: lack of resources and lack of knowledge on how software systems were developed and implemented. At the end of 2015 working methods in Metropolia, IT development team were based on about decade-old situation: developers wrote and deployed code independently and application code was stored in centralized version control. Required maintenance was usually done by the original developer. The support requests were gathered through helpdesk software, but actual repairs or code revisions were difficult to follow in the long term.

We had noticed that this method of working was outdated. We had also noticed, that it was difficult to combine modern tools to the process, which was almost solely based on IDE the developer's computer and a centralized code repository. We also required a fast sharing of information between developers on development projects and well-structured documentation of the software projects to secure knowledge transfers when responsibility is transferred from one developer to another.

The reform of software development process started in early 2016 at first as my own initiative, but as the first results were seen by the managerial team of Metropolia IT and more accurately my immediate supervisor I was given green light to the changes I was already implemented or planning to implement. My primary role as a software developer remained the same and my role as a process developer strengthened. The reform process did not have any concrete timeline when it should end, but rather it was decided that the process would be followed in the long term and it would not have any designated end date. My role in the reform process ended in the year 2018 due to changes in my role in Metropolia IT, but this thesis follows the process a bit further than that. In this thesis, I try to reflect the experiences I learned in this reform with my experiences on software development in my whole career and what I have learned with other employers in my career after Metropolia.

1.2 Research Target

I did not decide to write my thesis from the process reform at first, however early on, in March 2016, I noticed that this task has the potential to be subject of my master thesis. I had noticed two major problems: problems in resources and knowledge management

had started the first steps of the reform. Early on I also noticed that the reform must be a full software development process redesign, where changes should be implemented in small steps and those changes should be evaluated as teamwork.

The work was done step by step and for some of the development tasks I wrote reports to my software engineering studies and some of the development, tasks were reported as documentation, how-to guides or process directives. As a result of the development of the processes this thesis tries to answer the following research question:

What is the most practical software development process for the Metropolia IT software development team here and now?

We were not looking for “optimal” or “the best and the only” process. We were looking for a process where we could get things done reliably and accurately. We also acknowledged that we were looking for a modern and up to date solution, that we are hoping to be usable for the long term, but which also must be re-evaluated from time to time. We also knew that we must learn from our mistakes as a team and we also must redefine what teamwork means from our software development team point of view.

1.3 Research Method

When I decided to write a thesis on the development task, I knew that I needed a research method which could help me to react to the findings in the development task. The findings would mainly be problems that should be somehow solved. I also anticipated that I would find both human-related problems and technical problems in the process development, so methods used should also take human factors into account. In my previous studies, I have heard of the action research method, which is used to redevelop process where there is human action involved.

The term “action research” is in many sources said to become from the social psychologist Kurt Lewin’s article which was published in 1946. (Burns, 2005, pp.57–58; Carr, 2006, p.423; Koski and Kelo, 2019) In his article, Lewin introduces the term “action-research”. From experiences he had in human interaction situation during early 1940ies, where he needed to address human actions in social situations, he created a method to

evaluate human actions in different situations. He wrote that “the research needed for social practice can be best characterized as research for social management or social engineering. It is a type of action-research, comparative research on the conditions and effect of various forms of social action, and research leading to social action.” (Lewin, 1946, p.35) The method he found is quite close to what action research is said to be: he writes that in his process “proceeds in a spiral of steps each of which is composed of a circle of planning, action and fact-finding of the action.” (Lewin, 1946, p.38)

Ulla Suojanen writes in her article about the action research method that’s she calls every research action research if it fulfils the following conditions:

- the purpose of the research is to collaboratively develop some social target, functioning of a group, a named project or undertaking or a product

- the research is done with planning – action – observation – reflection cycles

- the members of the research team take active part in all the steps in the research process and

- the research process is documented

Suojanen emphasises the importance of the documentation because it will enable the researcher reflection about the process, it helps to get feedback from participants and documentation can be used as a method to evaluate the validity of the research. (Suojanen, 2004)

The action research process is built around planning – action – observation – reflection cycles. The development process reported in this thesis follows this principle in the natural transition phases of the development. During the development process, I also noticed that the action research method is very similar to the normal daily software development process, which is usually based on the process where a problem or a need is found, a plan to the fulfil situation is created and a software program is written, tested and deployed to end the development cycle. The reflection in the software development process is finding problems in deployed code or new needs informed by end-users. Thus, the development cycle can be continuous. I will discuss this further in section 4.

During the writing of this thesis, Metropolia started to use PDCA as a method for business process development. PDCA stands for Plan-Do-Check-Act and it is another method to handle changes in organizations. The action research and PDCA are very close to each other because both have the feedback loop of the development action phase through reflection. The PDCA emphasis a bit more the importance to react on findings in feedback phase, but the basic idea is the same: when the change is done, the impact of the change must be evaluated and if something has gone wrong, this must be corrected.

1.4 Object of Development Task

The object of the development task was to do a total reform of the Metropolia IT software development process. The aim was to combine the most suited tools with the best processes for teams development needs and achieve better quality in working processes and better quality of the final software product. This reform did not try to copy some company's or open-source projects process but tried to find the best possible combination of tools and processes for the Metropolia IT development team. The process team was trying to be able to react changes fast but at the same time secure long-term development needs.

1.5 Development Task Background and Introduction of Starting Point of Task

In section 1.1 I described the starting point of the development task as a crisis. That situation had to be handled as it happened, therefore this development task started without planning it beforehand. The reforms were just something that had to be done. Amongst the development that was already done, I noticed that I was doing larger changes than anticipated. That triggered a need to involve more people in the process and triggered a need to document the process and the results of the process somehow. This opened a possibility to write a thesis on this.

The resources were given to the development task as what is needed bases. The IT managerial team of Metropolia noticed a need for development process and gave permission to do tasks that do not prevent normal software development work. On the other hand, the managers gave permissions to urgently repair processes and tools that hamper software development process and that gave the software development team

possibility to rebuild tools and services if the team knew or found some tool or service to be outdated.

As the Metropolia IT software development team was in a state of a rebuild at the start of this development process, the timetable of this reform process was set to be open-ended. The process would take as long as it would take, and as long the process would generate positive results, it would continue.

In section 3 I will introduce more in-depth the early findings of the problems in the starting point of the software development process. In that section, those findings are expanded to the main development points covered in this thesis.

2 Metropolia University of Applied Sciences

As this thesis reports what was done in the reform process from the year 2016 to 2018, I will also try to open some background of what was happening in Metropolia in general and in Metropolia IT. What happened in Metropolia had a direct and indirect impact on the development task. The problems of the education system in Finland has been on the news for a long time and usually, the core of the problem is very often said to be lack of money: The Finnish society has difficulties to finance current education system.

Metropolia University of Applied Sciences is the largest university of applied sciences in Finland. In the year 2018 Metropolia had 16,408 students in over 60 degree programs. Metropolia had a staff of 917 employees and it operates in the Helsinki metropolitan area in 5 locations. Metropolia University of Applied Sciences operates at the higher education sector and provides education both on bachelor and master's degree level.

Metropolia is organized as limited liability company (official name: Metropolia Ammattikorkeakoulu Oy) and this company is owned by cities of Helsinki, Espoo, Vantaa, Kirkkonummi and Kauniainen. The Metropolia had a turnover of EUR 96.2 million in the year 2018 and Metropolia made a loss of EUR 0.06 million in the year 2018. (Metropolia Ammattikorkeakoulu Oy, 2019a)

Universities of applied sciences offer working life oriented higher education in 23 universities. Ministry of Culture and Education states that universities of applied sciences "are tasked with providing education for professional expert tasks is based on the requirements of working life and its development, as well as the premises for research and arts. In addition, they carry out applied research, development and innovation and artistic activities that serve education, support working life and regional development, as well as regenerate the industrial structure of the region." (Ministry of Education and Culture)

Metropolia fulfils its role in this by offering education to students in various degree programs. In 2018 Metropolia students completed 3 262 diplomas in bachelor or master studies. (Metropolia Ammattikorkeakoulu Oy, 2019a) Metropolia also offers continuing professional education for students who have completed bachelor or master studies but who needs deepened knowledge on a certain professional field.

At the beginning of the year 2016, the financing of universities of applied sciences in Finland started to change. To that date, the financing of universities of applied sciences was shared between the government and local authorities. The government financing was based on unit costs per student, project funding and performance-based funding. The universities also got financing from local authorities and from external sources. The framework for financing was agreed between the government, the university and maintaining organisations with three-year agreements on-target results and their monitoring and on national development tasks. (Opetus- ja Kulttuuriministeriö, 2013)

The reformed financing system of universities of applied sciences was based on the number of degrees, the quality of education processes, efficiency, the employment result of graduated students and with research and development functions of universities of applied sciences. 46 % of financing was based on the number of degrees and 24 % of financing was based on the minimum of 55 credit units accomplished by students per academic year. From the government point of view, this change will change the financing more performance and result based. (Opetus- ja Kulttuuriministeriö, 2013)

In 2013 Ministry of Education and Culture said that by concentrating on education quality and getting education processes more efficient the society in whole will get better results. (Opetus- ja Kulttuuriministeriö, 2013) In 2019 Ministry of Education and Culture is stating that Finland should be the most knowledgeable nation in the world and as a society, we need a workforce with better skills. Therefore, Finland must secure financing of higher education. In the 2019 financing model is increasing emphasis on degrees finalized: the more degrees higher education institution can produce more money they get. Also, higher education institutions are required to invest in developing continuous education. (Opetus- ja Kulttuuriministeriö, 2019)

The changes were done in the higher education financing model in Finland in the years 2013 and 2019 are part of the push to change the expense structure of the public sector. Also, the current economic situation and structure of financing of public services have a big influence on how much money the public sector can spend. Less tax income means fewer public services. One reason for public sector financing problem is the global depression that began back in 2008 with the sub-prime crisis in the United States. That crisis was followed in the European Union with Euro area economic problems. This has had a really bad impact on the Finnish economy, and it took a long time to recover: The

gross domestic product and the employment rate took nearly 10 years to recover. At the same time, the tax ratio had to be increased to ensure the functioning of public sector services. (Statistics Finland, 2019b, 2019a, 2019c)

When the Finnish economy is in crisis and the tax return of Finnish state descends, decreasing public funding for higher education is one of the tools to control state expenditure. The new higher education financing model is part of this. Finnish universities of applied sciences have criticized the trend of diminishing financing as they suspect it to endanger especially the quality of higher education. In one of the calculations by The Rectors' Conference of Finnish Universities of Applied Sciences (Arene) they have shown that external financing of universities of applied sciences has diminished 21.3 % from the year 2012 to 2019. (Rissanen and Ammattikorkeakoulujen rehtorineuvosto Arene ry., 2017)

There have been changes in political influence in the Higher education sector in Finland. The Government of Finland decided to cut primary funding from Finnish State to universities and universities of applied sciences after 2015 Parliamentary elections. A new government was elected after 2019 Parliamentary elections and this government has stated that it invests in knowledge and education. (Ministry of Education and Culture, 2019) From the higher education point of view, the crisis is not over. (Ammattikorkeakoulujen rehtorineuvosto Arene ry., 2019) In fact, the funding cuts and the actual impact of decisions made are an almost everyday topic: some are against the budget cuts, some understand the reasoning and there seems to be a consensus, that something must be done. (Ahtokivi, 2018; Hartio, 2016; Nieminen, 2017; Mauno, 2019)

Metropolia has reacted on the financing changes by changes in organisation and following through cooperation talks with staff in the year 2013, 2015 and 2018. At first, Metropolia set a target to cut costs by approximately EUR 6 million by the year 2015. Some layoffs were made but the biggest changes were made in organisation structure and with cutting two-degree programs. (Metropolia Ammattikorkeakoulu Oy, 2013) After 2015 co-operation talks there very not any layoffs, but after not renewing fixed-term employment contracts, retirements and other voluntary arrangements it was estimated to achieve cost cuts of EUR 7.6 million by the year 2018. (Metropolia Ammattikorkeakoulu Oy, 2016a) But in the year 2018, there was another round of co-

operation talks and layoffs of a maximum of 30 persons. It was targeted to have cost savings of EUR 5 million for the year 2019. (Metropolia Ammattikorkeakoulu Oy, 2018b)

The changes in Metropolia has somehow managed to also boost Metropolia's efforts in their core function: In education and enabling students to finalize degrees. In the following table, it is interesting to notice that financial figures and staff numbers are declining the number of degrees finalized has steadily risen from the year 2013 to 2018. (Metropolia Ammattikorkeakoulu Oy, 2019b, 2018a, 2017, 2016b, 2015, 2014)

Table 1. Selected facts and figures of Metropolia in the years 2013 - 2018

	2013	2014	2015	2016	2017	2018
Turnover	117.1 M€	109.7 M€	100.7 M€	97.8 M€	95.9 M€	96.2 M€
Net result	0.3 M€	2.6 M€	0.4 M€	0.3 M€	-1.0 M€	-0.1 M€
Completed bachelor's degrees	2 494	2 370	2 544	2 559	2 666	2 838
Completed master's degrees	232	289	327	409	394	424
Students	16 811	16 403	16 454	16 680	16 439	16 408
Staff	1 160	1 076	1 049	1 019	987	917

As an ex-employee of Metropolia, I must state that this cost-cutting was noticeable in every day working because savings was done in every other area of expenditure, as in the investment in IT development. At first Metropolia IT fared well in these turbulent times as there have not been any layoffs during the early rounds of co-operation negotiations, only in the last round one person in Metropolia IT was laid off. However, the actual working environment has changed, and work has, for example, become more fast-paced. In my opinion, the general atmosphere influenced a lot for peoples willing to work for Metropolia and was one of the reasons for the crises that started the reform process I am writing about in this thesis. In the year 2018 when I left Metropolia, I was not the only one in Metropolia IT considering other career paths, as quite many of my long-term colleagues decided to change employer.

Of course, the lack of money or problems in financing mechanism is only part of the picture: the higher education needed today is different than a decade ago. The higher education requirements of the future can be only estimated. Therefore, Metropolia must have means to react to what is happening in the society in whole, so Metropolia has revised and updated its strategy in years 2010, 2013, 2016 and 2019. The reform

process reported in this thesis was influenced by 2013 and 2016 strategies and in the following section, I will go a bit deeper on them.

2.1 Metropolia Company Strategy

The year 2013 and 2016 strategies of Metropolia have similar themes in them. For example, they share the same values. Metropolia created infographics to help personnel and other stakeholders to understand the strategies.

Figure 1: The 2013 Strategy Infographic (Metropolia University of Applied Sciences, 2013)

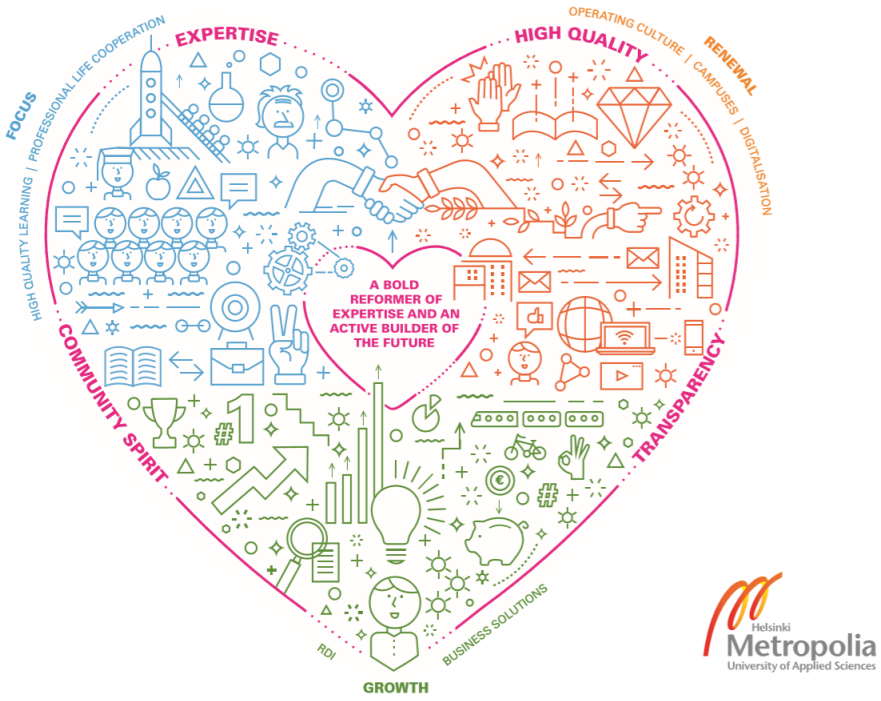

Figure 2: The 2016 Strategy Infographic (Metropolia University of Applied Sciences, 2016)

If the main themes were similar, the strategies had a totally different main vision. In 2013 it was “a vibrant metropolitan area” but in 2016 it was “a bold reformer of expertise and an active builder of the future”. The 2013 strategy stated that there are some megatrends that shape what Metropolia should be. These megatrends were sustainability, ubiquitous technology, increasing complexity of the world, changes in the ways of working and urbanisation. (Metropolia University of Applied Sciences, 2013) As Metropolia tried to identify megatrends influencing the operating environment it is obvious that this analysis has influenced 2013 strategy. I did not find similar operating environment analysis through megatrends in 2016 strategy documents that I had access.

The 2013 strategy’s vision statement emphasized Metropolia’s role in the Helsinki Metropolitan area. There was also the main goal set: Metropolia should be the most respected university of applied sciences in Finland. (Konkola, 2013, p.2) The 2016 strategy had three main goals: focus on high-quality learning and profession life co-operation, operating culture renewal through digitalisation and campus development and finally growth in RDI and business solutions to be more influential and get more external funding. (Metropolia University of Applied Sciences, 2016, p.3) From my point of view,

the 2013 main goal was a generic one and 2016 strategy was able to tell how Metropolia could become the most respect university of applied sciences.

In 2013 strategy there were four main themes that had a total of 12 strategic objectives for Metropolia. The main themes were inspiring learning, the energetic university community, workplace renewal and sustainable economy. The strategic objectives tried to set measures on how the main goal would be achieved. For example, creating flexible study paths, interdisciplinary studies and collaborative learning methods was mentioned as objectives how the target of inspiring learning can be achieved. (Metropolia University of Applied Sciences, 2013)

in 2016 strategy was categorized differently than 2013 strategy and did not use slogan style categorizing but more objective categories. There were six main categories that had a total of 25 strategic objectives for Metropolia. The main categories were education, operating culture, digitalisation, campus renewal, RDI and business solutions. The similar themes from the 2013 strategy can be found in 2016 strategy, for example in education flexibility of study paths can be found. The 2016 strategy of Metropolia can be compacted to following four statements (Metropolia University of Applied Sciences, 2016, p.4):

- We offer flexible, ever-renewing learning possibilities as well as services and solutions for the open world.
- We offer an environment where the staff, students and partners together develop their expertise and create something new.
- We are influential and have strong international networks.
- With our open, experimenting operating culture we improve society for everyone.

If we study the objectives in objective categories a bit further there are a couple of objectives that are puzzling to me: One of the objectives states that “we become we”. The second “we” is in bold. This indicates to me, that because Metropolia is multidisciplinary, there are difficulties to do internal co-operation. (Metropolia University of Applied Sciences, 2016, p.6) Another problematic objective is the whole digitalisation strategic intent: with these Metropolia is stating, that everything has not gone as it should have in the first eight years of the university in the digitalisation. (Metropolia University of Applied Sciences, 2016, p.7) In my opinion, if organizations do process development,

currently the digital methods are the normal methods to develop processes if the process can be supported with digital tools. In my experience, there has been a lot of development in higher education digitalization but there is still a lot to do. But as I think that digital processes are the normal way to develop processes, I would not put it in the strategy anymore.

The corporate level strategy of Metropolia had a small influence on my development task, especially on the operating culture strategic intent. I can really relate to the statement of that intent where it is set that “Metropolia’s operating culture is based on community spirit expertise, inspiring all to give our best to reach our common goals. Our operating culture is visible in both efficiency of the every-day operations as well as in our agile and experimental spirit”. (Metropolia University of Applied Sciences, 2016, p.6)

2.2 Values of Metropolia

According to Johnson et al, “the core values are the underlying principles that guide an organization’s strategy”. (Johnson, Scholes and Whittington, 2009, p.112) The values set the foundation of how the organization sees itself and enables developing mission and vision of the organization and this help forming objectives to the organization and further the strategy of the organization. In fact, Johnson et al state, that it is an important managerial task to decide how the organization expresses its strategic purpose stating values, vision, mission or objectives. (Johnson, Scholes and Whittington, 2009, p.115)

In both year 2013 and 2016 strategies of the Metropolia, the core values are in-fact the same. The values have a small description that opens them further (Metropolia University of Applied Sciences, 2016, 2013):

- expertise, our passion
- high quality, our target
- community spirit, our source of strength

- transparency, our operating policy

These values act as a compact guidebook on how Metropolia should work.

2.3 Metropolia IT Department

The organizational strategy is dependent on many things and is a long-term commitment. As Johnson et. al. states, “strategy is the direction and scope of an organisation over the long term.” (Johnson, Scholes and Whittington, 2009, p.18) They also state that strategy is implemented in different levels in the organization with different scopes.

In this section, I will introduce the operating environment where I did the development task for this thesis.

In 2016 when the reform process started Metropolia IT services were divided into three main service teams. These were:

- help desk and workstation support services team
- systems and network support services team
- information management and systems development services team

In 2016 Metropolia IT services was led by Chief Information Officer Tuomo Rintamäki and his chiefs who run before mentioned teams. The CIO of Metropolia reported to the Director of Academic Services Pekka Korhonen. The operations of Metropolia IT were also supervised by Metropolia IT steering group, which was re-established in 2017.

The main processes of Metropolia IT were following (Rintamäki, 2010, p.11):

- information systems project portfolio management from the service development point of view
- information systems project implementation management

- IT change management
- support request management and solutions
- identity and access management

In 2016 Metropolia IT services operate in 17 facilities with a network of over 5000 workstations and servers in the Helsinki metropolitan area. It offered support and services to four primary customer profiles: students, staff, partners and guests. All the services were offered in multiple platforms on desktop, web and mobile and the most popular operating systems were supported. Metropolia is changing its campus structure and the number of campuses will diminish to four in the year 2020. This will change what kind of IT resources are needed in Metropolia in the future. (Metropolia University of Applied Sciences, 2019)

While Metropolia IT offered support primarily for in-house customers, it also sold services outside customers mainly in the education sector. Metropolia IT offered support services for upper secondary level schools and IT consulting services for other higher education institutions in Finland. Metropolia IT also provided expertise on acquisitions and competitive biddings to other higher education institutions.

Metropolia IT had also developed software in house for its own use and for resale if the developed solution had markets for example in other higher education institutions. Metropolia IT is also a founding member in Peppi – consortium: Peppi is tailored ERP solution for higher education institutions which has been jointly developed by Metropolia and Tampere University of Applied Sciences. These two organisations are also the founding members of Peppi consortium. (Peppi-konsortio)

Since I have left Metropolia, it has come to my knowledge that there have been changes in the organization structure and managerial personnel of Metropolia IT. This thesis does not discuss them.

2.4 Strategy of Metropolia IT Department

Metropolia IT has had its own strategy document, but it was from the year 2010 and it was for years 2010 to 2014. That strategy document has been written with some knowledge of the recent big changes coming to the higher education sector. (Metropolia Ammattikorkeakoulu Oy. 2010, p.4) However, this document has not been updated to include the latest Metropolia level strategy updates and therefore cannot be used as a totally up to date description of Metropolia IT strategy.

In the IT services strategy document, it is stated that Metropolia IT services have a strategic and operative role and function. Firstly, Metropolia IT has to offer services to fulfil core business processes. Secondly Metropolia IT has organised to customer-oriented support services both to students and staff. Thirdly the IT infrastructure has to be reliable. (Rintamäki, 2010, p.4) In the 2010 strategy document, it is stated that Metropolia IT services have the same intent and goals as the Metropolia in whole. (Rintamäki, 2010, p.5)

Metropolia IT started to update IT strategy document in 2017 but the results of that process did not influence the reform process described in this thesis. This thesis does not discuss the updated Metropolia IT strategy.

3 Current State Analysis of Software Development Process in Metropolia IT

During the reform process, the Metropolia IT software development team was a small team: usually, Metropolia had only between 2 to 4 person-years of active development resources per fiscal year. The development tools were closely connected to programming languages and target platforms in use.

Metropolia IT created software mainly for web deployment. That meant that end-users were using developed software with a web browser. Even though the importance of mobile devices such as smartphones and tablet computers has risen lately, the main target was personal computers. The web applications in Metropolia were developed mainly with Java or PHP and programming language selection was done per-project basis. However, some of the development was done in Microsoft development toolchain: Microsoft SQL Server development can only be done with Visual Studio tools. Most of the time the developers programming skills were the deciding factor of the programming language selection.

The development was done with desktop computers that had developer preferred tools per development language. Some developers used IDE's like Eclipse, Netbeans or JetBrains development software, some preferred simple text editors. In most of the projects, the code was deployed manually to web servers. There was a couple of projects where projects were deployed using Jenkins continuous integration and deployment server. Those projects also used unit testing in their deployment process.

The source code was stored in centralized version control, which was using Subversion version control software. There was also a Git server that could be used because developers usually prefer Git over Subversion. The general popularity of these two solutions can be measured and compared for example using Google trends, where we can see that popularity of "git" as a search term has steadily risen since 2005 when using "subversion", "svn" and "mercurial" as a search term has fallen steadily since 2009. (Anon, 2016) As most of our developer team had used public Git-based version control services like Github and Bitbucket, we chose to move every source code repository to Git-based local server or remote version control web service provided by some cloud service provider.

At the beginning of the reform, the development process for new software was almost like a hybrid from the traditional waterfall and agile/lean development model. The following picture, which Huo et al. have created for conference proceedings, illustrate how waterfall and agile methods differ. The definition and the design phases have comparable steps; however, the actual software development phases differ completely. In the waterfall model, the target is to have one development phase which is concluded with required testing. In the agile method, the development team will create final software in smaller steps, in agile terminology sprints, and after every step development team will try to release new features or fixes to already released software. (Huo et al., 2004)

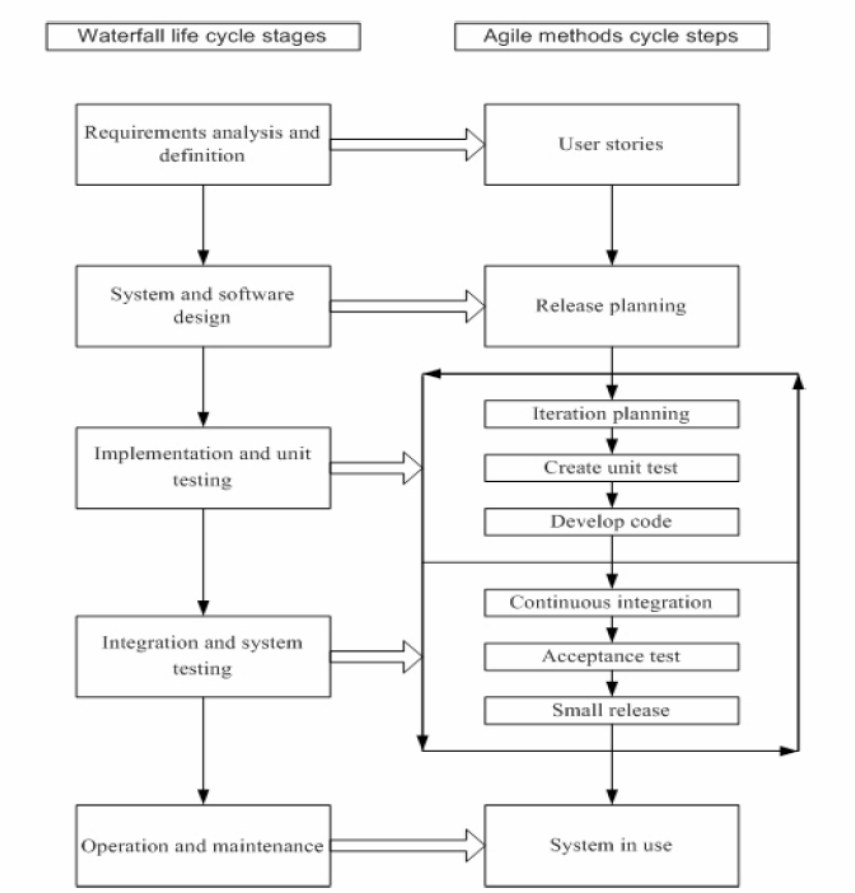

Figure 3: Waterfall model vs. agile methods life cycle (Huo et al., 2004)

In Metropolia IT the development method was usually such that one main developer created the software using hybrid model, where definition phase is quite traditional: user and process needs are collected to create minimum requirement set and before actual software development is started required system design is done, like database

modelling, use case modelling and user interface design using paper or wireframe prototyping.

The development phase was close to the agile method: as development is usually done by one developer, the developer can do develop – test – release – get feedback – react phases as agile or fast as the development process requires. In fact, this method is close to action research method, which can be used in developing working processes in companies and organizations: In the action research process, as shown in figure 4, process developer starts with a process that is in use and evaluates the problem as it is. Then he does require research on the subject, for example, interviewing process actors to get feedback or doing benchmarking of the process to best practice process models. After evaluation and research process re-development is done and required changes are implemented, so that process will reflect organizations needs for process best practice at that point in time. Usually, this process is not one cycle does its type of development task, but process developers do action research development as many times it is needed (Costello, 2003, pp.1–11)

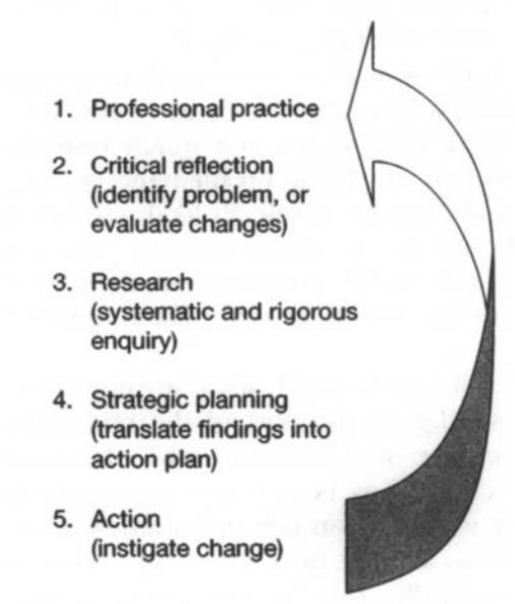

Figure 4: A representation of Denscombe's action research model (Costello, 2003. p.11)

In my point of view, the software development process in Metropolia IT was also very close to process development, especially when they did digitalization development. Metropolia IT had to have good knowledge of the process in use and needs on how the

process will be redeveloped. Only then the development team had enough knowledge to create software that meets requirements. The working method used could get requirements from end-users and could react on changes needed for design, development or support phase. However, the following problems in working methods were recognized:

1) Development is one developer's task: one developer designs and creates software, documents it and supports it as long it is needed. This leads to person dependency.
2) Deployment is done by hand and the deployment process usually does not include automated testing.
3) Technical documentation or code commenting is seldom done as thoroughly as needed, mainly because the original developer supports software as long it is needed.
4) As development is one developer's task, unspoken knowledge of the software is not shared every time. This has something to do with workloads also, but teamwork is also difficult for example because of the version control software
5) The version control software used made it difficult to just browse the source code or fork the software to a new software version.
6) Support requests are only followed by helpdesk software and actual changes to the software is difficult to follow.

From these problem points the main development targets for the Metropolia IT software development process are:

a) Enhance teamwork with new methods and tools
b) Create better documentation processes and tools
c) Begin utilizing unit testing procedures
d) Change of the version control system
e) Create deployment best practice process and take it into use
f) Create better end-user support processes and tools

4 Software Development Process in General

In modern software development, agile methods are becoming the almost de-facto working method. Either it is Scrum, Kanban, Lean or any other agile method, development organizations do think that they are creating better software through these methods. (Denning, 2015) The agile methodology itself is partly based on Agile manifesto, where a group of software development professionals laid down twelve principles of agile software. These principles are following (Kent Beck et al., 2001):

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

During the reform of the Metropolia IT software development process, I accomplished the Certified ScrumMaster certification by the Scrum Alliance (Scrum Alliance, 2019). This gave me another point of view into agile development methods.

Scrum methodology tries to help a development team to better work together. Its core values are commitment, courage, focus, openness and respect. (Schwaber and Sutherland, 2017, p.5) Development work is organized as the scrum team and members of the team have formal roles and responsibilities. The named roles are the scrum master, the product owner and the development team. The scrum master is a kind of a coach, which helps the scrum team to achieve set targets. A product owner is a person who tries to maximize the results of the scrum team, with for example watching over needs and targets of the customer or the organization. A development team is a group of development professionals with skills needed to achieve set targets in the Scrum process. (Schwaber and Sutherland, 2017, pp.6–7)

The Scrum method is organized in sprints. The sprint is a fixed set of tasks that will be done by the scrum team during a set time period. The sprint has fixed practices to help the scrum team to do the best possible work: The sprint planning, where tasks are set. The daily scrum meetings, where progress is constantly monitored and finally at the end of the sprint are sprint review and sprint retrospective. In these meetings, it is monitored what has been done and what must be learned to the following sprint. (Schwaber and Sutherland, 2017, pp.10–14)

The Scrum method has similarities with action research and PDCA methodologies, as it has a built-in feedback loop from the development cycle. Scrum method tries to prevent repeating mistakes or false design choices that create a bad product. Schwaber and Sutherland state in their book “The Scrum Guide” that “scrum employs an iterative, incremental approach to optimize predictability and control risk.” (Schwaber and Sutherland, 2017, p.4)

The things that I learned from the Scrum certification course influenced the reform of the software development process as I tried to take processes and methods that could help from Scrum methodology into the reformed process. However, the reformed process was not a by-the-book Scrum process.

Modern software development process commonly includes DevOps with agile development methods. Hüttermann states in his book, that there are many definitions of DevOps, but also gives his own definition of DevOps. Hüttermann writes that “DevOps is a blend of development (representing software developers, including programmers, testers, and quality assurance personnel) and operations (representing the experts who put software into production and manage the production infrastructure, including system administrators, database administrators, and network technicians). DevOps describes practices that streamline the software delivery process, emphasizing the learning by streaming feedback from production to development and improving the cycle time (i.e., the time from inception to delivery).” (Hüttermann, 2012, p.4)

The concept of DevOps was born from the need to recognize that agile methods have an influence on the infrastructure which is supporting the software development project and on the delivered software product. It also recognizes that development project benefits from agile infrastructure. The idea of DevOps is said to originate from Patrick Dubois, who wrote one of the first articles on “Agile Infrastructure” and the actual term was popularized through DevOpsDays conferences, which started the year 2009 in Ghent, Belgium. (Debois, 2008; DevOpsDays, 2009; Hüttermann, 2012)

Now, DevOps is seen as a combination of process culture, process automation, action measurement and information sharing principles. Process culture must support the idea that end-user software quality experience is dependent on every member of software development and deployment process. Process automation can and must support timely and accurately development and deployment process and this whole process must be measured with valid and measurable metering. All these operations are built on sharing - information, code, problems – and sharing, therefore, has to be supported both in development and infrastructure processes. (Fitzgerald and Stol, 2015)

Fitzgerald and Stol have collected business development tools, software development process tools, and DevOps tools to a combined continuous software engineering roadmap, which is pictured in figure 5.

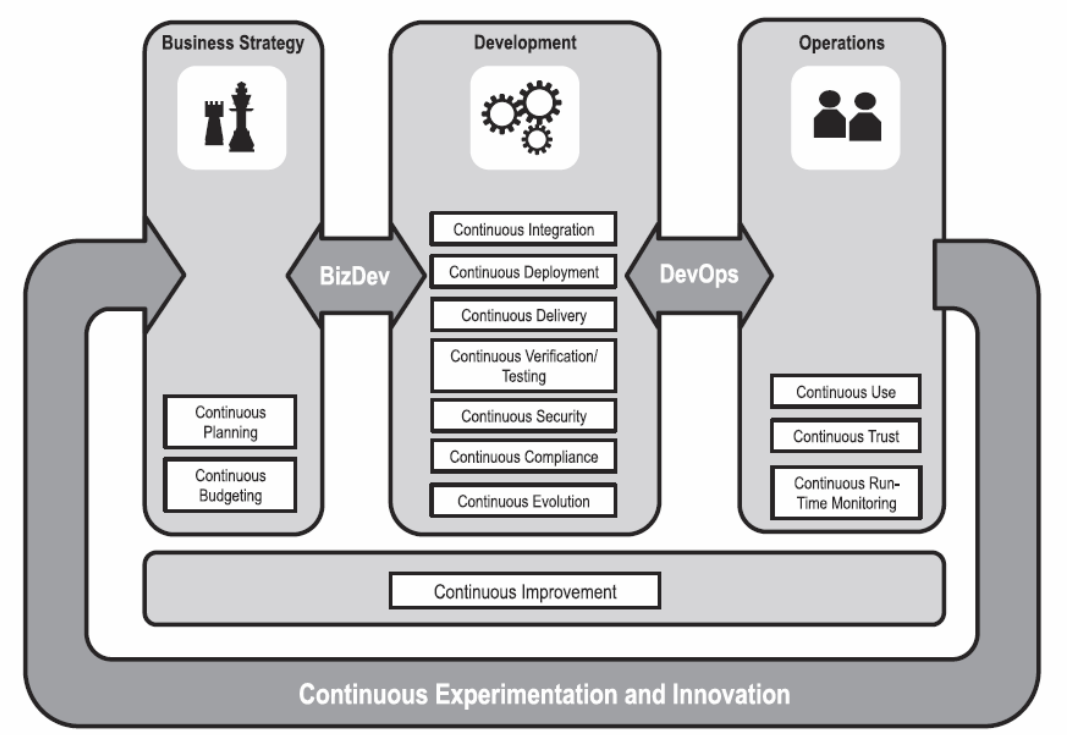

Figure 5: Continuous∗: a holistic view on activities from business, development, operations and innovation (Fitzgerald, Stol. 2015. p 183.)

This continuous software engineering roadmap is called Continuous* (or Continuous Everything) and it consists of three main sectors: Business strategy, Development, and Operations. Therefore, it recognizes that organizations have all the time business development needs that have an influence on software development. Software development reacts to both changes in business and changes in technical operations infrastructure. All of the continuous improvement processes at the end enables continuous experimentation and innovation in organizations. (Fitzgerald and Stol, 2015, p.184) There are, again, quite a lot similarity with Continuous* and action research methodology: both methodologies have a big emphasis on that things do not stay the same and organizations do have to notice, learn and react to change.

Metropolia IT could use Continuous* to fine-tune its software development process reform. Continuous* could help Metropolia in asking the right questions of what they are doing right and wrong. It also categorizes continuous processes in software development in basic level and, therefore, streamlines the search for suitable processes and tools for Metropolia IT.

5 Development on Software Development Process in Metropolia IT

In section 1.1 I stated the starting point of the development process was a resource and knowledge management crises in the first months of the year 2016. The actual process run throughout the year 2016 and the last process development tasks I took part in were done in the first months of the year 2017. Process development tasks were done in a way that it would not cause to long disruption to our normal working process. Another aim was that the solutions to tasks should be rapidly implemented and easy to use.

The development tasks were usually done in the following pattern:

- Recognize the development task
- Take opinions from the team about possible task solution in round table talks
- Plan the development task solution
- Test the development task solution with one team member
- Refine the solution according to test feedback
- Create team-wide task solution implementation
- Receive feedback, react to accordingly and use gained knowledge in following development tasks.

Not every development task was implemented the same way as adjustments were done if needed. A few setbacks were met during the tasks but most of the targets of the development tasks were eventually met. We did not have a fixed timetable for tasks because important events, like the student management system change, would always be the higher priority. However, the development tasks did have a positive impact also into the major system change.

In the following sections, I will open the development tasks in more detail.

5.1 Reorganization of Software Development Team

As stated previously, the re-development of the development process started almost without an actual decision to start it. However, before the actual beginning of the process, we had noticed commonly in the development team that something will have to be done. We had planned some steps to be done, mainly to the tools of the development process, but we had not had enough time to test and implement new tools.

At first, we thought that most of the needed improvements could be achieved with software tools. For better documentation and development tracking, we thought that using more modern repository tools than Subversion would be enough. For better software deployment process needs we thought that implementing test-driven development process where possible and building a continuous integration and deployment environments would be enough. In fact, we had already started the evaluation of different products for this development task.

By evaluating the whole development process, we can now notice that the actual beginning of the process was the crisis where we lost developer resources from the team. At that point, we decided to take time out on tools development and concentrate on creating a functional development team. The first task was to find new developers with suitable skills. As a higher education institution, we have a possibility to contact our internal partners in information technology faculty and ask them to help locate suitable candidates for trainee positions. So, as a stop-gap for acute resource need, we decided to contact them and found a good candidate with needed skills and hunger to learn more tools. That candidate was hired almost immediately. We also decided to try to recruit a software developer in a permanent position, but we did not get any applications with needed experience and skills. In 2016, there were many open positions for experienced Java developers and as a public-sector institution, we were not competitive enough for potential persons. After the recruitment process, the developer who we had hired for trainee position had shown us what he can do so we decided to offer him a permanent position and he accepted. After that, we decided to hire one more trainee so that we had enough developer resources for big projects in the year 2016.

Before the starting point of reform at the end of 2015 we had following developer resources:

- Senior Java-developer, backend, 100% resource, decided to leave Metropolia in January 2016.

- Senior PHP-developer, full-stack, 80% resource, was not available for most of the year 2016.

- Senior PHP-developer, full-stack, 50% resource

- Senior data warehouse developer, 80% resource

- Java/PHP-developer trainee, full-stack, 100% resource, decided to leave Metropolia in December 2015.

At the same team where software developers work, there was also one trainee, who focused on technical documentation and another trainee that concentrated on technical prototyping. These persons were not used in the software development projects at the start of the reform, so at the beginning of the reform, these persons were not included in the developer team.

In early 2016, we decided to form a close working software development team in Metropolia IT. Before the forming of the team, each developer had worked independently: developers were responsible for their work almost eternally and responsibilities were not shared. Early on it was decided as a team that this must be changed. Most of the time developers had a full workload, but in problem situations, it was needed to find a developer with needed skills to solve the problem. Sometimes our original way of working would create bottlenecks were problem-solving would wait for an open time slot in developer's pipeline, or in total emergency situations, we would have to make decisions which problem is more fatal. This type of problems is common in every organisation with tight resources, but we believed that doing small changes in our working methods we could offer better and more timely service to our customers.

Early on we noticed that we could not just state that now on we share responsibilities on every development project or supported software. We noticed that, firstly, every software must have a lead developer, whose responsibility is to guide development according to technical specification. Secondly, we must lower the bar on information exchange in

between development team. Therefore, in January 2016 we decided to start weekly informal meetings within the software development team which the only agenda is to keep everyone in the software development team informed about what is being done. We gave meetings a bit “laid back” name, “Koodarikahvila”, or “Coder Café” in English so that everybody involved would now that even though we had set of principals we would try to create an open and relaxed environment for developers to share information and ideas.

The form of the weekly developer meeting is round table discussion on what everybody has done or will be doing soon. At first, everybody was informed of three principals of the meetings. The principals were:

- Everyone gives other team members a presentation on what they have been doing since the previous meeting

- Everyone gives other team members a heads up on what they are doing soon. They should also inform others if they need some help or other resources on a specific task.

- The presentation should be short, max. 10 minutes or even shorter.

At first, we decided to create a memo for meetings during the meeting as normal meeting minutes. However, after the first meeting, we noticed that the form of the meeting would benefit everybody writing their bullet points on meeting wiki page beforehand. We also decided to create a simple method to measure workload, where simple traffic light metering was used to indicate workload.

- The green light indicated that everything is under control. The developer can do tasks in a timely manner and new larger development can be started with a minimum delay. Also, emergency development tasks, like fixing broken software are started without a delay.

- The yellow light indicated that the developer’s workload is busier than desired. Developers tasks are delayed somewhat, and larger development tasks should

not be started without some task management. Emergency development tasks could disrupt the normal workflow and could be started with a delay.

- The red light indicated that the developer's workload is not desired and development tasks will be delayed. Larger development tasks should not be started at all, because new tasks would disrupt workflow. Emergency development tasks should be transferred to another developer.

We also created a wiki page template for meeting memo and meeting memos were decided to be open for the team which virtual developer team is part of. At first, only developers took part in the meeting.

Few months after we had started the developer meetings, we evaluated the results we had gained from the meetings with the following criteria:

- Do we have better information about what the developers are doing or what challenges they have?

- Could we react to acute resource needs more timely manner?

- Do we create valuable information for the team with meetings?

- Should we include some other persons to the meetings?

We believed that we got a better picture of what the team is doing with these meetings. Everybody got a rough picture of what other developer is doing and through memos, the managerial team has the possibility to follow the work as needed. The response time to acute development needs did not have clearly metered impact, however collegial discussion on acute situations gave better tools to problem-solving and some situations responsibilities could be shared effectively.

The meetings created valuable information on what and how the software development team is working. However early on we noticed that even though we can share information in face to face meetings and with meeting notes, our documentation was sometimes lacking details or in some cases totally absent. Therefore, we decided early on

developing tools and methods for better documentation both in technical form and as code documentation.

After we got the desired results from the meetings, we decided that everyone who is involved somehow in the software development should be included in the meeting. At first data, the data warehouse developer did not take part in the meetings but soon we decided to include him also. Also, meetings gave us a unique insight into how we could organize our work. The trainee who was at first only responsible for technical documentation in one large data system took more responsibility for creating documentation guidelines. We also decided to give this trainee a possibility to widen her skills, and after a while, she took a bigger role in user interface creation process and for example created commonly usable theme templates for web user interfaces.

The weekly developer meetings were set a permanent fixture. They were usually compact meetings taking 30 to 45 minutes per week. However, if the team had a more difficult subject to discuss on, it could always have a longer meeting and really find common ground on the subject. The meetings could be held as face to face meetings or virtual meetings using Skype for Business. In fact, in early 2017 one of the developers had the opportunity to work remotely from Thailand and later in 2017 he had the opportunity to work remotely from Greece and he took part on the meetings with Skype for Business without any problems regarding distance or time zone where he took part to the meeting. Problems he had could have easily happened from the same building on our campus, like WIFI-connection problems or bad microphone configuration on Skype.

5.2 Introduction of New Software Development Tools

After the development team had made changes on how we communicate on development on the campus, the team decided to check, evaluate and possibly update the software development tools it was using during the development processes. On tools, the team decided early on, that if the target is to create continuous everything environment or even only DevOps environment, then tools chosen should support this change. We decided on the following development tasks.

- Software development source code repositories should be moved from Subversion repositories to more modern code repository software.

- Repositories should support collaboration. Collaboration should be possible for external developers out of Metropolia and support different levels of privileges. Preferably source code repositories should be accessible through web-interface so that persons who only require review access to code can access code without desktop repository client

- Software development documentation should be centralized to one location and updated to a more current state.

- Workload management should be expanded for example using ticketing software in software development projects.

- If possible, all new software tools should be connected somehow and be a kind of “active tools”. These tools connections could be achieved using instant messaging system integration.

- And therefore, we should test different team-based instant messaging platforms that have integrations to software development tools software.

The team started these development tasks parallel to each other, however some tasks required to be finished before we could go forward with other tasks. As the team was using Subversion as main software repository system and because no one of the software developers was not eager to use it any longer, the team decided to start with that task.

At first, the team was required to make the decision of the repository system. As the team had enough of Subversion, it looked at the other modern repository systems which have current ongoing support. Overall, the best experiences the development team has had with Git, which is an open-source distributed source code repository system. The team have had some experience with other software, like Mercurial, but those experiences were not good enough to compete with Git.

After Git was chosen to be the new code repository software, the team started to think about how it could reach other goals we had set, like flexible privileges and web browser access to source code. As the team had decided that it should have web browser access to code repositories, it noticed that it must decide whether repository software is going to be hosted on Metropolia servers, or should Metropolia get the software hosted as a service from a service provider.

Web-accessible Git repositories as a service are provided by many software vendors, like Github, Gitlab and Atlassian Bitbucket. Github is paid service which had a free tier for open source projects, but private repositories were not possible in free licence tier. Github did not distribute its software for on-premise installation. For these reasons we did not evaluate Github further, because early on we decided that we need private repositories in almost all software development projects.

Gitlab had git-based repository software available both in the cloud and on-premise installations. They also had a community version of their software that could be installed without license costs to on-premise servers. Atlassian Bitbucket was offered with a bit different license scheme, where on the cloud a small team could have limited number of private repositories, but actual on-premise installation or large-scale cloud Bitbucket usage would always require a paid license. However, the team did some preliminary tests with Bitbucket's cloud service to evaluate if the team was on track on our plans on what was required for modern source code repositories. For example, the team tested Bitbucket's integrations with Atlassian's own team based instant messaging system HipChat and was able to effortlessly create integration where a message was relayed to common chat channel in HipChat about a code deployment to code repository in Bitbucket.

While the team was evaluating cloud-based repository systems it was contacted by a Finnish company called Deveo, which offered a comparable product to Github, Gitlab and Bitbucket: code repositories with a web-based user interface, integrations to different software development tool systems and some more, like an inbuilt wiki for creating software documentation. Deveo had also very flexible user privilege system on paper and it could be installed on-premise in our own servers or be used as cloud service. As Deveo were interested to have higher education customer they decided to offer the team

long and flexible terms on evaluating Deveo using their cloud service and the team decided to evaluate it using real-life usage scenarios.

For the actual evaluation task, the team created the following evaluation questions:

i. How could Deveo enhance teamwork in Metropolia IT?
ii. Is Deveo Wiki features enough for Metropolia IT?
iii. Is Deveo Wiki features easy to use?
iv. Are Deveo source code version control features enough for Metropolia IT? Deveo should be evaluated in at least the following feature sets:
    a. version control workflows
    b. code preview
    c. code review
    d. branching code repositories
    e. merging branches
    f. using version control as a document repository
v. Are Deveo source code version control features easy to use?
vi. How do integrations work in Deveo?

In the evaluation, it was found that In Deveo workflow developers worked in projects. The project was managed through a set of views, which include Activity, Repositories, Code Reviews, Hooks, Issues, Milestones, Team, and Wiki. Activity combined project activities to a single view. Repositories were version control-based repositories for code and other files. Code Reviews were used to control feature integration voting. Hooks were used for controlling external software or services. Issues were lightweight issue tracking and Milestones was used to set project milestones for the development team. The team itself is controlled through Team view. There was also a Wiki view, which was a simple wiki platform compared to more complex wiki platform like Atlassian Confluence.

Usability of the web user interface of Deveo was as good as expected for a modern and up-to-date web application. Even though the application had a large feature set, the user interface was not cluttered. Also, most of the functions were self-explaining and seldom users had to read the user guide to learn how to use functions of the system. The source code repository functions and integrations were found to be good and on par with

competitive products and even though wiki was found to be quite simple, that fact that wiki was git-based and therefore under version control, it was quite handy for software development purposes.

The result of the evaluation process was a recommendation for the head of the Metropolia IT development team on should we continue planning our reform process using Deveo tools. The recommendation was that from the developer team point of view Metropolia can start using Deveo and after the recommendation, the decision to acquire Deveo license was done.

As the development team was evaluating software repository systems, it had already started the change in the documentation process. As stated previously, the team had started to have weekly developer meetings that were documented as wiki-pages in Metropolia corporate wiki, which is based on Atlassian Confluence. These meeting minutes were decided to be saved to “almost developer-only” wiki-site. The team had a developer wiki before for documenting different development projects but that had been almost forgotten. The site was called “coding best practices”-site but either we did not have any good practices, or we did not know how to report them. Sometimes It is not required to reinvent the wheel, but some action is needed, so we decided to rename wiki-site in the same spirit as developer meetings. The team gave the site name “Koodariwiki” (in English “The Coder Wiki”) and established new rules on how the site will be used in the future.

The development team had noticed that fact, that most of the times code is not enough as software documentation. If the team has some software packages in support that do not require constant redevelopment, the knowledge on technological base, software routines and control flow of the application is forgotten quite rapidly. “Koodariwiki” was decided to be developed into a hub of the software development documentation.

As the team got started with enhancing the documentation, the team also stopped for a while to think about if it could find suitable tools for the development process and the event handling while users of our software find errors in the software. Usually, these event handling tools support ticketing systems and luckily, Metropolia had already such installed: The Atlassian JIRA. JIRA had been installed into Metropolia servers to support

a totally different support process, but as it was already installed, the team had a very low threshold to start testing JIRA in the software development process.

The actual testing process was quite brief. Early on the found out, that basic workflows of JIRA have all the steps usually needed in ticketing a software bug fixes or new development tasks. The biggest problems were with user permissions of JIRA. Finding the right combination of needed permissions to users and roles was a bit difficult because there were some changes to JIRA settings compared to the default installation of the software.

In the beginning, the main usage method of JIRA was ticketing of the development tasks e.g. tasks, new features, improvements and, of course, software bugs. The software packages which were under constant development or support were organized as individual projects. Early on it was also decided that new development projects that were identified to create software that would be continuously supported, would be added to JIRA immediately when development started.

The normal Atlassian JIRA ticket is a project task, which has data required to describe the task (the summary, which is also the title of the task, the description and labels), task typing (as described before: task, feature and so on), prioritizing (blocker, critical, major, minor, trivial) and task assignment to developer. The task can also have due dates and users can set estimations on how much time it will take to solve the task. The time used is usually followed during solving the task as users log work done in the project.

While JIRA has a quite simple system for collecting ticket information, it has a few simple tools to organize workflow. The projects can have components that could be used to organize software into less complex sets. For example, a software project could be organized in the user interface, command-line interface, database backend or API-components. The components can also be organized by user roles. The components help to organize workflows especially when the software project has more than one developer.

JIRA can also be used in simple release version numbering management. Using the version numbering during development process help monitoring what has been done during a certain release cycle, if the tickets have information about which versions of the

software has certain bugs and in which version the bugs have been corrected. In fact, one can download release notes straight from JIRA and those notes can be used to give the required information to users about new software releases.

During the evaluation of Deveo, the team had tested the integrations with Atlassian HipChat, which is a workgroup chat service developed especially with software development projects in mind. The integration tests were done to evaluate if information created in repository management system could be used to help the development team to be more aware of what is happening in the development team. The tests were simple: every deployment to the test repository triggered a message to integrated chat group in HipChat.

The evaluation of Deveo revealed that the development team will benefit from the additional information created through the integrations. Also, because the Metropolia software development team is quite small, the additional information is not exhausting or distracting. Therefore, a choice was done that the development team would be starting to use group chat service that could be integrated with Deveo. However, the team was not totally happy with HipChat, so we decided to test also another developer orientated chat service. We could have used Skype for Business, which was used in Metropolia as the federated chat and video conferencing platform. But Skype for Business is not easy to integrate as for example, HipChat is. In HipChat any user with enough user permissions can create an integration to another service with the web user interface. Integration to Skype for Business must be done with service API.

While the development team were testing HipChat, the team had heard of another alternative to it: Slack. Slack Technologies had created a system which had already impressed some of our team members and we decided as a team to do a quick evaluation of Slack. Our team members were impressed with how well Slack could be used in the desktop, web browser and mobile environments. Also, the integrations were as easy to create as in HipChat and we could also find easy to use language libraries so that we could create simple prototypes for connecting software directly to Slack's chat channels. In fact, we created commonly usable software library extensions to the internally used logging library that could be used to stream log information to chat. The idea was that reacting to critical software errors could be made faster and more

accessible when developers would get error messages through group chat messages rather than through email.

While the team was testing Slack, it also thought about the costs of the chat system. Skype for Business is licenced very competitively through academic licensing, but it also requires servers and some features requires specified hardware, like video conferencing systems for meeting rooms. Testing Slack did not require any extra investing, because the team could do all required testing with freemium licence tier of the service.

5.3 The Importance of Documentation in Software Development

The next step in the development task was to focus on documentation. One of the biggest problems of big changes in the personnel of the development team was recurring brain drain: some knowledge was lost every time someone left the team.

Before the team decided to find suitable documentation methods for our development process, we acknowledged that software developers are the key to this change. On the other hand, there was a general lack of software developers while doing the reform. Even though we could easily understand that developers can find new jobs, the team had not emphasised the importance of documentation in software development enough, because we had situations where the only thing left of the software project was the software source code when the developer left us. This had to be changed because, in the long run, the situation on the documentation was intolerable.

The team had already started to use wiki platform to document meeting minutes of the weekly coding team meetings, and because it had good experiences on using wiki also in other development projects, it was logical to expand the role of the wiki to be the platform of the coding team documentation.

The created documentation was divided into sections that supported the needs of the development team. The team needed introductory material for possible new team members. The first days for a new developer are difficult if there are is not any pointers where to start learning. The introductory documentation contained pages for example on what is required to learn as a PHP developer in Metropolia.

The team also required documentation pages on development projects, if there were not larger documentation sets available for the projects. The minimum requirements for project pages were set and requirement set included the following information:

- Who is responsible for the data stored by the application? This is usually the administrator user of the data system.

- What is the programming language used and the required language version?

- The general information about software frameworks and libraries used in the project.

- The information about testing, staging and production environments, including information about data stores

- The information where source code is stored and how the source code can be accessed.

- The information about deployment methods to different environments

- The information about API-interfaces used or offered by the project

The documentation of the software project was required to be expanded to such level that any one of the team could continue development. It was also noted that sometimes developers themselves forget details of the project, so documentation written previously can help them in the future especially in the projects that are in the maintenance stage of the software lifecycle.

As the technical project documentation methods were getting better, the development team was also tasked the create better code documentation. To this point of time code documentation was done only using free form comment styles of development languages. However, most of the development languages have standardized or de-facto methods for comments. There is a big difference in the free form versus the formalized commenting. The free form comments usually only include the data developer thinks is needed to give enough information about the code. The formalized methods, like

Javadoc in Java or Phpdoc in PHP, are usually supported by IDE software. The best IDE software encourages developers to use correct code documentation style by giving feedback if documentation style is wrong and also uses code documentation to help the developer to use created code correctly by showing code documentation while the developer is writing code.

Every developer was tasked to update documentation, e.g. both project and code documentation. This meant, that if someone noticed errors or lack of information in the documentation, that person should take charge and fix the errors or create the documentation needed. Every team member has an equal responsibility to take care of the documentation.

In current web application development methods application programming interfaces (APIs) are commonly used to integrate different web-based applications together. If data is needed to be used collectively or if data is needed to be transferred between the applications APIs are central. There are different types of methods for connecting applications like web services using XML-based data payloads or RESTful interfaces using JSON data payloads. Also, people commonly write about open APIs, where data can be used more freely between the data system even over organizational barriers.

Even though there are open or free APIs, there is sometimes one final obstacle in using data interfaces, and again it is the documentation. Sometimes those who create interfaces for connecting data systems do not emphasize good and thorough technical documentation.

The deployment phase of the student information management system part of the before mentioned Peppi-project was closing in the autumn of the year 2016. The student information management system, like every other software systems in Peppi-project, is created with service-orientated architecture, where data manipulation through data interfaces and APIs are one of the main design principles. This puts emphasis on accessible, thorough and understandable API-documentation. In RESTful API’s there is software like Swagger, which gives good documentation tools, where API-documentation is created programmatically from source code and published as an interactive webpage. Properly configured Swagger installations also have live testing features, where the developer can test interface with web interface.

Even if there is good enough API documentation available the developers sometimes need pointers on where to start looking for the needed documentation. As the development team had some technical documentation about the API's in the student information management system and some of the API's were testable through Swagger, the only expansions for the documentation were those pointers: where to start looking for answers to technical challenges. This information was put into the wiki to help the developers. It is important to understand that the documentation is not done for today. The documentation is written to be used when people forget the details. In my experience, the source code is seldom passable as the only documentation of a software system.

5.4 From Traditional to Agile Software Development Process

After the development team's weekly meetings were a regular habit for the team and common documentation practices were set, the focus on reform was moved to getting more data from the processes the get things done more efficiently. The weekly meetings started to have a fixed structure, so every team member knew how they are supposed to give their input into the common knowledge about the situation in the development team. The meeting minutes were documented, and that data was usable for the team and even if some of the team members were unable to be present in the weekly meeting the minutes were done and readable by other team members. The meeting routines were accepted by the team, but at the same time, even semi-regular meeting schedule required someone to take care of continuity. Basically, somebody had to send invitations to team members.

As mentioned previously, the development team started to use JIRA ticketing system. JIRA is very flexible in organizing software projects and projects can have different workflow types. JIRA does also get regular updates to its features. Metropolia had a bit outdated JIRA in early 2017 on the on-premise JIRA installation compared to Atlassian cloud installation of the JIRA. For example, agile board like Kanban boards were missing Metropolia JIRA installation.

The development team was increasingly using JIRA for following what is happening in the development projects. Especially when there was more than one developer working

on the project it made sense to make tasks about everything that is done in the projects. At the same time, the team also wanted to follow ideas found in different agile project methods like Kanban board, but the ideas were modified to the needs of the development team.

The team noticed that it is possible to connect Confluence wiki with JIRA project management as both use same single sign-on system (Atlassian Crowd) and as both are Atlassian products, Atlassian has built direct connections between the products so that they are closely integrated. The project task data created in JIRA can be fetched to Confluence and different datasets can be displayed interactively in Confluence using filtered queries. This data fetching did not require any actual programming as everything was done with Confluence macros. Also, the datasets were organized as customized views which were “Kanban-like” or “agile minded”. We named this customize board “Kanile”, pun intended. With this naming, we recognized that as a team we have a need to agile in the development of software. We also recognized that we don’t have a need to use formalized method to be agile, we only need to find methods that help us to be what we wanted to be.

The development team’s agile board was simple. It had only four lanes: backlog, ready, in progress and done. The backlog lane was for the tasks that were not ready for development. The ready lane was for the tasks ready for development and the in-progress lane was for the task under development. The done lane was for the development tasks. With these lanes, the development team could start to follow what is being done to what and what kind of resources would be needed soon. The agile boards were also organized in two different ways: a view of all the tasks of the team and per project views of the tasks. Of course, the data used is only as good as the developer’s data inputs to JIRA, so if the developer does not use JIRA properly or in the agreed way. As we were not using JIRA agile boards, the tasks were categorized using labels in tasks when tasks were set in the backlog or ready lanes. If developer inadvertently did not use correct labels in the tasks, the tasks were not set in the right stage in the wiki board. And if the developer did not use JIRA to log tasks, then the development team did not get any usable data.

After the development team started to use JIRA for task tracking, the team got better information about what had been done and what is going to be done soon. After starting

to use the agile board, the data about the tasks got a bit more depth, but also the data was organized in a visual way so that it could be more easily internalized. The actual changes done to the process were small and easy to implement and the development team were mostly comfortable to implement them.

5.5 First Steps Towards DevOps

After the development team got more used to a bit more agile and documentation orientated development processes, the team noticed that there is much to be done to the process on the server-side of the software development process. First, some of the servers that were hosting web-based software were getting outdated, especially the servers hosting software developed in PHP language. Also, the Jenkins server, that the team used to use for continuous integrations, broke down, so there was some server work needed.

The server support team of Metropolia installed new servers for PHP software hosting. The support team installed production and testing environments and those servers were configured as similar as possible. The PHP runtime version was upgraded from 5.6 series to PHP7 which required significant updates to some software running on the servers. While doing the updates the development team run into the recurring problem of missing technical documentation on some of the oldest development projects: Either the information was lost or difficult to find, e.g. only stored into emails, or the source code was the only documentation. Luckily the software systems needing updates to be PHP7 compatible were not difficult to update without documentation.

Jenkins is a continuous integration and deployment software package, which is developed as open-source software. It is developed in Java, but it can be expanded with plugins to support almost any programming language so that automated testing and deployment can be done in connection to the development process.

The term Test-driven Development (TDD) contains ideology, which states that during software development while the developer writes the code the developer also must write code that tests the code. Agile Alliance defines TDD with the following process (Agile Alliance):

- write a "single" unit test describing an aspect of the program
- run the test, which should fail because the program lacks that feature
- write "just enough" code, the simplest possible, to make the test pass
- "refactor" the code until it conforms to the simplicity criteria
- repeat, "accumulating" unit tests over time

The basic idea is that you write the code and then test it using unit testing tools, which are usually programming language-specific tool packages, like Junit for Java and PHPUnit for PHP.

Some of the software testing frameworks enable user interface usage testing programmatically. For example, in PHP Codeception testing framework can be configured to run acceptance, functional and BDD (Behaviour Driven Development) testing against the development project. (Codeception, 2019) This means that developers can build automatic source code testing which can broaden the whole testing of the product. Developers deploy at least automatically tested product to the testing environment which is then tested by human testers.

At the start of the reform, the using of automatic testing in Metropolia development team was dependant of the development project age. If the project was old, it was more likely, that any automatic testing was not used. It was also noted that some of the old projects were almost impossible to convert totally to TDD paradigm. The developers were encouraged to use TDD as often as possible and unit tests were deemed necessary to new starting projects.

After the introduction of new toolchains and methods, it was time to start uniting different technologies and paradigms to common unified processes. At this moment, Jenkins templates for PHP and Java CI/CD toolchains were written and tested. The end target, building the continuous integration and continuous deployment working processes, was almost there. All that was needed was to get things done.

5.6 The Summary of Process Development Tasks

The main development tasks of the reform software development process in Metropolia IT were stated previously in this thesis. These development tasks have usually process and tool factors to be developed: Some of the tasks can be solved by changing how things are done, some must be solved with new tools. In the following table, I will introduce development tasks done at the end of my role in the reform process until summer 2017.

Table 2. The development tasks that were done in Metropolia IT reform

Development Task	**Process Actions or Process Development Done**	**New Tools or Software to Support Development Process Introduced**
Enhance teamwork	• Started weekly short review meetings with the development team. • Introduced agile working methods in development projects where suitable. • Streamlined developer communication between during development phase. • Started peer to peer code reviews between developers.	• Created “only for developers” wiki in current wiki-platform (Atlassian Confluence) • Introduced team communication tools and elected Slack as the platform

Create better documentation processes and tools	• Created guidelines for technical documentation, which must be done for in-house developed and supported software. • Created the process of how and where documentation documents are saved. • Created guidance for better in-code documentation using language-dependent standards like Javadoc, PHPDoc etc.	• Created technical documentation template documents. • Created the platform for saving documentation on the wiki (Atlassian Confluence) and document repositories (internal network drivers and Google Docs Team Drives).
Utilize testing procedures	• Started adaptation of unit testing using TDD-methodology and continuous integration.	• Installed and started using CI server Jenkins.
The change of the version control system	• Created common process on how, where and why source code is stored into organization owned version control.	• Installed and started using Git-based version control server as on-premise installation first using Deveo and then Gitlab CE.

Create deployment best practice and take it into use	• Created the CI process. • Planned the creation of the DevOps process.	• Created CI deployment templates for software projects for each programming language for CI-solution chosen (Git combined with Jenkins).
Better end-user support processes and tools	• Created a support plan for each of the software under on-going support.	• Begin using issue tracking software JIRA in the software development support process

All the groundwork of the reform was ready in spring 2017. At the end of the process in the summer of 2017, my role in Metropolia changed: I moved from IT development role to the data protection officer role. At that point, my role in this process reform decreased to an advisory position.

My colleagues in the Metropolia development team continued the reform and tuned their working processes as needed. They also had to change an integral software tool in the process, the source code repository software. Deveo was sold to Perforce (Anon, 2017) and Metropolia lost pricing advantage of the source code repository software. The development team chose to use Gitlab Community Edition (Gitlab CE), and the source code repository hosting was moved to the on-premise server. As all the source code repositories were Git-based, the migration of the services was done quickly. Gitlab also includes pipeline and deployment services, which means that if the development team wants to change CI/CD toolchain from Jenkins to Gitlab CE, that is possible to do. However, this requires building new toolchains.

The other service that was changed in the toolchain was a gradual movement from Slack to Microsoft Teams as the communication platform. Microsoft Teams was launch in

spring 2017 and early on it was informed that it will replace Skype for Business as Microsoft's business communication platform offering. In that time Metropolia used Skype for Business but Teams was enabled early on and when its features were compared with Slack's development team decided to start experimenting with Teams in autumn 2017. It had similar structure were workplace teams could create open or closed communication groups and it features the possibility to connect Teams to other software platforms using API.

At this point in the reform process, one of the last steps that the development team wanted to experiment with was platform were software is run. The development team wanted to have more in-depth control like build, install and control actual or virtual servers were software is being run. The development team decided to evaluate software-based infrastructure platforms like OpenStack, but by summer 2018 this was not implemented in the production systems. In fact, I think that Metropolia IT should move directly to container architectures in running self-developed software hosted in the on-premise datacentre. Without a more close connection between the software development team and technical infrastructure, i.e. where the actual software service is running, the Metropolia IT software development team will have difficulties in building and starting to use a complete DevOps platform.

6 Discussion and Conclusions

As this process development was done as a side project in the normal software development and process development workflow, the resources used, and time spent was spread over a long period. We also tried to empower every team member to the process, so everyone's opinion mattered, and the team made decisions together.

After I moved to another position in Metropolia IT in autumn 2017, I paused to reflect the whole development process as an insider to the process. The greatest success of the reform was creating some structure in the working methods of the development team. Also, better documentation and communication practices helped the team to do better work together and in the long run, I am sure that these processes help when other personnel changes happen in the development team. The development team made solid choices about the software tools: Atlassian has developed tools that help to document and track projects: Confluence and Jira are very easy to connect to make processes better. Also, Git-based source code management is both the de-facto choice and the best choice in source code management for modern software development projects. Losing Deveo as code repository platform was a setback for the reform process, but Gitlab CE was a good substitution. However, one software tool choice was problematic: Jenkins. In my experience, Jenkins requires constant, almost daily, maintenance to keep the software up to date and secure and creating a continuous integration and delivery pipelines are sometimes very troublesome and laborious to build. Gitlab CE has inbuilt CI/CD pipelines and it is also much simpler to maintain so, in my opinion, Metropolia software development team should examine those and after evaluation probably should move into using CI/CD pipelines in Gitlab CE and abandon Jenkins. So, in the long run, Gitlab CE could replace two main software components in the development pipeline.

The other not so good software tool choice was Slack, but that is part of the reality were public cloud-based services are being developed rapidly. When testing of Slack started, Microsoft's business communication platform of choice was Skype for Business. However, Slack was disruptive and showed Microsoft and everybody else that businesses and organizations were ready for cloud-based communication platform where more control was given to end-user. In Slack end-user can create independently communication groups and invite users freely without organization barriers, if a user is the owner of the communication group. Microsoft created a new cloud-based

communication platform to compete Slack and other similar new platforms, with similar ideology on openness, but with possibilities to have more strict control on who is using the platform. This service was named as Microsoft Teams and especially in the situation where the organization has Microsoft Office365 cloud platform in its service catalogue, it is practical to use Microsoft Teams if the organization wants to use the federated user account to access Teams. If we think about a scenario, where a company has internal and external communication needs, in internal communication it is very important to have control on access to the communication groups. The end-users even might use accounts that are not tight to the work email or use accounts that are tied to consumer email services creating so-called “shadow computing” environment. In-fact Slack was almost in like that sense because using Slack started from informal experimentation, so it was just about necessary to stop using it and move into a more controllable platform.

For the actual reform process, it was important to have a clear goal: to create the best software development process to the development team was like that, a good clear goal, however at the same time it was actually helpful that Metropolia’s strategy begun to include PDCA methodology as a method to continuously develop university functions and processes. If we have tried to do all the development work in one project of a waterfall style, then we would probably have made more wrong decisions. The step by step approach in process development gave precious feedback throughout the process and wrong decisions were corrected. In-fact the development process follows quite closely plan-do-check-act cycle: plan action before doing and check what went wrong. And then if required, act on findings. We were not so conscious at the beginning of the process that we are using some fancy method, that has a name. For software developers testing what has been created is second nature. You should always test when you write code, always and all the time. You can use technical methods or toolset, but something or someone always must test the product before shipping it to the customer. Therefore, it was a natural choice for reform project to make decisions and preferably fail as soon as possible, because failures that take time to find and rectify cost resources. If the development team makes bad choices on software tools or methods, it will also have a psychological impact on reducing job satisfaction.

As stated before, process development requires resources, and that’s why sometimes there is the temptation to leave everything as is, because “everything is ok and works”. But even in this process, I noted that in two years’ time it was possible to gather direct

or in-direct technical debt in the Metropolia software development team point of view. The software development tools are usually either developed or abandoned. What happened to team communication platforms are a good example of a constantly changing IT field. Slack disrupted the market creating a new type of team communication platform, or this was a common belief, that they created a totally new product category. But there was already IRC, which was developed in the late 1980s and offered the possibility to create closed chat rooms for teams and IRC was once de-facto place for developer team chatting. Atlassian developed HipChat before Slack and HipChat were one of the first web-based communication platforms for developer teams. But Slack made the breakthrough to the mainstream when it found footing outside of the software development world in cross-organization co-operation and when it started to offer connectivity between it and business software platforms. In the end, Microsoft decided it must enter the market with up to date team communication platform with Team. Eventually, Hipchat was sold to Slack. (Redfern, 2018) This whole combination of technologies and service providers are a good example that everything changes. HipChat offered connection API. If the software process developer made a commitment to using HipChat without any backup plan, the investment used to create technical processes using HipChat API's would have had a very short repayment period. But if tool platform has vanished, and you are dependent of platform services and you can find compensatory service, if you don't move services and create service processes in the new platform, you are creating technical debt. In the constantly changing infrastructure world, it is vital to have options and important to understand the impacts of the choices you have made because in two years' time everything has changed again.

Another viewpoint on technical debt is the skills of the developers: if developers do not have opportunities to develop their skills, keep their knowledge on development methods and tools up to date, then they have skills debt that may contribute to the accumulation of technical debt. In my opinion, it is important for development organizations to invest in continuous education of the workforce, and this need is highlighted in software development because the field of software development is in constant change.

While I was working as the data protection officer a noted in the spring 2018 all the needed tools and development methods were still available to be used by the Metropolia IT software development team. The software tools like Gitlab were maintained and up to date, however, the other methods, like weekly meetings, were somewhat difficult to

continue and sometimes forgotten. More than one team member stated that it was easy to take part in weekly meetings with simple structure when responsibility for organizing the meetings was clearly someone's task and the development team concentrated just on the development tasks. When I was guiding the development team to have regular meetings and actively encourage them to open communication, I also tried to make them understand that the responsibility of the development process is common, so if someone is taken out of the equation, then the team should fill the void somehow. It is easy to state that the team should work like this like automatically react and adapt changes but, almost every team requires some leadership and teams does not always adapt automatically. The change processes sometimes require someone to be the change advocate, with set and named responsibilities and goals. If the change advocate has started the process by persons own initiative and if that person stops developing, then the team might stop developing if responsibilities for change are not set and named to another person, especially if the team has no time or no interest for self-development. The development teams can sometimes be content on achievements they have reached and that can also stop them from seeing the need for self-development. In that point change advocates are needed to keep continuous improvement alive.

In autumn 2018 I moved to another job with another employer. My daily connection to Metropolia IT development team was cut, but I have stayed in contact with them trying to encourage them to keep working with constant and tried methods, by as an outsider my influence on what they would and should do to their software development process is not unreasonably not existent. I can only hope that they do not encounter similar problems that I encountered when I did not have any team colleagues after Christmas holidays because that would indicate that the organization has forgotten what I learned on that hardship.

At my new employer, I was given a glimpse into how a multi-vendor software development team deploys software into cloud-based infrastructure. The service this development team supports is a widely used public sector software serving both individual citizens and public sector institutions. The service has requirements to offer constant service and has peak service periods throughout the year. The peak service periods require scalability on the service. The whole development process is very close to the by-the-book DevOps development model. The infrastructure architecture is developed and maintained as a code. The software development toolchains include

continuous integration and deployment pipelines. The developers are close to the infrastructure and they can give input to infrastructure development when at the same time infrastructure development is done by specialists in that field. The whole development and infrastructure architecture are constantly developed, just because the cloud platform used is changing all the time. The changes in the cloud platform are done for a better technology solution, better data security and for lowering the total cost to service users. If my current employer would not require our vendors to keep their knowledge up to date and to develop our infrastructure, we would gather skills and technical debt and in long run the service platform would have to be shut down, because technical progress would have made our software outdated and possibly impossible to run in the current architecture.

This is the parallel I can draw between my current employers and Metropolia IT's needs for constant development of the software development processes. Both have the change as common constant and role and targets of change management in development teams are important. From Metropolia IT point of view, if development team management loses focus, or they do not give enough resources to the development team to develop working methods and tools, then I would consider if it is wise to keep in-house development team. My reasoning on this is, that if the organization is not willing to invest in or direct the development team to keep up to date, then management should get software development services from organizations that are willing to do that. Usually, organizations that are software development companies, must keep their workforce knowledgeable. So, in my opinion, Metropolia IT has a choice to make; either keep investing in the continuous development of the software development team or change direction and adapt to a new situation. If they keep in house software development, they should have a person that did a similar software development process development that I have described in this thesis.

Another area that needs constant attention is documentation in software development. This need can be expanded in entire IT development because, throughout my career in IT, I have noticed many times, that documentation is difficult. When you have no changes in the workforce, you have all the knowledge you need because you have people who you can ask specified questions about the problem you have. Sometimes this information is found on common documents, many times data is in emails and quite often data and information derived from the data is only in the form of silent data. Only some of the

people really know what is needed and this silent data is seldom written down as documentation. When people with silent data leaves the organization is left with a knowledge gap, and this gap very often must be crossed somehow. Sometimes organizations must rebuild some specified knowledge many times because of lacking documentation. Sometimes organizations must abandon old technologies and adapt new because it is too expensive to rebuild knowledge which has a lack of documentation. My conclusion is that organizations like Metropolia IT must invest continuously on documentation, both in where and how documentation is done and maintained. Also, the documentation platforms must be maintained and one of the critical factors in choosing of documentation platform is how portable documentation is. Documentation can be done with different technologies but if standards are common file formats are used, there should be methods for accessing documentation for a long time.

At the end of this thesis, I will return to the first citation of this text: “An optimal software development process is regarded as being dependant on the situational characteristics of individual software development settings”. (Clarke and O’Connor, 2012) After going through the reform of the Metropolia IT software development process I completely agree with this. Clarke and Connor lists a long list of factors and sub-factors influencing on the software development process and in my experience this is true: when building an optimal software development process for a named team, you can learn from methods and experiences from other teams, but you cannot copy a process from another team without taking into account the characteristics of the team and operating environment of the team you are creating the process for. In a sense, it could be argued, that because everything is changing all the time in software development, one should not create the development process that is finalized but is constantly evolving.

References

Agile Alliance. *Glossary - TDD*. [Online]. Available at: https://www.agilealliance.org/glossary/tdd/ [Accessed 28 October 2019].

Ahtokivi, I. (2018). *Korkeakoulujen rahoitus kaipaa vakautta – ei tempoilua*. [Online]. Available at: https://www.verkkouutiset.fi/korkeakoulujen-rahoitus-kaipaa-vakautta-ei-tempoilua/ [Accessed 29 October 2019].

Ammattikorkeakoulujen rehtorineuvosto Arene ry. (2019). *Arene, Unifi, Samok, SYL: Korkeakoulutuksen laajentamiseen tarvitaan vuoteen 2030 ulottuva rahoitusohjelma*. [Online]. Available at: http://www.arene.fi/ajankohtaista/arene-unifi-samok-syl-korkeakoulutuksen-laajentamiseen-tarvitaan-vuoteen-2030-ulottuva-rahoitusohjelma/ [Accessed 20 September 2019].

Burns, A. (2005). Action research: an evolving paradigm? *Language Teaching*, 38 (2), pp.57–74. [Online]. Available at: doi:10.1017/S0261444805002661.

Carr, W. (2006). *Philosophy, Methodology and Action Research*. p.15.

Clarke, P. and O'Connor, R. V. (2012). The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54 (5), pp.433–447. [Online]. Available at: doi:10.1016/j.infsof.2011.12.003.

Codeception. (2019). *Introduction*. [Online]. Available at: https://codeception.com/docs/01-Introduction [Accessed 28 October 2019].

Costello, P. J. M. (2003). *Action Research*, Continuum Research Methods. Bloomsbury Academic. [Online]. Available at: https://books.google.fi/books?id=epJYsvt8BXAC.

Debois, P. (2008). Agile infrastructure and operations: how infra-gile are you? In: *Agile, 2008. AGILE'08. Conference*. 2008. IEEE. pp.202–207. [Online]. Available at: http://ieeexplore.ieee.org/abstract/document/4599477/ [Accessed 24 March 2017].

Denning, S. (2015). *Agile: The World's Most Popular Innovation Engine*. [Online]. Available at: http://www.forbes.com/sites/stevedenning/2015/07/23/the-worlds-most-popular-innovation-engine/#4a4b2f812d4c [Accessed 24 March 2017].

DevOpsDays. (2009). *DevOpsDays Ghent 2009*. [Online]. Available at: http://www.devopsdays.org/events/2009-ghent/ [Accessed 24 March 2017].

Fitzgerald, B. and Stol, K.-J. (2015). Continuous software engineering: A roadmap and agenda. *The Journal of Systems and Software*, 123, pp.176–189. [Online]. Available at: doi:10.1016/j.jss.2015.06.063.

Hartio, I. (2016). Tuhansia työpaikkoja leikkuriin – korkeakoulujen rahoitusta on leikattu yli 280 miljoonaa euroa neljässä vuodessa. [Online]. 16 January. Available at: http://www.ess.fi/uutiset/kotimaa/2016/01/16/tuhansia-tyopaikkoja-leikkuriin--korkeakoulujen-rahoitusta-on-leikattu-yli-280-miljoonaa-euroa-neljassa-vuodessa [Accessed 24 March 2017].

Huo, M. et al. (2004). Software quality and agile methods. In: *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*. 2004. IEEE. pp.520–525. [Online]. Available at: http://ieeexplore.ieee.org/abstract/document/1342889/ [Accessed 24 March 2017].

Hüttermann, M. (2012). *DevOps for developers*, The expert's voice in Web development. New York: Apress : Distributed to the book trade worldwide by Springer Science+Business Media New York.

Johnson, G., Scholes, K. and Whittington, R. (2009). *Fundamentals of Strategy*. Essex: Pearson Education Limited. [Accessed 30 October 2019].

Kent Beck et al. (2001). *Manifesto for Agile Software Development*. [Online]. Available at: http://agilemanifesto.org [Accessed 24 March 2017].

Konkola, R. (2013). *Strategia auttaa arjen valinnoissa - Metropolian strategia 25.10.2013*. [Online]. Available at: https://tuubi.metropolia.fi/portal/group/tuubi/tiedotteet1?p_p_id=eduixannouncement_WAR_eduixannouncementportlet&p_p_lifecycle=0&p_p_state=normal&p_p_mode=view&p_p_col_id=column-2&p_p_col_count=1&_eduixannouncement_WAR_eduixannouncementportlet_advancedSearch=false&_eduixannouncement_WAR_eduixannouncementportlet_keywords=&_eduixannouncement_WAR_eduixannouncementportlet_delta=20&_eduixannouncement_WAR_eduixannouncementportlet_action=view&_eduixannouncement_WAR_eduixannouncementportlet_andOperator=true&_eduixannouncement_WAR_eduixannouncementportlet_articleId=2352664&cur=1 [Accessed 24 March 2017].

Koski, P. and Kelo, M. (2019). *Toimintatutkimus menetelmänä*. [Online]. Available at: https://blogit.metropolia.fi/masterminds/2019/09/30/toimintatutkimus-menetelmana/ [Accessed 31 October 2019].

Lewin, K. (1946). Action Research and Minority Problems. *Journal of Social Issues*, 2 (4), pp.34–46.

Mauno, P. (2019). Korkeakoulut eivät saa euroakaan uusiin aloituspaikkoihin - '"Oppilaskohtainen rahoitus laskee, kun sen hallitusohjelman mukaan piti nousta"', sanoo työmarkkinajohtaja. [Online]. 24 September. Available at: https://www.kainuunsanomat.fi/artikkeli/korkeakoulut-eivat-saa-euroakaan-uusiin-aloituspaikkoihin-oppilaskohtainen-rahoitus-laskee-kun-sen-hallitusohjelman-mukaan-piti-nousta-sanoo-tyomarkkinajohtaja-170315125/ [Accessed 29 October 2019].

Metropolia Ammattikorkeakoulu Oy. (2013). *Yt-menettelyt päätökseen: irtisanottavia lopulta 26*. [Online]. Available at: http://www.metropolia.fi/ajankohtaista/uutiset/?tx_ttnews[tt_news]=4356&cHash=e16811088fbfcbf622163209514864f8 [Accessed 24 March 2017].

Metropolia Ammattikorkeakoulu Oy. (2014). *Vuosikertomus 2013*. [Online]. Available at: http://vuosikertomus.metropolia.fi/2013/fi.html [Accessed 29 October 2019].

Metropolia Ammattikorkeakoulu Oy. (2015). *Vuosikatsaus 2014*. [Online]. Available at: https://www.metropolia.fi/tietoa-metropoliasta/vuosikatsaus-2014/ [Accessed 29 October 2019].

Metropolia Ammattikorkeakoulu Oy. (2016a). *Metropolian yt-neuvottelujen tulos: irtisanomisilta vältytään*. [Online]. Available at: https://www.metropolia.fi/ajankohtaista/uutiset/?tx_ttnews%5Btt_news%5D=5379&cHash=3c134393afe3845745acd24ef7a8083b [Accessed 29 October 2019].

Metropolia Ammattikorkeakoulu Oy. (2016b). *Vuosikatsaus 2015*. [Online]. Available at: https://www.metropolia.fi/tietoa-metropoliasta/vuosikatsaus-2015/ [Accessed 29 October 2019].

Metropolia Ammattikorkeakoulu Oy. (2017). *Vuosikatsaus 2016*. [Online]. Available at: https://www.metropolia.fi/tietoa-metropoliasta/vuosikatsaus-2016/ [Accessed 29 October 2019].

Metropolia Ammattikorkeakoulu Oy. (2018a). *Vuosikatsaus 2017*. [Online]. Available at: https://www.metropolia.fi/tietoa-metropoliasta/vuosikatsaus-2017/ [Accessed 29 October 2019].

Metropolia Ammattikorkeakoulu Oy. (2018b). *Yt-neuvottelut päättyivät: Metropolia irtisanoo enintään 30 työntekijää*. [Online]. Available at: https://www.metropolia.fi/ajankohtaista/uutiset/?tx_ttnews%5Btt_news%5D=6275&cHash=bc6b760bcef492b21b2c616b05c54e73 [Accessed 29 October 2019].

Metropolia Ammattikorkeakoulu Oy. (2019a). *Tilinpäätös 2018: Metropolian tulos parani*. [Online]. Available at: https://www.metropolia.fi/ajankohtaista/uutiset/?tx_ttnews%5Btt_news%5D=6566&cHash=bb769316720590fda94523f7ccec6ff5 [Accessed 28 October 2019].

Metropolia Ammattikorkeakoulu Oy. (2019b). *Vuosikatsaus 2018*. [Online]. Available at: https://www.metropolia.fi/tietoa-metropoliasta/vuosikatsaus-2018/ [Accessed 29 October 2019].

Metropolia University of Applied Sciences. (2013). *Metropolia Strategy 2013-2016 - a leaflet*. [Online]. Available at: https://oma.metropolia.fi [Accessed 10 August 2019].

Metropolia University of Applied Sciences. (2016). *Metropolia's Strategy 2020*. [Online]. Available at: https://oma.metropolia.fi [Accessed 10 August 2019].

Metropolia University of Applied Sciences. (2019). *Metropolia's network and IT services down during October 11-15, 2019*. [Online]. Available at: https://www.metropolia.fi/en/about-us/news-and-events/?tx_ttnews%5Btt_news%5D=6712&cHash=339dbab5ce31529e2987cdd2071b3132 [Accessed 29 October 2019].

Ministry of Education and Culture. (2019). *Minister of Education Andersson and Minister of Science and Culture Kosonen: Government invests in knowledge and education*. [Online]. Available at: https://minedu.fi/artikkeli/-/asset_publisher/opetusministeri-andersson-ja-tiede-ja-kulttuuriministeri-kosonen-hallitus-panostaa-koulutukseen-ja-osaamiseen?_101_INSTANCE_vnXMrwrx9pG9_languageId=en_US [Accessed 29 October 2019].

Ministry of Education and Culture. *Higher education institutions, science agencies, research institutes and other public research organisations*. [Online]. Available at: https://minedu.fi/en/heis-and-science-agencies [Accessed 28 October 2019].

Nieminen, I.-M. (2017). *Professori pelkää: Lukukausimaksut pian myös suomalaisille*. [Online]. Available at: http://yle.fi/uutiset/professori_pelkaa_lukukausimaksut_pian_myos_suomalaisille/8638915 [Accessed 24 March 2017].

Opetus- ja Kulttuuriministeriö. (2013). *Ammattikorkeakoulujen rahoitusta muutetaan tuloksiin perustuvaksi*. [Online]. Available at: http://www.minedu.fi/OPM/Tiedotteet/2013/11/amk_rahoitus.html?lang=fi [Accessed 4 June 2015].

Opetus- ja Kulttuuriministeriö. (2019). *Korkeakouluille uusi rahoitusmalli*. [Online]. Available at: https://minedu.fi/artikkeli/-/asset_publisher/korkeakouluille-uusi-rahoitusmalli [Accessed 29 October 2019].

Peppi-konsortio. *Jäsenet*. [Online]. Available at: http://www.peppi-konsortio.fi/jasenet/ [Accessed 29 October 2019].

Redfern, J. (2018). *Announcing our new partnership with Slack*. [Online]. Available at: https://www.atlassian.com/blog/announcements/new-atlassian-slack-partnership [Accessed 28 October 2019].

Rintamäki, T. (2010). *Tietohallinnon ja tietohallintopalvelujen strategia 2010-14*. Metropolia Ammattikorkeakoulu Oy. [Online]. Available at: https://oma.metropolia.fi/delegate/download_workspace_attachment/483685/THstrategial10-12-01.docx [Accessed 24 March 2017].

Rissanen, R. and Ammattikorkeakoulujen rehtorineuvosto Arene ry. (2017). *Ammattikorkeakoulujen rahoituskehitys 2012* -. Eduskunta Sivistysvaliokunta. [Online]. Available at: http://www.arene.fi/wp-content/uploads/Lausunnot/2017/Ammattikorkeakoulujen-rahoituskehitys-ja-julkisen-talouden-suunnitelma_2018-2021_Arene-ry.pdf [Accessed 29 October 2019].

Schwaber, K. and Sutherland, J. (2017). *The Scrum Guide*. [Online]. Available at: https://scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.

Scrum Alliance. (2019). *Certfied ScrumMaster*. [Online]. Available at: https://www.scrumalliance.org/get-certified/scrum-master-track/certified-scrummaster [Accessed 1 November 2019].

Statistics Finland. (2019a). *Employment grew considerably in 2018*. [Online]. Available at: http://www.stat.fi/til/tyti/2018/13/tyti_2018_13_2019-04-11_tie_001_en.html [Accessed 29 October 2019].

Statistics Finland. (2019b). *Gross domestic product (GDP) 1975-2018*. [Online]. Available at: http://www.stat.fi/til/vtp/2018/vtp_2018_2019-06-20_tau_001_en.html [Accessed 29 October 2019].

Statistics Finland. (2019c). *Tax ratio, 1976–2018*. [Online]. Available at: http://www.stat.fi/til/vermak/2018/vermak_2018_2019-09-20_kuv_001_en.html [Accessed 29 October 2019].

Suojanen, U. (2004). *Toimintatutkimus ammatillisen kehittymisen välineenä*. [Online]. Available at: https://metodix.fi/2014/05/19/suojanen-toimintatutkimus/ [Accessed 31 October 2019].

The PHP Group. *What is PHP?* [Online]. Available at: https://www.php.net/manual/en/intro-whatis.php [Accessed 28 October 2019].

(2016). *Google Trends search using search terms "git", "svn", "subversion" and "mercurial".* [Online]. Available at: https://www.google.com/trends/explore#q=git%2C%20svn%2C%20subversion%2C%20mercurial&cmpt=q&tz=Etc%2FGMT-2 [Accessed 24 March 2016].

(2017). *Welcome to Perforce*. [Online]. Available at: https://www.perforce.com/deveo-customer-info [Accessed 28 October 2019].