MASTER'S THESIS

Zefektívnenie procesov vývojového tímu v startupovom prostredí na základe projekt manažérských a agílnych metodík

Process streamlining of a software development team in a startup based on project management and agile methodologies

DEGREE PROGRAM

Innovation Project Management

SUPERVISOR

Ing. Petr Fanta, Ph.D.

BUI THUY

HANH MGR.

2021

BUI THUY, Hanh. *Process streamlining of a software development team in a startup based on project management and agile methodologies*. Praha: ČVUT 2021. Diplomová práce. České vysoké učení technické v Praze, Masarykův ústav vyšších studií.

MASARYKŮV ÚSTAV
VYŠŠÍCH STUDIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bui Thuy** Jméno: **Hanh** Osobní číslo: **487679**

Fakulta/ústav: **Masarykův ústav vyšších studií**

Zadávající katedra/ústav: **Institut manažerských studií**

Studijní program: **Projektové řízení inovací**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Zefektívnenie procesov vývojového tímu v startupovom prostredí na základe projekt manažérských a agílnych metodík

Název diplomové práce anglicky:

Process streamlining of a software development team in a startup based on project management and agile methodologies

Pokyny pro vypracování:

The objective of the proposed thesis is to design and apply improvements to the processes of a dysfunctional software development team. The main value of this case study is anaysing a specific informal startup environment and suggesting adjustments based on software development and best practices in order to utilize and create an effective work setup. The study also aims to normalize custom adjustments to rigorous methodologies in order to effectively apply them to each company needs. It will attempt to do so in following chapters: 1. Introduction 2. Literature 3. Software Development in Theory and Methodologies 4. Startup Introduction 5. Analysis of Current Situation 6. Suggestion of Improvements 7. Implementation and Results 8. Implications 9. Concusion

Seznam doporučené literatury:

(1)Kelly, Allan. 2008. Changing Software Development : Learning to Become Agile. New York: John Wiley & Sons, Incorporated. Accessed October 20, 2020. ProQuest Ebook Central.(2)Koch, Alan, and Koch, Alan S. 2004. Agile Software Development : Evaluating the Methods for Your Organization. Norwood: Artech House. Accessed October 20, 2020. ProQuest Ebook Central.(3)Harned, David. 2018. Hands-On Agile Software Development with JIRA : Design and Manage Software Projects Using the Agile Meth

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Petr Fanta, Ph.D., institut manažerských studií MÚ

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **25.01.2021** Termín odevzdání diplomové práce: **20.08.2021**

Platnost zadání diplomové práce: **19.09.2021**

Ing. Petr Fanta, Ph.D.	Ing. Dagmar Skokanová, Ph.D.	prof. PhDr. Vladimíra Dvořáková, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Podpis studentky

Datum převzetí zadání

Declaration

I hereby declare that the presented thesis is my own work and that I have correctly cited all sources of information which are recorded in the attached List of Used Literature. I do not acknowledge any significant reason to prevent its accessing stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended.

In Prague on August 20, 2021 Signature:

Acknowledgment

I would like to express my gratitude for the support and help during the writing and organization of the thesis to my supervisor Ing. Petr Fanta, Ph.D. and to the Student's Department of Masaryk Institute of Advanced Studies.

Abstrakt

Software proces je efektivním nástrojem pro softwarové firmy na zodpovězení citlivých prvků, které softwarová produkce obsahuje. Zároveň minimalizuje risky špatně provedených kroků v procesu, které jsou hlavně ve softwarové produkci značně drahé. Software proces je o to víc příznačný v dynamickém prostředí startupu orientovaném na risk, které se snaží nereflektovat omezené zdroje na kvalitě produktu. Někteří autoři věří, že metodika zlepšení softwarového procesu je příliš rigidní pro malé podniky s omezenými zdroji. Tato diplomová práce kombinuje etablované metody zlepšení softwarového procesu, které zajišťují metodický rámec a metriky evaluace, s moderními trendy softwarového vývoje – agilními metodikami tak, aby zajistila jejich aplikaci na model startupu. Startupový model organizace, které je rovněž stoupajícím trendem softwarových organizací. Navíc tato práce nabízí podrobný popis kroků v procesu softwarového zlepšení, které může sloužit pro další implementace.

Klíčová slova

Software proces, zlepšení softwarového procesu, agilní metodika, startup

Abstract

Software process is an effective tool for software companies to answer to delicacies of the software production while minimizing the costly risks badly executed steps of software production result in. Especially in dynamic and risk oriented companies such as startups, who are operating in specific, limited conditions while trying not to reflect them on the product's quality. Software process improvement a method which can provide startups with stability and process' efficiency. Many argues that software process improvement is too rigid for small companies with limited resources. This thesis, however, combines established software process improvement methods, which provide framework and evaluation metrics, with modern trends in software development – agile methodology in the way it can be applied an effective application on a startup. Startup model of company as it is also an emerging, rising trend in software companies. On the top of that, this thesis provides a step by step process improvement using the aforementioned tools, which can serve as a guideline for further implementations.

Key words

Software process, software process improvement, agile, startup

Contents

Introduction

With the technological advancement in today's society, software product has become an inseparable part of corporate organizations and the progression of software study has been a dynamic but frequent trend in the academic field. Quality of the software production became a focus point of not only software companies that try to bring the biggest value to their customers, but also in the area of studies and researches. The specific nature of a software production requires specific methods where traditional manufacturing models cannot be applied. The creation of a software is an intellectual, creative process requiring engineering skills and imagination. This delicate nature of software development means that many variables can positively, but especially negatively impact the outcome of the software production. In order to mitigate the negative outcome and to leverage on positives, there were multiple software process models crated to provide software companies guideline in producing the product. This field has been constitutionalized as Software Process Improvement.

` While the topic of development methodologies, especially agile, is heavily studied from both theoretical and practical application, there are usually missing components of the studies such as framework and metrics to evaluate when implementing one. The proposed thesis aims to address this gap and contribute to the case studies on software process transformation. It seeks to do so by using the combination of established models, but also tailored and customized models in a specific environment of a startup. What this thesis attempts to answer in comparison to its counterparts, is not just how to improve software processes – e.g. application of agile methodologies, but also why and what is being improved. More specifically, the presented thesis will consider several models in the field of software process improvement and software process methodologies, in order to create a tailored model best suited for startup companies.

The thesis is divided into two parts. The first, theoretical one, focuses on defining key concepts used throughout the research. It starts with providing definitions and classifications on startup companies and their specifications, which later influence decision making in tools used in the research. The next part continues with definition of software process and various models of processes which are widely used. Integral part to a software process is individuals, and software teams. This is reflected in the next subchapter. Finally, software process improvement is conceptualized as a central focus of this thesis. Software process improvement is then reviewed from the point of current research and genesis of the field. The second chapter is important to highlight the background behind choosing the software process improvement model for the thesis.

The second part of the presented thesis is a practical research conducted on the chosen startup. It opens with providing contextual framework of the researched startup which played an integral role in the research shaping. Next, the methodology using multiple models of software process improvement, and project and agile methodologies is synthetized and there are four hypotheses conducted.

A detailed guideline throughout each step of the chosen model is laid out in the rest of the chapters, among the improvement plan and revised process model as a result. Impacts on the startup's organization are also described. Finally, the thesis concludes with interpretation of the researched data and answers the researched questions asked when proving the defined hypotheses of the thesis.

THEORETICAL PART

1 THEORETICAL BACKGROUND

In order to construct a theoretical framework and describe methodology used in the research of this thesis, it is important to start with clear definition of key terms used throughout the work. All concepts laid out in the following chapter were either subject of the research or played important role in decision making during the research.

1.1 Startup company

A *startup company* is a term spurring from the entrepreneurship activities. One of the most prominent authors in the field of startups, Steve Blank, defines startup as a "temporary organization in search of a scalable, repeatable, profitable business model."[1] It is not just a smaller version of a large organization and should not be studied nor led this way. In another definition, startups are considered starting points for young companies that aim to "develop a unique product or service, bring it to market and make it irresistible and irreplaceable for customers."[2] Startups operate in high-paced environments of innovation as they wish to create yet not existing goods and services and fill the hole in the market for unaddressed needs of customers. They are known as "disruptors".[3]

1.1.1 Startups classification

Blank classifies startups into 5 categories based on their scalability, origin and potential.

A) Small Business Entrepreneurships
are startups that were created in order to answer one specific need, usually one of founder's. These are normally service-oriented businesses like gas stations, dry cleaners and do not have ambition to become more sophisticated companies.

B) Scalable Startups
Majority of technology startups are defined as scalable. The vision of these startups is to change the world and transforming the startup into a large company of millions of sales. These startups require large investments in order to scale to these ambitions.

C) Buyable Startups
were created to be sold. Goal of these startups is to create and establish product on the market in order to raise their value and profit from the sale to a buyer, usually larger companies.

D) Large Business Entrepreneurships
When large established companies experiment with disruptive product, innovations or another business models, they fall into this category.

---

[1] Blank, Steve, and Bob Dorf. 2012. *The Startup Owner's Manual: The Step-By-Step Guide for Building a Great Company*. California: K & S Ranch, Inc., p. 16.
[2] Baldridge, Rebecca, and Benjamin Carry. 2021. "What is a Startup?" Forbes. https://www.forbes.com/advisor/investing/what-is-a-startup/#544a2a9a4c63.
[3] Ibid.

E) Social Entrepreneurs
are created to change the world using nonprofit innovation model. Rather than profits, social entrepreneurs seek solution in health, water, environment, agriculture, etc.[4]

1.1.2 Startup's characteristics

Approaches in startup business model developments, management and internal organization are accustomed to the specifications and difficulties startups face in comparison to other organizations. Startups are temporary organizations that wish to stop being startups and start being mature companies. Two very important variables in that process are time and money, and both tend to be limited in case of startups.

The success of startups depends on their *financing*. Most startups start with founder's own capital and are self-funded. From there, startups tend to proceed in one of these financing models:

Bootstrapping
They try to continue using founder's personal savings in what is called *bootstrapping.* Bootstrapping is financing a startup with limited funds, where through hard work, confidence, risk-tolerance and determination the startup aims to become self-sustaining using sales profit from customers.[5]

Angel investor
Another option for startups is to seek investment from so called angel investors. These invest in startups in their early stages in exchange with agreed portion of equity ownership interest. Other than financial resources, angel investors usually provide startups with consultation, partnership and customer network, contacts to potential employees, lawyers, accountants, etc.[6]

Venture capital
This type of financing is provided by established investors, investment companies or other financial institutions. It is hard to obtain as investors are looking for startups with long-term potential and high growth. The venture process is very time consuming, as it consists of several negotiations and documents/contracts preparations. Venture investors also expect a significant amount of return, not only in forms of finances, but also in form of decision-making, rights to sell, approval vetoes, etc.[7]

Crowdfunding
is a quite new tactic to raise funds through multiple sources. These sources are found on crowdfunding websites, which help startups to start their campaigns and promote its products and/or services. The more compelling or inspiring story behind the startup's launch, the bigger chance of raising the funds.[8]

---

[4] Blank, Steve, and Bob Dorf. 2012. *The Startup Owner's Manual,* p. 16-18.
[5] Kenton, Will. 2020. "Bootstrapping." Investopedia. https://www.investopedia.com/terms/b/bootstrap.asp.
[6] Harroch, Richard D., and Mike Sullivan. 2019. "Startup Financing: 5 Key Funding Options For Your Company." Forbes. https://www.forbes.com/sites/allbusiness/2019/12/22/startup-financing-key-options/.
[7] Hayes, Adam. 2021. "Venture capital." Investopedia. https://www.investopedia.com/terms/v/venturecapital.asp.
[8] Harroch, Richard D., and Mike Sullivan. 2019. "Startup Financing: 5 Key Funding Options For Your Company."

Concept of *time in startups* is important in two dimensions. First one is time spent on producing the product/services. The amount of time startups have is dependent on the financial resources and financing model of a startup. If time spent on production exceeds resources, product can never launch, and startups fail before even entering the market. Second dimension of time in startups is product launch on its own. Startups need to understand when is the right time to launch the product while also surpassing their potential competition.[9] As mentioned earlier, startups are disruptors. However, there is no assurance some other startup does not have the same idea in disrupting existing markets with similar, or even same product/services. Startups need to be quick to act and understand when to launch and/or when to look for investments. This, however, cannot be based on a random hunch. It takes deep understanding of all variables that play role in successful time management – understanding your product, your market, your competition, your resources.

These two factors in startups create a high pressure and dynamic environment as it has to continually address changing conditions within both external and internal dimensions. When the financial resources are limited, it puts a significant pressure on the founder and the team. Limited financial resources lead to limited time resources and this can create a chain of side effects and decision making under pressure. Production can be rushed; quality of the product can end up worse than initially envisioned and growth of a startup and its employees can be inhibited.

Other characteristics of startup include your and immaturity, as there is a very little accumulated experience. On the other hand, startups operate in a dynamic, flexible organizational or process structures. It accommodates the newest technologies, communication channels and more.[10] This high-paced model and quick adaptation serve as a benefit in comparison with established companies.

1.2 Software process

Second of all, the thesis will expand on the *software process* definition. Software process is described by Humphrey as: "the set of tools, methods, and practices we use to produce a software product."[11] Processes overall guide organizations through every step of their production and operations to assure efficiency and high-quality product. It is not different in software development. Software process has certain specifics in comparison to other process management industries. The software engineering is not a routine activity like a repetitive manufacturing. It is, as author suggests, an intellectual process that has to flexibly adjust to the creative necessity of software developing.[12] Due to its distinctive nature in comparison to other manufactural production processes, failures in

[9] Forourharfar, Amir. 2014. "Entrepreneurial Timing Theory : Time Entrepreneurship and Time Strategy." *Asian Journal of Research in Business Economics and Management* 4 (11): 1-27. https://www.researchgate.net/publication/268031860_Entrepreneurial_Timing_Theory_Time_Entrepreneurship_and_Time_Strategy, p. 2-3.
[10] Sutton, Stanley M. 2000. "The Role of Process in a Software Start-up." *IEEE Softw.* 17: 33-39. doi:10.1109/52.854066, p. 34.
[11] Humphrey, Watts. 1989. *Managing the Software Process*. I. Addison-Wesley Professional, p. 10.
[12] Humphrey, Watts, 1989. *Managing the Software Process,* p. 247-248.

software development can be quite costly.[13] Next subchapter unfolds basic components of a software process and different approaches modeling the process.

1.2.1 Parts of a software process

There can be found various outlines and views on how software process can be divided into individual parts. According to Sommerville, there are:

1. **software specification**
   it is a process when the system's requirements are being identified. List of functionalities of the software product is then listed and analyzed
2. **software design and implementation**
   customers' needs and requirements are then translated into executable software system in form of software architecture, data models, etc. The system is then produced and delivered through implementation and coding
3. **software validation**
   this stage summarizes all parts of testing, but also regular checks of the software process
4. **software evolution**
   is a stage when the software continues to evolve and improve to meet customer's needs[14]

Traditional division of a software process is known as Software Development Lifecycle (SDLC). It consists of 5 parts:

1. **Requirement analysis**
   customers' requirements are defined
2. **Design**
   requirements are designed into an architecture of the final product, desired software is designed and documented. Programming language, used technology and system's architecture is chosen
3. **Coding**
   the product is being coded using programming languages, usually in smaller parts or units
4. **Testing**
   the developed product is tested in a set of quality assurance, beta tests, unit tests
5. **Maintenance**
   the product is delivered to the customer and maintained throughout its usage. It can be debugged and there are new versions of the code deployed answering to customers' usage and feedback [15]

---

[13] See for example: Di Tullio, Dany and Bouchaïb Bahli. 2013. "The Impact of Software Process Maturity on Software Project Performance: The Contingent Role of Software Development Risk." *Systèmes d'Information Et Management* 18 (3) (09): 85-116,147. https://www.proquest.com/scholarly-journals/impact-software-process-maturity-on-project/docview/1491961460/se-2?accountid=17203.

[14] Sommerville, Ian. 2011. *Software engineering*. 9 ed. Boston, US: Pearson Education, p. 6.

[15] Dora, Sumit Kojar, and Pushkar Dubey. 2013. "SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) ANALYTICAL COMPARISON AND SURVEY ON TRADITIONAL AND AGILE METHODOLOGY." *Abhinav institute of technology and management* 2 (8).

Some other definitions of SDLC provide also a sixth stage named *Deployment*, which is placed between Testing and Maintenance. Deployment is a specific stage when the newly produced code is successfully tested, the code moves on to the production for the customer to use.[16] The existence of an individual stage for Deployment implies that the act of moving the code to the production consists of many partial actions which should not be neglected, such as system integration or security check.

Some other divide Requirement analysis to two stages of *Planning* and *Defining*.[17] Stage of planning is used for deadline and guidelines identification, resources planning and feasibility study. Defining focuses more on the business requirements on the software product.

1.2.2 Software Development Lifecycle Models

Specific parts of a process and the method of advancement are specified in models of SDLC. Most common SDLC models are waterfall model, incremental model, spiral model, prototype model, V-model and agile. As there is an individual subchapter for agile methodologies, the rest of the models are laid out below.

Waterfall model
uses basic SDLC stages. What is specific for the waterfall model is sequential advancement through different stages and is also referred to as plan-driven, as the software product is thoroughly analyzed and planned before moved to the production phase.

Incremental model
uses SDLC stages but in a series of versions (increments) while each new version contemplates the older one with added functionality[18]

Spiral model
is a combination of both sequential and incremental models placing more emphasis on risks analysis and overall risks management. This model uses different stages than ones identified in SLCP. These are planning, risk assessment, development & validation, evaluation of results and plan of next loops.[19]

Prototype model
does not differ from the incremental or waterfall models in terms of stages, but uses additional method called prototyping. Prototypes are initial versions of a software system. Usually, they are used to initially try out different design options. Building prototypes can help the software teams validate designed functionalities against the requirements, as prototypes allow users to visualize how the final product will look like.[20]

---

[16] See for example, "SDLC (Software Development Life Cycle) Phases, Process, Models." 2021. Software Testing Help. https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/.
[17] See for example, Kazim, Ali. 2017. "A Study of Software Development Life Cycle Process Models." *International Journal of Advanced Research in Computer Science* 8 (1) (01). https://www.proquest.com/scholarly-journals/study-software-development-life-cycle-process/docview/1901446145/se-2?accountid=17203.
[18] Sommerville, Ian. 2011. *Software engineering*, p. 27-29.
[19] Alshamrani, Adel, and Abdullah Bahatab. 2015. "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model." *IJCSI International Journal of Computer Science* 12 (1): 106-111, ISSN: 1694-0784, p. 107.
[20] Ibid, p. 45-46.

V-model
is based on the waterfall model but answers one of the biggest shortcomings of it – lack of testing. V-model integrates tests throughout all processes of requirements, specification, and design. There are 4 type of tests used in this model, which are also part of other SDLC models in the testing phase:

- Unit tests – are tests executed on the code of the new functionality produced
- Integration tests – test the newly produced code against the other components of the module, whether it coexists and communicates properly
- System tests – test the whole system and its communication with external systems
- Acceptance tests – are tests conducted on user's environment and check the functionality/product on customer's requirements[21]

1.3 Agile methodology

Third integral part of theoretical background of the proposed thesis is agile methodology and its usage in software development. Agile methodologies emerged as an alternative to heavyweight software development. Traditional plan-driven development, or a waterfall model as described in the previous subchapter, had shown certain shortcomings and asked for a more flexible development methodology that could react to the dynamic and ever-changing nature of not only software engineering, but also software implementation and requirements of the customers.

1.3.1 Traditional vs. Agile

As the name suggest in waterfall model, development team moves on to the next stage only after the previous stage is fully finalized and thoroughly documented. This model is extremely easy to use and adapt and with numerous benefits such as clean code, thought through design and on time delivery. On the other hand, there were certain limitations to the model. For example, design was not adaptive and the whole process needed to start over when any discrepancy was identified. It also does not provide customers any way to include their feedback in the software development process. Internal testing of the product is also delayed to the end of the process and debugging so late into the development, can be quite lengthy if there is any bug found in an integral part of the product.[22]

In order to address these limitations and keep the level of quality of a software product, an iterative, incremental method of development became an integral part of all agile methodologies. Incremental approach in software development allowed teams to go through the whole process of design, coding and testing in timeboxed iterations - all while answering shortcoming of waterfall model. A significant distinction between iterative model mentioned in the previous subchapter and agile methodology is that while incremental model releases increments that are new versions but all of them consists of all planned features, in agile methodology increments are partial functionalities of the system. Frequent releases of agile models allowed testing team to find complex bugs in time

[21] Kazim, Ali. 2017. "A Study of Software Development Life Cycle Process Models." p. 19.
[22] Malakar, Sudipta. 2021. *Agile Methodologies In-Depth,* p. 54-55.

before the code progressed to other parts. It also provided customers a platform for continual feedback when receiving increments of their required product.

In her in-depth analysis, Malakar mapped agile practices to traditional stages of waterfall model:

Table 1 Malakar's agile practice mapping to the waterfall practices[23]

Integration Management	Agile Planning
Develop project charter or plan	Develop roadmap and backlog
Execute the project plan	Do iteration work
Direct, manage, monitor, control	Facilitate, lead, collaborate
Integrated change control	Constant feedback, ranked backlog
Scope and Time management	Time boxes
Collect requirements, define scope	Develop and select product backlog
Create Work Breakdown Structure	Team selects features, tasks for sprint
Control schedule	Refine story estimates per team velocity
Quality management	Integrated quality
Quality planning	Responsibility of entire team
Quality assurance	QA integrated into sprints, retrospectives
Quality control	Unit testing in each iteration
Human resource management	Self-managing teams
Human resource planning	Dedicated team of 7 (plus minus 2) members

As Table 1 illustrates, agile practices do not abandon traditional and effective principles of software development which have been working for the last 60 years. Management is still needed in agile methodologies, but it relies more on self-management rather than control. Quality assurance has an important role, but the responsibility and execution are distributed among the whole development team and is being implemented to the coding process with unit testing. Requirements are voiced and analyzed, but not all at once, rather in bulks. This allows product to be gradually created and shaped to the customers' needs because of the continual feedback provided. It does not advocate for ad-hoc organization of a software team and software development. It rather changes the angle of the view on certain practices, rephrases essence of activities in order to comply with agile principles and puts emphasis on different variables than traditional approach.

[23] Malakar, Sudipta. 2021. *Agile Methodologies In-Depth*, p. 57.

1.3.2 Agile methodology formalization

While separate individual models emerged throughout 1990s, the formalization of agile principles occurred in 2001 with Agile Manifesto.[24] Four values of agile software development (ASD) became principles of every agile method.

1. Individuals and interactions over processes and tools

Much more than other methods, agile places emphasis on software developers.[25] Agile methods attempt to create a team of highly skilled and motivated individuals, as it considers them to be a key to a project's success.

2. Working software over comprehensive documentation

A human factor is also highlighted in this second principle. Software developers are empowered, because they can focus on polishing the software product rather than spend time on heavy documentation.[26] Clear objectives, feasible scope and valid operational results are valued over document deliveries and overall documents compliance.[27]

3. Customer collaboration over contract negotiation

A rigid and formal relationship between a supplier of a software and a customer has been revised in a third agile principle. Contract compliance made space for frequent communication with customers, regular meetings and frequent feedback all in order to strengthen customer relationship.

4. Responding to change over following a plan

An important part of agile principles is the flexibility and ability to response to the dynamic, ever-changing instability that software development brings - be it in form of engineering process itself, specifications of customers/business requirements or need to deliver software products fast and with high-quality. Project plans, scope and cost compliance were therefore sidelined in favor of flexibility – of the process, system, technology, management and thus, of the whole organization. [28]

1.3.3 Agile approaches

Agile methodology recognizes various individual frameworks that can be used simultaneously. Each focuses on certain areas of software process and develops tools in order to streamline the specifics within them.

---

[24] "Manifesto for Agile Software Development." 2021. https://agilemanifesto.org/.
[25] McAvoy, John, and Tom Butler. 2009. "A Failure to Learn by Software Developers: Inhibiting the Adoption of an Agile Software Development Methodology." *Journal of Information Technology Case and Application Research* 11 (1): 23-46. https://doi.org/https://doi.org/10.1080/15228053.2009.10856152, p. 24.
[26] Ibid, p. 24.
[27] Malakar, Sudipta. 2021. *Agile Methodologies In-Depth,* p. 39.
[28] Ibid.

Scrum

is a general framework based on agile methodology created to help teams develop, deliver and sustain complex products through adaptive solutions.[29] It provides set of definitions of team compositions, roles and responsibilities within, guidelines of software process stages and additional tools such as scrum artifacts and scrum rituals.

Scrum teams operate in time-boxed *sprints* during which an increment of product is being delivered. Each sprint has a defined *sprint goal* that is to be achieved by the end of every sprint. Both time limitation and goal definition provide scrum teams to be fully focused on job in front of them without any necessary distraction.

Sprint goal is defined based on *sprint planning* which is conducted before a sprint starts. On sprint planning, team defines what tasks it will work on. The tasks are gathered in *product backlog* and are moved on for development only after prioritization on sprint planning. This process can also be called *product backlog grooming*.

In the end of the sprint, there is a *sprint review* conducted. On sprint review, team presents increment of sprint to stakeholders and feedback is gathered. The teams with stakeholders define next increments of the product, so that the team can prepare for another sprint planning.

Sprint retrospective follows, which only development team attends. There, team discusses what went right and what when wrong during the sprint in order to improve the points for future sprints. This is also a good tool to boost team's morale in vocalizing positive parts and actions during the sprint.

The whole process is being monitored by *daily scrums*. They are short, daily meetings of a team where they discuss progress on their tasks, obstacles they face and possible solutions are offered. Daily scrums are powerful tool to have daily overview on team's work and progress, identify any impediments on time and overall streamline communication within the team. The need for other meetings is minimized and team can focus on coding.[30]

The team composition, roles and defined responsibilities in Scrum are discussed in the following subchapter.

Extreme programming (XP)

This framework was created when trying to push recognized good practices, such as incremental development, to extreme levels.[31]

One of the basic parts of XP are *user stories*. User stories are system requirements told and defined from the perspective of the user. They follow the format of:

Me <user> want <functionality> so that <why do I want it>.

Stories are then broken down to individual tasks. In XP, user stories accelerate two processes. They serve as a subject for estimations. In process called *planning poker*, development team estimate how much time will be spent delivering each user story. Members place their estimation and discussion

[29] Schwaber, Ken, and Jeff Sutherland. 2020. "Scrum Guide." https://scrumguides.org/index.html, p. 3.

[30] Scrum summary was based on the reseacher's knowledge and Scrum guidel - Schwaber, Ken, and Jeff Sutherland. 2020. "Scrum Guide.".

[31] Sommerville, Ian. 2011. *Software engineering*, p. 64.

is undergone if the estimations do not match to find out the most balanced estimation of the user story.[32]

XP places big emphasis on testing. User stories enable *acceptance tests*. As user stories are written from user's perspective, and in some cases by customers themselves, definition of acceptance tests can be directly derived. Second tests which XP deems a must are *unit test*s. XP believes no part of the code can be produced without unit tests. By this, developers are obligated to at least create primary tests of the code function.

Coding practices are pushed to extreme levels as well. XP recommend *pair programming* in which the same part of the code is being developed by two developers. This ensures continuous monitoring and support in every stage of coding. On the other hand, it is not a very efficient use of resources. Alternative to that with similar effects are *code reviews*. When the code is finished, but before integrated to the system, it is being reviewed by either technical lead or other developers in peer reviewing. Some mistakes can be found, some parts commented to redo, and higher quality of a software is ensured as a result. Developer has also access to a constant feedback and improvement.[33]

As XP relies on *continuous incremental deployments*, new features can disable features deployed in previous increments. *Refactoring* is, therefore, encouraged whenever there is a need and/or time. Refactoring means optimizing and checking the old, already developed code, to keep the code simple and maintainable. Another principle of XP is *collective ownership*. This places responsibility for the code function to the whole team, not only to the person who developed a certain part. On the brighter side, everyone can change anything in the system. This proves useful when reinforcing refactoring of the system.[34]

Kanban

is a method for defining and improving deliveries of services. Kanban is in software development suitable for all design proceedings. Its tools and practices can also be incorporated to the software engineering process.

Kanban relies on the visualization of work in forms of *kanban cards*. As work is intangible, Kanban makes them visible and enhances the process of getting the work done. Cards are then placed on *kanban boards* which are visible to the whole team. Cards can be customized based on the needs of each team. For example, there can be different colors used for different people that have assigned task. Or colors can be used based on priority of a task. If the task is an emergency, the color red can be used for that specific card. The team can thus see on the first sight, what they have to turn their attention to.

On the horizontal axis there are *kanban stages* through which each card's flows. These stages are usually determined by each team individually as each team needs different workflow. The most basic workflow is To Do, In Progress and Done. Every progression to stage needs to have a clear definition on how the card can move to another stage.

---

[32] Grenning, James. 2002. "Planning Poker: How to avoid analysis paralysis while release planning." http://renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf.
[33] Sommerville, Ian. 2011. *Software engineering*, p. 71-72.
[34] Ibid, p. 66-71.

Cards can proceed through streamlined process in so called *swimlanes*. These are used for hot fixes, emergencies, when the process needs to be expedited and does not have ability to move through defined stages one by one. Cards are moved to next stages on *kanban meetings* where the team provides regular status while moving the cards based on their progression.
What is important in this process are *work in progress* (WIP) limits. Kanban believes in an effective pull mechanism, when cards needs to be completed in later stages in order for more work to arrive from earlier stages. [35] For example, let's say the WIP limits for To Do stage is 3 and for In Progress is 4. If there are 4 cards already in In Progress stage, we cannot move any cards from To Do stage to this stage as the WIP limit does not allow us to do so. First, we have to finish one of the cards in In Progress stage and move it to Done. The number of cards in In Progress stage is now 3 and we can move the card from To Do stage now. The idea behind is that one should not accumulate and start many tasks at once, but rather one should try to always complete a task before starting another. By this the focus is set and probability of completion is higher.

1.4 Software teams

Previous subchapter showcased how integral are software developers and individuals in software teams. The thesis, therefore, elaborates on usual software teams compositions and roles played. The emphasis on the software teams positions and functions also acted as major factor to improve in the practical part of this thesis.

1.4.1 Organizational structures

A software team composition is influenced by the overall *organizational structure* used in companies, and also within the software department itself. Basic organizational structures are:

A) Hierarchical
is the most traditional way to structure an organization. The chain of command goes from top to the bottom. Everyone has one person to report in a pyramid level chart with numerous layers of middle to upper management. While it has obvious benefits for the management, as everyone knows what their responsibility is and who is their supervisor, this structure can disengage lower-level employees from organization's vision and create unmotivated environment.
B) Functional
Functional structure also relies on hierarchical top-down approach but divides an organization to functional parts based on their skills and competence. It is what we understand under marketing department, software department, sales department, etc. Each department is then managed individually which can sometimes lead to lack of intradepartmental communication.
C) Flat

[35] Kanban summary was conducted based on the researcher's knowledge and Anderson, David J., and Andy Carmichael. 2016. *Essential Kanban Condensed*. Lean Kanban University Press.

has one or very few levels of hierarchy. It is flatted out throughout the whole organization. This form of structure be found in startups before they grow into a larger company. The responsibility and competence are divided between all employees of organization. This can have positive effect in their encouragement, but negative effect in creating confusions and ineffective decision-making.

D) Flatter

Quite new concept of getting the most out of a flat organization while addressing their shortcomings. It is based on hierarchical and/or functional structure but encourages communication across and within the levels. This creates an environment where everyone has their own area of responsibility, specific person/people to report to and has a platform to voice their concerns and influence decisions.

E) Divisional

Divisional structure happens when an organization is divided to basically few little suborganizations within. Each division has its own budget, resources planning and can have their own functional departments such as e.g., IT or marketing. This structure is helpful in large companies to improve the effectiveness of their operations but can lead to insufficient communication within divisions or easily duplicable resources.

F) Matrix

Matrix structure is when different functional teams are formed across their departments for a certain project. There are multiple lines of command, when an individual reports to their manager in department, but also to the project lead/manager on a project they are assigned to. This fact alone can lead to certain conflicts tight with work division and priority definitions. It, however, promises flexibility and serve greatly for organizations which function in time-scoped projects rather than in continuous production.[36]

1.4.2 Roles in software teams

Second part of this subchapter focuses on *software roles* that can be usually found in software teams. They can be derived from the software process specifications as each process needs to be carried out by a competent individual skilled to perform the process. The variety of skills that are included in the software process alone ask for the need to have variety of software roles and responsibilities in a software team.

---

[36] This classification was formed with various contemplating sources:
Steiger, Jen S., Khalid Ait Hammou, and Md Hasan Galib. 2014. "An Examination of the Influence of Organizational Structure Types and Management Levels on Knowledge Management Practices in Organizations." *International Journal of Business and Management* 9 (6). https://doi.org/10.5539/ijbm.v9n6p43.
Morgan, Jacob. 2015. "The 5 Types Of Organizational Structures: Part 2, 'Flatter' Organizations." Forbes. https://www.forbes.com/sites/jacobmorgan/2015/07/08/the-5-types-of-organizational-structures-part-2-flatter-organizations/.
Williams, Shanon. 2021. "7 types of organizational structures (+ org charts for implementation)." Lucid Chart. https://www.lucidchart.com/blog/types-of-organizational-structures.

Project Manager

Role of a Project manager does not differ whether applied on traditional or agile software approaches. The project – product development – is a process that needs to be overseen, organized and facilitated. Project manager supervises work of software engineers and monitor the progress of the development. As project management is usually tied with plan-driven approach, role of a project manager had to change in agile methodologies from commanding to empowering.[37] Project manager no longer controls and enforces, but rather facilitates and enables the development process. Some of the tools of agile methodologies also helped project management to function in incremental. For example, above mentioned Sprint planning of Scrum methodology institutionalized process of planning which is in incremental models harder to execute In Scrum, title of a project manager role changes to scrum master who facilitates basic rituals, oversees progress, communicates with customers and other colleagues outside of a team.[38]

Next roles are tied with product, customers' requirements and specifications and analysis.

Product Owner

As the name suggests, Product owner is responsible for the product's value and product backlog.[39]They are in close contact with customers and translate their needs and even unspoken requirements to the development team. Product owners are the ones that define what is going to be developed next in incremental and agile models. They are also the ones who validate the produced functionality before it proceeds to customers. Along with Scrum master, Product owner is one of the three roles defined in Scrum.

Product Manager

Responsibilities of Product manager and Product owner do not differ. If a company does not adhere to Scrum, Product manager has similar role and responsibilities of a Product owner. If both roles are defined in one organization/team, these responsibilities change in scope, not in their content. Product manager is more focused on external stakeholders, while Product owner works closely with internal development team. Product manager defines the vision of the product, and Product owner defines how will this vision be achieved. [40] It is implied Product managers work more on a conceptual way of a product definition, while Product owners on a day-to-day operations layer of the process.

Business Analyst

Business analyst is a person who takes the requirements and transform them into detailed specifications of a functionality including all diagrams and process flows.[41] Business analyst must understand the need behind requirements while also have a very deep knowledge of already developed parts of systems. It is because they need to ensure all codependent functionalities in the system are considered. Business analyst is also the person who writes user stories in case XP is incorporated. If Product

---

[37] Malakar, Sudipta. 2021. *Agile Methodologies In-Depth,* p. 106-112.
[38] Sommerville, Ian. 2011. *Software engineering*, p. 72.
[39] "What is a Product Owner?" 2021. Scrum. https://www.scrum.org/resources/what-is-a-product-owner.
[40] Mansour, Shefir. 2021. "Product Manager: The role and best practices for beginners." Scrum. https://www.atlassian.com/agile/product-management/product-manager.
[41] Murray, Anna P.. 2016. *The Complete Software Project Manager : Mastering Technology from Planning to Launch and Beyond*. Hoboken: John Wiley & Sons, Incorporated, p. 53-54.

owner and/or Product manager roles are not in the team, Business analyst gathers customers' feedback and define requirements on the software.

User Experience (UX)

UX is derived from user experience. UX sets how the functionality acts in user's environment and how are users navigating throughout the system.[42] Fundamentally, UX is the one who defines what click leads to what. UX needs to ensure the usage of the system/product is as smooth for the user as possible and there are no unnecessary clicks or reroutes that could have been avoided. If prototypes of final product are used in a software process, UX is the one who is creating them.

Product Designer

is the one responsible for the product's graphic representation and their design. Product designer takes either role of UX, or very closely works with the UX in a team.

Following part defines technical roles in a software team.

Software Architect

Software architecture is set of standards, models and components of the whole system.[43] Software architects are the ones designing the architecture and its development based on new requirements coming. Based on business requirements and/or user stories, software architects design new features, components needed for their development and integration to the existing system. Based on this design, software architects break down the work needed to produce the new functionality into development tasks for developers.

System Architect

Difference between Software and a System architect is again in scope. System architects focus on high-level decisions affecting whole system such as coding standards, security, monitoring and maintenance. Software architects design a specific part/module/functionality of the system in a software development process lifecycle. System architects operate outside the lifecycle.

Developer / Software engineer

Developers are the ones finally producing designed requirements into a functional code to the system. There are three types of developers based on the specializations – backend, frontend and full-stack developers.

Backend developers are the ones operating on the data access layer, while frontend developers use these data on presentation level. Frontend is the part of the system that users see, while backend is are the databases and logic behind it.[44] Frontend and backend both use different technologies and majority of developers specialize in one of them. Then there are full-stack developers who can code in both layers and can work on frontend and backend tasks.

Developer is the last role included in Scrum teams. While there is only one Scrum master and Product owner, there can be numerous developers in one Scrum team.[45]

---

[42] Murray, Anna P.. 2016. *The Complete Software Project Manager*, p. 54.

[43] "Software Architecture." 2021. Software Engineering Institute, Carnegie Mellon University. https://www.sei.cmu.edu/our-work/software-architecture/.

[44] Murray, Anna P.. 2016. *The Complete Software Project Manager,* p. 56-57.

[45] "What is a Developer in Scrum?" 2021. Scrum. https://www.scrum.org/resources/what-is-a-scrum-developer.

DevOps Engineer

is a combination of "development" and "operations". DevOps engineer probably needs the widest range of skills in a software team. That includes coding, infrastructure management, system administration. DevOps engineer enables whole development process with automation and orchestration tools (tools in which developers produce the code) and work with development team to deploy the code to different environments including production.[46]

Technical Lead

Not always present, technical lead is person who is part of a development team but bears certain responsibilities of the team. Software architects break down a cohesive functionality into small tasks. These small tasks need to be distributed among the development team, but are usually connected, and/or overlap, and/or are dependent on each other. That's where technical lead comes into play as they are responsible for division and assignment of these tasks, communication and integrity of separate tasks. In the end, it is Technical lead who is delivering a coherent functionality their team put together. Technical lead, therefore, needs not only programming skills, but also leaderships, organization and communication skills.

Tester / Quality Assurance Engineer

Finally, tester is a responsible for validating the produced code against the requirements. Unit tests are conducted by the developer who produces the code. Integration tests are performed by combination of architects, developers and/or technical leads. Testers test the code in user's environment. Role of a tester does not require programming skills. With good understanding of process and requirements, tester can test the functionality from their frontend part. However, some of the defects or bugs happen on backend part of development. To be able to conduct tests also on this layer, basics of programming are recommended for the role of

1.5 Software process improvement

One of the flagship institute of a software process defines a process "as a leverage point for an organization's sustained improvement."[47] This, among other things, implies that a high-quality software is a direct result of a high-quality software process. Developing a software is an expensive and long journey. Reports show more than half of the projects exceeds their initial budget estimates by 189%.[48] Need for optimization of software production and mitigation of risks during the process has led to a separate field developed to address these challenges - *software process improvement* (SPI).

It is defined by Rico as "discipline of characterizing, defining, measuring and improving software management and engineering processes, leading to successful software engineering management, higher product quality, greater product innovation, faster cycle timers and lower development

---

[46] Kulshrestha, Saurabh. "Who Is A DevOps Engineer? — DevOps Engineer Roles & Responsibilities." Edureka. https://medium.com/edureka/devops-engineer-role-481567822e06.

[47] CMMI Product Development Team. *CMMI for Systems Engineering/Software Engineering, Version 1.02, Continuous Representation (CMMI-SE/SW, V1.02, Continuous)* (CMU/SEI-2000-TR-019). 2000. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5267, p. 16.

[48] Ibid, 85.

costs".[49] Card uses a vaguer description of SPI, where he believes it consists of software assessment and improvement of all processes throughout software development.[50] Coleman and O'Connor also uses abstract definition of SPI to be: "the way an organization develops its software products and supporting services, such as documentation."[51] For the purpose of this thesis, we can define SPI to be a set of methods and tools, which will allow software development teams and projects to improve their production development process.

Concept of *reliability* and *predictability* is also important for the field of SPI. Reliability is defined as "an extent to which the same measurement procedure yields the same results on repeated trials."[52] If reliability is reached, process can become predictable. Anything that is predictable can then be produced over and over again with the same result. Once the process is perfected and it can produce a quality software, it is desirable the same level of quality is reached with every software delivery.[53] A defined process can also help everyone in the software team to understand their role and what is expected from them.

In comparison to exact studies, abstract concepts of social and behavioral studies such as software process cannot be measurable and objectively observable. In this case, one cannot measure reliability on figures and numbers, rather on empirical indicants. SPI, therefore, developed *key process areas*, in which improvement of a software process can be measured.[54] Key process area is used in CMM (later CMMI) and all models and frameworks that were inspired or derived from CMM. The frameworks describe them as set of practices that are undergone in order to reach certain objectives. Key process areas are, therefore, actions (*specific practices*) but also anticipated results that are supposed to be achieved through these actions (*specific goals).*[55] List of software development key process areas varies based on the model and approach of a SPI model. Next chapter elaborates on SPI and process areas concepts further.

2 RESEARCH and LITERATURE REVIEW

The following chapter describes the evolution of software process in the academic field and in the research area of software engineering. It deals with traditional models used in software process

---

[49] Rico, David F. "Software Process Improvement: Impacting the Bottom Line by using Powerful "Solutions"." 1997. http://davidfrico.com/spipaperpdf.htm., p.1.
[50] Card, David. 2004. "Research directions in software process improvement." *IEEE Xplore: Computer Software and Applications Conference, 2004*. https://doi.org/http://dx.doi.org/10.1109/CMPSAC.2004.1342834.
[51] Coleman, Gerry, and Rory O'Connor. 2007. "Using grounded theory to understand software process improvement: A study of Irish software product companies." *Information and Software Technology* 49 (6): 654-667. https://doi.org/10.1016/j.infsof.2007.02.011., p. 654.
[52] Ho-Won Jung, & Dennis Goldenson. 2002. *The Internal Consistency of Key Process Areas in the Capability Maturity Model (CMM) for Software (SW-CMM)* (CMU/SEI-2002-TR-037). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=6279, p. 2.
[53] Humphrey, Watts, 1989, p. 10-12.
[54] Ibid, p. 2-3.
[55] CMMI Product Development Team. *CMMI for Systems Engineering/Software Engineering, Version 1.02, Continuous Representation (CMMI-SE/SW, V1.02, Continuous)* (CMU/SEI-2000-TR-019). 2000, p. 24.

improvement (SPI), but also with tailored models which will be later expanded in the practical part and will be applied to the research of the proposed thesis. The importance of this chapter is to support the arguments behind choosing the SPI model for the researched thesis.

Genesis of agile methodologies and its establishment in the software development is, for the purpose of this thesis, neglected. First of all, while it uses their tools in implementing frameworks, the thesis aims to contribute to works of software process improvement rather than one of agile methodologies. Second of all, the evolution and genesis of agile methodologies seems to be exhausted in not only international, but also local studies with the emerging popularity of methodology.

2.1 Initial standardization of software development process

In a relatively young study field such as software engineering, it is hard to define and describe an academic path from establishment to present. In order to profoundly evaluate genesis of an academic field, one might need more than just 60 years of existence. However, even in such short history, there can be identified certain milestones in studying software development – both taking place in 1960s.

One of the first attempts to conceptualize study field of software engineering was in 1960 embodied in Technical Committee 97 which purpose was to provide international standardization of the field of information processing.[56] What later become JTC1/SC7, had initial purpose of addressing standardization of flowcharting techniques and representations.[57] It had later focused interpreted its study area as standardization of processes, supporting tools and supporting technologies for the engineering of software products and systems with the emphasis being on a process.[58]

Another international attempt took place later in 1960s with NATO Garmish conference in 1968. The Science Committee of NATO established a Study Group which was tasked to assess entire field of computer science in an effort to define and accelerate possible international actions in this field.[59] These assessments were then later, besides other things, presented and discussed on the aforementioned conference and its second part held in 1969. Main deliveries of these conferences were two coherent reports, set of best practices to be used in software development and initialization of software engineering as an academic field to study and research.[60]

Up until 1960s, the focus of studies centered around the technical aspect of software development – technologies used, programming languages, coding, etc. What this international attempt contributed to was emphasis on the software processes. As such, best practices in software development

---

[56] Note: Technical Commitee 97 is part of International Electrotechnical Commission – an international commitee providing standards for electric, electronic and other related technologies. Coallier, Francois, and Motoei Azuma. 1997. "Introduction to Software Engineering Standards." In SPICE: The Theory and Practice of Software Process Improvement and Capability Determination, Khaled El Emam, Jean-Normand Druoin, and Walcélio Melo, 1-18. Wiley-IEEE Computer Society Pr, p. 5.

[57] Ibid, p. 5-6.

[58] Ibid, p. 5.

[59] Naur, Peter, and Brian Randell. "SOFTWARE ENGINEERING: Report on a conference sponsored by the NATO SCIENCE COMMITTEE Garmisch, Germany, 7th to 11th October 1968." *NATO SCIENCE COMMITTEE* 1969. https://www.scrummanager.net/files/nato1968e.pdf, p. 8

[60] Brennecke, Andreas, and Reinhard Keil-Slawik. 1996. "History of Software Engineering." *Position Papers for Dagstuhl Seminar 9635*. https://www.dagstuhl.de/Reports/96/9635.pdf, p. 37.

basically conceptualized what later become software process improvement (SPI). Another reason of such shift was due to increasing strategic importance of software product. As in history of every manufacturing process, the process of creating software has become a center of attention in order to utilize and streamline the path from initial inputs to a finished product. This need for systematic approach in software development was translated into creation of numerous methodologies around which even recent studies have been centered.

A systematic state of the art of software process improvement (SPI) found out that especially from 1996, there can be seen a steep, more than 400% increase in frequency of SPI papers published per year.[61] The following subchapters define some of the most established models and their variants.

2.2 SPICE (ISO/IEC 15504)

A developed and established version of the subcommittee's SC7 efforts discussed above was formalized into a SPICE project. SPICE was one of the first attempts in software process assessment which encouraged software teams to self-evaluation. The aim of SPICE standards was to provide companies a determination framework in assessing a software supplier, in improving own software development and in self-assessment to assist organizations in identifying its ability to implement new software projects.[62] The model can, therefore, be used for not only software development companies, but also any other organizations buying a software, using a software or implementing a software. Which is basically every single corporate organization, as trends nowadays demonstrate.

The standards consist of nine parts which lead the organization in the assessment of software process. Besides introductory parts, references or glossary, the main parts of the model is part 2, which defines framework in defining processes' capability. This part consists of five key areas - customer-supplier process, engineering process, project process, support process and organization process.[63] As it emerged after the initial drafts of later discussed Capability Maturity Model, which provides framework for process improvement, SPICE did not attempt to provide guidelines in this area. It focuses solely on evaluating software organization's process capability.

2.3 Capability Maturity Model (CMM) and Capability Maturity Model Integration (CMMI)

One of the most established methods of software process improvement started to develop in 1980s. As the need for the quality software grew higher, the U.S. Department of Defense requested a method, that would enable them to evaluate contractors who can make software on budget and on

[61] Kuhrmann, Marco, Philipp Diebold, and Jürgen Münch. 2016. "Software Process Improvement: A Systematic Mapping Study on the State of the Art." *PeerJ Computer Science*. doi:http://dx.doi.org/10.7717/peerj-cs.62, p. 10-12.
[62] Konrad, Michael D., Mark C. Paulk, and Allan W. Graydon. 1995, p.2-3.
[63] Ibid, p. 3.

time.[64] At this request, the Software Engineering Institute, funded by the U.S. government, launched an initiative to develop a maturity framework model for software processes. The first findings were drafted in 1987, which developed into conceptualized *Capability Maturity Model* (CMM) in 1991.[65] In short, the CMM model identifies five maturity levels in which every software company or software team finds itself. Depending on the starting position it can move to the higher maturity level based on model's guidelines and methods. The levels range from ad hoc development to optimizing stage. What indicates in which level every company finds itself are key process areas (KPAs) – their existence and their organization's support.[66] CMM model can also be considered as an establishment of project management in software development, as it puts emphasis on project management tools and quality management control in order for a company or a team to move from their stage to a higher one.

In parallel with developing CMM, SEI's director Watts Humphrey published a book *Managing the Software Process* in 1989.[67] There, the author uses process management methods and applies them to software development. He builds on foundations of CMM with five maturity levels and expands the theory with practical implementation of the model. Humphrey's publication places emphasis on improving, rather than assessing software process.

In the span of 1990s, SEI developed five more versions of CMM in order to overcome certain shortcomings of the model. The versions focused on formalization of maturity levels in terms of key process areas.[68] Not even newer versions of CMM prevented its eventual retirement in favor of a developed Capability Maturity Model Implementation (CMMI).

As its title suggests, the CMMI focused on the implementation part of the model, not just on the theory which was one of the main drawbacks identified by experts and even authors of the CMM model. Another setback experts found in CMM was that it focused on the process and artefacts associated with traditional waterfall process such as requirements specification, documented plans, etc. All that at the expense of addressing results of the process such as software product, or associated engineering artefacts.[69] Upgrade to CMMI from CMM was supposed to bring in benefits in the area of visibility and coherence of all activities across the development cycle. These include, for example more explicit connection between management and engineering activities to business goals, to provide better clarity on objectives the engineers work on or bigger emphasis on customer satisfaction.[70] These improvements addressed one of the hurdles of CMM which was lack of integration of separate models. To move from activity-based approach, which only focused on whether the key process area

---

[64] Keshta, Ismail. 2019. "Approaches to Software Process Improvement: A State-of-the-Art Review." *Journal of Software*: 519-529. https://doi.org/10.17706/jsw.14.11.519-529, p. 520.
[65] Paulk, Mark C. 2009. "A History of the Capability Maturity Model for Software." Software Quality Professional 12 (1). https://www.researchgate.net/publication/229023237_A_History_of_the_Capability_Maturity_ModelR_for_Software, p. 5.
[66] Ibid, p. 8.
[67] Humphrey, Watts. 1989. *Managing the Software Process*. I. Addison-Wesley Professional.
[68] Paulk, Mark C. 2009. "A History of the Capability Maturity Model for Software.", p. 11-13.
[69] Royce, Walker. 2002. "CMM vs. CMMI: From Conventional to Modern Software Management." *The Rational Edge*. https://www.csc.kth.se/~karlm/CMMI.pdf, p.2-3.
[70] "Upgrading from SW-CMM® to CMMI®." 2004. *Software Engineering Institute, Carnegie Mellon University*. https://resources.sei.cmu.edu/asset_files/WhitePaper/2004_019_001_29417.pdf.

was achieved, without further exploration of the results, the CMMI also defined goals for every key process area. This allowed organizations implementing CMMI to track their progress gradually.

Mark C. Paulk, summarized this model to have "an enormous impact on the software community". He estimated billions of dollars spent in variety of companies and software projects to improve methods based on CMM.[71] Even more impressive testament of the model is number of models and methods that deviated from the CMM model. In the sense, CMM was the first model to establish a procedural view on software engineering, but through its shortcomings, software engineering was able to move to the iterative, results-based production, which is finding its success nowadays.

2.4 Software process improvement for small companies

Unfortunately, due to complexity of both CMM and CMMI, small companies or startups do not have resources needed to apply software assessment and improvement methods. The insufficiency of CMM for small companies is also implied by the model itself, where it considers a small team to be a team of 70 people.[72] However, nowadays small companies, with fewer than 50 employees, and very small companies, with fewer than 25 employees, represent almost 85% of all software companies in countries which have a developed IT industry.[73] The general consensus is that SPI can be applied to small organizations only with limitations. Sutton argues that a highly systematic approach to software process is not compatible with fast-paced and reactive environment of software development. Even more so, if such environment combines with a startup. On the other hand, he believes startup companies should concern themselves with SPI models, especially CMM's central concepts. It can be problematic to adhere to all of them strictly and a startup should start at lower levels and perfect them before moving to higher ones. Sutton recommends selection and prioritization of process issues to startup's needs and resources.[74]

Batista and Figueirdo attempted to apply CMM in 2000 to a small team. However, the lack of resources that can be seen in small teams and lack of managerial human recourses to lead the implementation of CMM led to not satisfactory levels – a team could not move to Level 2 of maturity as per CMM.[75] For that, not only some variants of CMM model were tailored to suit the resources and necessity of small companies, but also individual attempts at software process improvement methods started to develop with the millennial change. This trend can be demonstrated by the increased frequency of SPI papers focusing on small or medium sized companies.[76]

---

[71] Paulk, Mark C. 2009. "A History of the Capability Maturity Model for Software.", p. 5.
[72] Batista, J., and A. Dias de Figueiredo. 2000. "SPI in a Very SmallTeam: a Case with CMM." *SOFTWARE PROCESS IMPROVEMENT AND PRACTICE* 5. https://www.academia.edu/4324846/SPI_in_a_very_small_team_a_case_with_CMM, p. 243.
[73] Lee, Seiyoung, and Hwan-Seung Yong. 2013. "Agile Software Development Framework in a Small Project Environment." *J Inf Process Syst* 9 (1). https://doi.org/http://dx.doi.org/10.3745/JIPS.2013.9.1.069, p.69.
[74] Sutton, Stanley M. 2000. "The Role of Process in a Software Start-up.", p.35-39.
[75] Batista, J., and A. Dias de Figueiredo. 2000. "SPI in a Very SmallTeam: a Case with CMM.", p. 245-249.
[76] Kuhrmann, Marco, Philipp Diebold, and Jürgen Münch. 2016. "Software Process Improvement: A Systematic Mapping Study on the State of the Art.", p. 16-18.

Process Improvement for Small to Medium Software enterprises (PRISMS) is one of the examples. As the title suggests, PRISMS was created to meet the specific needs of small companies, or rather to accustom to the limitations small companies have in comparison to bigger ones in implementing any improvement models.[77] The framework uses CMM's key process areas but do not dwell on the rigid levels and their areas, rather focuses on both development's team evaluation and business team set of goals to identify the key process areas to improve on themselves.[78] The model places emphasis on defined business objectives and priorities before implementing a software process improvement methods, similar to Sutton's recommendations.

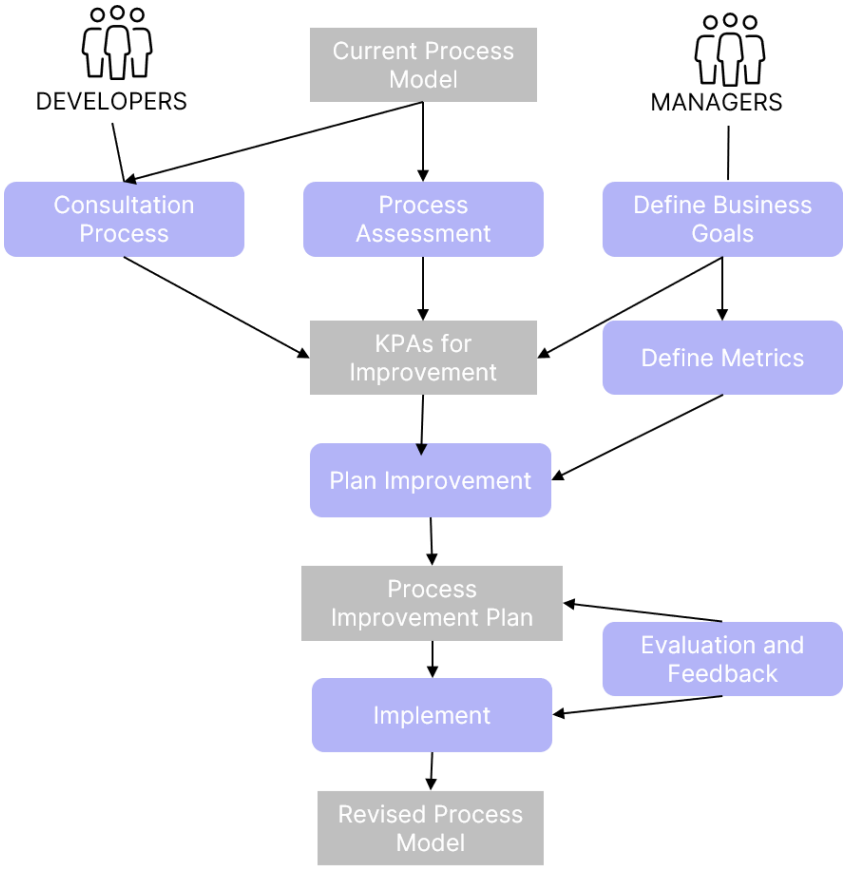
The whole layout of the model can be seen in Picture 1. Lightly highlighted cells are for deliveries or artefacts of certain process, while dark cells define procedural steps in model. Key process areas to improve are only defined after current process assessment, consultation with the development team and definition of business goals. The research team of PRISMS model assists in two of the described steps: assessment and measurement. For assessment, authors provide *awareness* and *business-case workshop* to be useful tools for companies to adopt. For measurement of process' success, authors use simple Karner's payoff method, which simply weighs in investment resources used for SPI and returns an improved software process brought to the organization.[79] The model then builds its research on CMM framework in identifying key process areas with the organization. The key process areas are then prioritized to align with the established business objectives.

[77] Allen, P., M. Ramachandran and H. Abushama. 2003. "PRISMS: an approach to software process improvement for small to medium enterprises." *Third International Conference on Quality Software, 2003. Proceedings*: 211-214. https://www.semanticscholar.org/paper/PRISMS%3A-an-approach-to-software-process-improvement-Allen-Ramachandran/54e62c1a6c2c4c77d3a1aebd59648b13ceecfe1a

[78] Ibid, p. 212-213.

[79] Krasner, Herb. 2001. "Accumulating the Body of Evidence forThe Payoff of Software Process Improvement." *Software Process Improvement*: 519-539. https://www.researchgate.net/publication/245582411_Accumulating_the_Body_of_Evidence_for_The_Payoff_of_Software_Process_Improvement.

Figure 1 PRISMS model for software process improvement, author's own work based on PRISMS definition[80]



Another attempt in accustoming SPI models on small companies was based on Quality Function Deployment (QDF). QDF relies on the constant customer's voice throughout the process of delivering the product. It was developed to have the constant customer's requirements while progressing the production.[81] For SPI, the process is considered to be customer's requirement and is applied for procedural needs. To follow the SPI process, QDF suggests providing thorough documentation of the process and systematic assessment, plan and implementation of software design. Similar to PRISMS, this model understands small companies' constraints such as limited resources and/or managerial time. It, therefore, highlights the importance of clearly defined business goals. Later, it provides a measure of importance of each process in the SPI project every organization decides to undergo. By providing organizations definition of priority for each process they decide to improve, it allows them to focus on key process to improve and move on to next ones in an

[80] Allen, P., M. Ramachandran and H. Abushama. 2003. "PRISMS: an approach to software process improvement for small to medium enterprises.", p. 212.
[81] Richardson, Ita. 2002. "SPI Models: What Characteristics are Required for Small Software Development Companies?" *Software Quality Journal* 10 (2) (09): 101-114. doi:http://dx.doi.org/10.1023/A:1020519822806, p. 102-103.

incremental process.[82] The implementation of SPI methods is then spread out over time and the organization can benefit from its results earlier. In a sense, QDF uses similar philosophy as today's incremental, agile methodology.

2.5 CMMI-DEV

To accustom to the specifics and trends of small software project, SEI also developed a CMMI version. CMMI-DEV version is aimed at developing software projects. The rigidity of CMM and CMMI is in this version minimized. It does not define that an organization has to follow a certain set of steps. It only specifies: "…that an organization should have processes that address development related practices."[83] For that, CMMI-DEV provides two paths in which organization can progress – continuous and staged. A continuous path provides a more incremental improvement and based on processes each individual organization defines for itself. Another follows traditional approach of CMMI with defined set of process areas. Based on the path chosen, the organization progresses from either capability levels – in case of a continuous path, or from maturity levels – in case of a staged path. *Capability levels* are applied directly to achievements or goals of a certain key process area. On the other hand, maturity levels are applied across multiple key process areas at once.[84] In that sense, capability levels can be used to track progress in iterative, smaller teams. Apart the other things, this distinction is highlighted by the comparison of levels in Table 2.

Table 2 Levels of continuous and staged path of CMMI-DEV framework, authors own work based on CMMI-DEV[85]

Level	*Continuous path – Capability levels*	*Staged path – Maturity levels*
0	Incomplete	N/A
1	Performed	Initial
2	Managed	Managed
3	Defined	Defined
4	N/A	Qualitatively managed
5	N/A	Optimizing

[82] Richardson, Ita. 2002. "SPI Models: What Characteristics are Required for Small Software Development Companies?", 109-110.
[83] CMMI Product Team. *CMMI for Development, Version 1.3* (CMU/SEI-2010-TR-033). 2010. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661, p. 21.
[84] Ibid, 19-21.
[85] Ibid, 23.

As Table 2 demonstrates, the capability levels do not reach the same optimizing stage as maturity levels. Other than that, the addition of Level 0 to Capability levels implies that continuous path can be used for organizations that lack even the basic principles and process standardization. Descriptions of the capability levels in CMMI framework are as follows:

Capability level 0 – Initial

means the process is not performed at all or is, but very partially. This is the level most small companies or startups find themselves in the early stages.

Capability level 1 – Performing

a performed process is a process which is making actions but may not be that stable. Performed practices might move the company towards its goals, but if not institutionalized, the process can revert to incomplete stage and goals would no longer be met

Capability level 2 – Managed

once the management of the processes is established, it means the process is planned and executed accordingly. The process is then monitored and controlled and all stakeholders in the process have skills and time resources to perform these processes

Capability level 3 – Defined

a defined process is a process established in company's policies. Standards, process descriptions and procedures are tailored for organization's standards and not just for one of organization's projects, as is the case in a managed capability level 2. Defined process has a clearly described purpose, inputs, entry criteria, activities, roles, measures, verification steps, outputs and exit criteria[86]

Maturity levels, on the other hand, provide a stricter and more structured definition of each stages.

Maturity level 1 – Initial

is a stage where an organization is in chaotic, ad-hoc processes. The results of the processes usually produce products and services, but the production exceeds budget and time scope of initial estimations

Maturity level 2 – Managed

does not significantly differ from capability level 2

Maturity level 3 – Defined

does not significantly differ from capability level 3

Maturity level 4 – Quantitatively managed

defines a stage of an organization which uses quantitative objectives to measure and improve their process performance and quality of the product. These two variables are considered as statistical terms and individual steps of the processes are statistically measured and analyzed

Maturity level 5 – Optimizing

is an organization which continuously improves based on quantitative understanding of its processes. The difference between level 4 and level 5 maturity level is that while level 4 focuses on controlling

---

[86] Ibid, p. 24-25 and CMMI Product Development Team. *CMMI for Systems Engineering/Software Engineering, Version 1.02, Continuous Representation (CMMI-SE/SW, V1.02, Continuous)* (CMU/SEI-2000-TR-019). 2000, p. 25.

and formulating the quantitative objectives, level 5 moves on to managing and improving these objectives.[87]

Process areas are common for both continuous and staged path. Each process area has *a specific goal* defined which determines if the process area is fulfilled. What is different is the way an organization advances through the levels. An organization can move on to the next capability level by applying general practices or alternatives for the processes in the associated process areas. They are measured by *generic goals*. It is demonstrated on Picture X. For a process area to move to the next level, all its specific goals needs to fulfill generic goals of the level. Generic goals are for the capability levels:

GG1 Performed process – all specific goals in the process area are fulfilled

GG 2 Managed process – institutionalized managed process

GG 3 Defined process – institutionalized defined process

Figure 2 Structure of CMMI-DEV goal (number of element is illustratory), created by the author

Maturity levels are evaluated by the reached achievements on the specific goals of each predefined set of process areas.[88] From the brief descriptions, maturity levels provide more room to improve for organizations that have already established set of processes. Especially level 4 and level 5, which provides foundation for internal analytics and statistical reports for a company to optimize its processes and products.

Process areas defined in CMMI-DEV framework are divided based on their function to these parts:

Process management process areas are classified as high maturity process areas which are used only in the staged path. Continuous path progresses through other areas. Once it reaches capability level 3 on these, it can move to the following high maturity process areas:

Organizational Process Focus (OPF)

plans, implements and deploys process improvements after understanding organization's strengths and weaknesses

Organizational Process Definition (OPD)

determines and maintains organization's standard processes, work environment standards and other assets based on the organization's process needs

[87] CMMI Product Team. *CMMI for Development, Version 1.3* (CMU/SEI-2010-TR-033). 2010, p. 27-29.
[88] Ibid, p. 26-27.

Organizational Training (OT)
identifies and conducts trainings for skills and methods needed to maintain identified processes and standards
Organizational Process Performance (OPP) – advanced process areas
derives and analyses quantitative objectives for process' quality and performance against business objectives
Organizational Performance Management (OPM) – advanced process areas
is proactive management of organization's performance to meet its objectives bases on the OPP analysis

Project management process areas:
Project Planning (PP)
defines project plan including activities, stakeholders, and scope
Project Monitoring and Control (PMC)
ensures monitoring and controlling activities, while also taking corrective actions. Progress is measured by comparing project status to the plan.
Requirements Management (REQM)
is fundamental to the engineering processes as it ensures any changes of requirements are reflected in project plans and activities
Supplier Agreement Management (SAM)
manages suppliers needed to satisfy project requirements. This includes tracking their work and progress, arranging and revising their agreements
Integrated Project Management (IPM) – advanced process area
establishes and maintains tailored processes derived from the organization's set of standard processes. All stakeholders coordinate their effort in the most effective way through coordination of shared issues
Risk Management (RSKM) – advanced process area
deals with risks across all activities of the organization in an coherent way with risks identification, assessment and mitigation strategy definition
Quantitative Project Management (QPM) – advanced process area
determines objectives for quality and process performance using statistical and other quantitative techniques

Engineers KPAs:
Requirements Development (RD)
is understanding and translation of customer's needs into product requirements
Technical Solution (TS)
is creation of software architecture, software components and design
Verification (VER)
ensures that produced solution fulfills all requirements defined in RD. It is usually set in coding process and is conducted by peer reviews of the code
Validation (VAL)
validates the functionality against customer needs

Product Integration (PI)
then takes care of integrating and delivering the new product to customers

Support process areas:
Measurement and Analysis (MA)
operates across all processes and provides guidelines by aligning measurement needs and objectives with a measurement approach
Process and Product Quality Assurance (PPQA)
supports all process areas as it provides practices to objectively evaluate performed processes, products or services against CMMI-DEV
Configuration Management (CM)
establishes and maintains integration of all work products using configuration identification, control, status accounting and audits. Examples can be plans, process descriptions, requirements, design data, drawings, product specifications, etc.
Casual Analysis and Resolution (CAR) – advanced support area
is an optimizing process area where a team identifies cases of selected results and define methods to prevent negative results and/or leverage positive results
Decision Analysis and Resolution (DAR) – advances support area
determines which issues should be formally evaluated and then proceeds to apply the formal evaluation process[89]

Continuous path allows organizations to select in which process areas it wishes to progress. Target levels can be set individually for each process area. If a set target level is reached in one process area, organization can either move to another one or continue progressing in this one. Staged path moves the respective set of process areas to higher maturity level simultaneously. Advancement is intertwined and sequential.[90]

2.6 Software process improvement with agile methodology

There are beliefs SPI and agile are contradictory. Mainly because SPI had been, for many years, tight with traditional models. Secondly, one of the four principles of agile methodology basically diminishes role of processes. However, some shortcomings of SPI models, even discussed in the former parts of this thesis, were accelerated by the rigidity of waterfall processes of software companies which accelerated SPI involvement in agile methods. The tailored SPI models for small companies also called for a more flexible and adaptable environment than the ones in large software project/organizations provide. Even the recent data show both approached to be used simultaneously. Although small in share (circa 10%), agile methodologies are considered to be one of growing trends in SPI studies.[91] It is not impossible to merge SPI traditional approaches with agile framework as it

[89] CMMI Product Team. *CMMI for Development, Version 1.3* (CMU/SEI-2010-TR-033). 2010, p. 42-63.
[90] Ibid.
[91] Kuhrmann, Marco, Philipp Diebold, and Jürgen Münch. 2016. "Software Process Improvement: A Systematic Mapping Study on the State of the Art.", p. 15, 23.

can be used in a hybrid model of both. Next part uncovers some of the studies using merges of both approaches.

One of the examples that implementing agile does not mean refusing traditional methods which have found successes in the past, can be Barry Boehm and Richard Turner's publication. There, they suggest a hybrid of two as they understand the shortcomings of agile methodologies. They describe their attempt to unify dispersed and misinterpreted agile practices into a coherent guideline for software companies as: "change from slow, reactive, adversarial, separated software and system engineering processes to unified concurrent processes"[92] Authors address frustrations of managers on difficulty of integrating agile processes into their software organizations bound with traditional processes. A bottom-line cause for such difficulties, is according to the authors, the contrast between "lightweight" process of agile methodology with traditional industry processes. On the one hand, the idea of agile methodology is agility and flexibility of the process, on the other, there have been years of system refining that have to be taken into account.[93] For that, authors identify three areas of where conflicts between these two worlds occur – development process, business process and people conflicts where they combine best practices of both.

Wang also provides a less distinctive description in differentiation between traditional and agile. He argues that while agile methodologies provide different guideline in using some of the traditional models' framework, they do not change their content and main principles.[94] In that sense, he makes a point for small sized software projects to apply agile methodologies, as it answers limitations they have in comparison to larger projects with bigger resources. It does not change the result in a quality software, it changes how the organization achieves such result.

Implementing agile methodologies in the journey of SPI does not mean abandoning traditional tools used in established models. A case for that is made by McAvoy and Butler in their research of the project management role in agile teams. The authors describe the importance of decision-making in agile development teams. As it lies in their philosophy and definition, agile teams share the responsibility of decisions-making throughout the whole project team, rather than just one individual. As such, socio-psychological forces come into play during decision-making process as in the agile teams, there is an emphasis on cohesiveness, trust and empowerment while sharing the collective responsibility for the delivery.[95] The authors' main focus point of research is to highlight the importance of a traditional project manager role in the agile teams which tend to be disorganized and the level of cohesiveness within the team can have contra productive effects on decision making.

A direct implementation of CMMI and agile methodology is attempted in the publication of Al-Tarawneh, Abdullah, and Alostad from 2010. The authors tailor both approaches for the need of small companies. The main premise is to continue using established CMMI framework combined

---

[92] Boehm, Barry, and Richard Turner. 2005. "Management Challenges To Implementing Agile Processes In Traditional Development Organizations". Online. *Ieee Software* 22 (5): 30-39. https://doi.org/10.1109/MS.2005.129, p.7.

[93] Ibid, p. 1-2.

[94] Wang, Lingfeng. 2007. "Agility counts in developing small-size software." *IEEE Potentials* 26 (6): 16-23. https://doi.org/10.1109/MPOT.2007.906114, p. 17-18.

[95] McAvoy, John, and Tom Butler. 2017. "The Role Of Project Management In Ineffective Decision Making Within Agile Software Development Projects". Online. *European Journal Of Information Systems* 18 (4): 372-383. https://doi.org/10.1057/ejis.2009.22, p. 373-376.

with light-weighted methodology that allows companies to produce high-quality software under pressure of ever-changing conditions. The authors not only believe both can co-exist, but rather they complement each other.[96]

In CMMI-DEV v.1.3, SEI itself developed mapping of process areas to organizations using agile methodology. This came as an answer to the growing number of organizations using agile methodologies in their development. Authors encourage own interpretation of process areas in agile environments as it depends on organization's context, business, project teams, etc.[97]

[96] Al-Tarawneh, Mejhem Yousef, Mohd Syazwan Abdullah, and Jasem Alostad. 2010. "Software Development Process Improvement Framework (SDPIF) for Small Software Development Firms (SSDFs)." *IJCSI International Journal of Computer Science Issues* 10 (1): 475-486, p. 476.
[97] CMMI Product Team. *CMMI for Development, Version 1.3* (CMU/SEI-2010-TR-033). 2010, p. 87-88.

PRACTICAL PART

3 CASE STARTUP

The following chapter discusses main conditions of the researched startup which shaped the methodology used in the proposed master thesis. The researched organization was founded in 2016. It has been operating in various industries as it aims to connect them with one, revolutionary solution. The following chapter introduces the researched organization (the thesis will refer to the organization as the startup from now on) from its:

- Business proposition and market position
- Software product and its complexity
- Financing
- Organizational structure and human resources state of the startup

All four points heavily affects the startup's and researcher's decision making in software process improvement and application of chosen methods. The next chapter, therefore, introduces the startup from these viewpoints and provides contextual background for the rest of the practical part.

3.1 Business proposition and market position

Facilities management, real estate and utilities industries are closely connected in their product and services, but very vaguely in operations, data transparency and communication with their customer. Moreover, we can find outdated processes heavily relied on manual actions or obsolete software tools such as Microsoft Excel. What the startup aims to provide is a complex software and hardware solution that will consolidate all their processes, databases and products while also creating new services they can offer to the end customers. Based on startups classifications, the researched startup can be identified as a scalable startup as it has ambitions to not only contain the whole domestic market, but also expand globally.

The startup operates on B2B market, with later ambitions to move to the B2C market. The integrated solution the startup aims to provides does not recognize a direct competition on the market. Organizations are either fragmented and provide only a partial solution in comparison to the services the startup provides, and/or are seeking partnerships with the startup. By the time research started, it had already contracted customers, which could be considered big names in the industry.

The startup, therefore, has a very strong foundation in business and sales department. Thanks to that, it has found itself in a very unique position in comparison to its counterparts. It is not the startup chasing the customers but the other way around. As mentioned above, there are already customers contracted. Unfortunately, due to its ambitious nature, the finalization of the product, or its MVP, has not been completed. There are two interconnected reasons for that. First of all, the product's scope is considerably large which needs either huge development team and resources or time. Which leads us to the second point - financing does not provide the conditions to hire such development team.

3.2 Financing

The startup falls into the category of self-funding. Up until today, the startup has been financed by the sole owner in what is called bootstrapping. It aims to be become self-sustaining by a direct sale model reaching a B2B market, but as mentioned earlier, while the product is not finished, there are no significant revenues flowing from external sources.

Strategic wise, startup uses model of experience curve. Experience curve showcases a situation, when direct costs per unit are significantly high in the beginning of the process, but decreases over-time with the cumulative volume of production.[98] As the startup's product is quite complex, its sole development requires high costs. A quite big development team is needed from the beginning to produce the system. These costs diminish within time when the product is already finished and launched. Implementation of the product is also costly with the first customer. The complexity of the software requires series of trainings, materials and documents for the users to be able to start using the system. These documents are with every additional customer only edited or adjusted, and costs are decreasing accordingly.

3.3 Software product

The startup provides software and hardware integrated solution. The developed software is a cloud-based ERP and billing system which provides integrated database, document management system and effective process workflows within. From the engineering point of view, the software product is based on object-oriented programming and is divided into microservices. Microservices architecture enables large applications to divide their structure to the collections of services. This allows them to be independently deployable and maintainable.[99] It is also architecture that can flex-ibly organize around business goals and capabilities, which, as mentioned in the above subchapter, are limited. That's why it is used in the researched startup as it can accommodate to the complexity of software but also to the ambitious business model and limited resources of the startup.

The system is divided into modules from the business perspective as well. Said microservices group into coherent parts of the system. The startup had finished 3 of eight modules for the core functionality by the time this research started. On the top of core functionality, the system aims to develop set of additional services such as communication channel within the system.

In the next development stages, the startup aims to progress the ERP web-based software and connect it to the mobile application for Android and iOS operating systems.

As the system was internal product of the startup and not a custom software made for any particular customer, the role of a customer was played by the CEO. CEO had spent multiple years consulting and discussing the needs of the potential clients and collected their needs and require-ments.

---

[98] Ghemawat, Pankaj. 1985. "Building Strategy on the Experience Curve." Harvard Business Review. https://hbr.org/1985/03/building-strategy-on-the-experience-curve.
[99] "What are Microservices?" 2021. Microservices Architecture. https://microservices.io/.

3.4 Internal structure

The internal structure of the startup has changed few times throughout the research. At the beginning, the startup was extremely centralized around its founder and CEO as he was involved in every stage of the production and other operations in the startup. Besides software department, the startup consists of hardware department. As the topic of the research is software process, the following subchapter focuses on internal organization of a software team.

The analyzed software team was organized under the Head of Software department (HoSW). However, as mentioned earlier, the CEO was involved in operations and work division. That means the startup's structure resembled a flat organization structure which is usually common in starting companies. In the researched team, there were no formalized positions of the software engineers, nor were there any software roles defined between them. Outside HoSW, no one in the team held a different position than their peers. This changed in the beginning of the research when the researcher was hired as a Project manager to apply SPI tools, streamline the production along with it and facilitate the day-to-day operations within the team.

There are three divisions line in software the team. First one is based on functional division, where the team consisted of: 2 front-end developers, 3 full-stack developers, 6 back-end developers, 1 software architect and 1 tester. The HoSW served as a manager but was also involved in software engineering itself either from the side of a software architect or a software developer. The second division line went along the capacity of the team as there were 5 part-timers in comparison to 8 full-timers. The last one was geographical. While 3 members worked on site where startup was located, the rest was dispersed in the Czech Republic and Slovakia.

These division lines imply that the software team was far from being homogenous. Along with other preconditions discussed in this chapter, it played an integral role in decision making during the SPI process. This, along with the overall research approach, is identified in the next chapter.

4 METHODOLOGY

The main hypothesis researched in the proposed thesis is:

H: A SPI model can be used to streamline processes in a startup company if used with appropriate and tailored agile and project management methods.

The hypothesis is built on Sutton's assumptions and recommendation on integrating software process methods and tools to the specific startup environment. The methods should not be implemented as a rigid set, but rather answer to specific needs of the startup. This master thesis can be defined as a qualitative case study focusing on one startup company developing a software product. As the thesis works with an initial hypothesis and identified certain pattern, this research falls into

the category of instrumental case studies. Such study aims to build or extend a theory or to examine a certain phenomenon.[100]

It can be argued that positionality is a central component in qualitative methods of social studies. What is the role and relationship of a researcher and the ones researched can impact process and also quality of data collection.[101] The researcher had vital part in shaping the process and was therefore not an outsider in the process. The research could be, therefore, conducted in a more sensitive and responsive manner. There were also not any specific time window limitations during which researcher had access to the researched group, as can be the case in qualitative studies. No processes or data slipped through and the research was conducted without these constraints.

The software team was researched for 12 months. During the period, the researcher fully utilized observation method as a primary data source. Detailed notes were reported and analyzed throughout the process. For the process assessment part, the researcher performed unstructured individual and collective interviews.

For individual interview, question asked was: "What would be the one thing you wished to change except money and your boss?".

For collective workshop, the questions asked were:

"What are we doing good?"

"What are we doing bad?"

"What we should start doing?"

Theories laid out in the previous chapters were used for implementation plan definition and analysis and interpretation of discoveries. The results of the improved software process were collected in set of documentation and progress reports, which can be considered as secondary source of research data. The final source of data was synthesis of findings summarized in chapter 7.

Constraints of the startup shaped the used methodology. Bootstrapping model of financing puts a significant pressure on the CEO and the whole team. This means time and money were very valuable assets in startups. Therefore, there could not be significant time or financial resources reserved for the SPI. Traditional or large in scales model like CMM or CMMI were eliminated in the beginning. For the research, combination of models was implemented in order to meet software team's needs.

First of all, PRISMS model was chosen to navigate individual steps of SPI. Reasons for PRISMS model were, foremost, its application on small companies. The unique business position of the startup where it is customers who are waiting for the product to be finished, business vision and goals play an essential focus in the startup's decision-making. PRISMS model places business objectives in the shaping process of SPI model implemented.

Unlike other models, PRISMS is one of the few that include development team to the process areas definition. Startups usually tend to merge separated roles in software development. In this

---

[100] McAvoy, John, and Tom Butler. 2009, p. 32.

[101] Ganga, Deianira, and Sam Scott. 2006. "Cultural "Insiders" and the Issue of Positionality in Qualitative Migration Research: Moving "Across" and Moving "Along" Researcher-Participant Divides." *FORUM:QUALITATIVE SOCIAL RESEARCH SOZIALFORSCHUNG* 7 (3). https://www.researchgate.net/publication/49913614_Cultural_Insiders_and_the_Issue_of_Positionality_in_Qualitative_Migration_Research_Moving_Across_and_Moving_Along_Researcher-Participant_Divides.

sense, startup software developers are more independent and have bigger insight to the companies' objectives, functioning and priorities. The flat organizational structure of the startup supports this phenomenon. Including developers in the assessment process is not only beneficial, but integral in order to correctly evaluate the capability level of the team. For this, a partial hypothesis was formed:

H1: Startup developers have an integral role in software development and their insights are crucial in software process assessment.
The research question to answer this hypothesis is:
R1:What are the differences in software process evaluation between developers and business?

Although PRISMS model primarily uses CMM process areas in the step of defining key process areas, for the purpose of this thesis, CMMI-DEV framework will be applied instead. As discussed in theoretical overview, CMMI-DEV provides similar framework for small businesses, while building on foundation of CMM. Another benefit of this framework's version is its continuous path which does not apply levels the whole scale of software processed but rather chooses objectives of each process area. It also allows to prioritize certain process areas at the expense of those deemed as less important. This factor is essential for the startup, as it can continue functioning while gradually applying prioritized process improvements. A cohesive process improvement suggested by CMM or CMMI-DEV staged path is not only something a startup cannot afford, but also something startup does not have time for. That is why, instead of CMM framework used in PRISMS model, the thesis applies CMMI-DEV guidelines. Metrics for the process areas were also not determined as the thesis uses a hybrid model between of PRISMS and CMMI-DEV. In CMMI-DEV there are already generic goals defined to assess the process improvement which will be followed in evaluating the SPI.

These assumptions were formalized in the next partial hypothesis as follows:

H2: With limited resources, startup companies have to tailor established models and practices to streamline their process without big investments.
In researching second partial hypothesis, the following question will be asked:
R2: What are the tools to minimize financial investments needed for SPI?

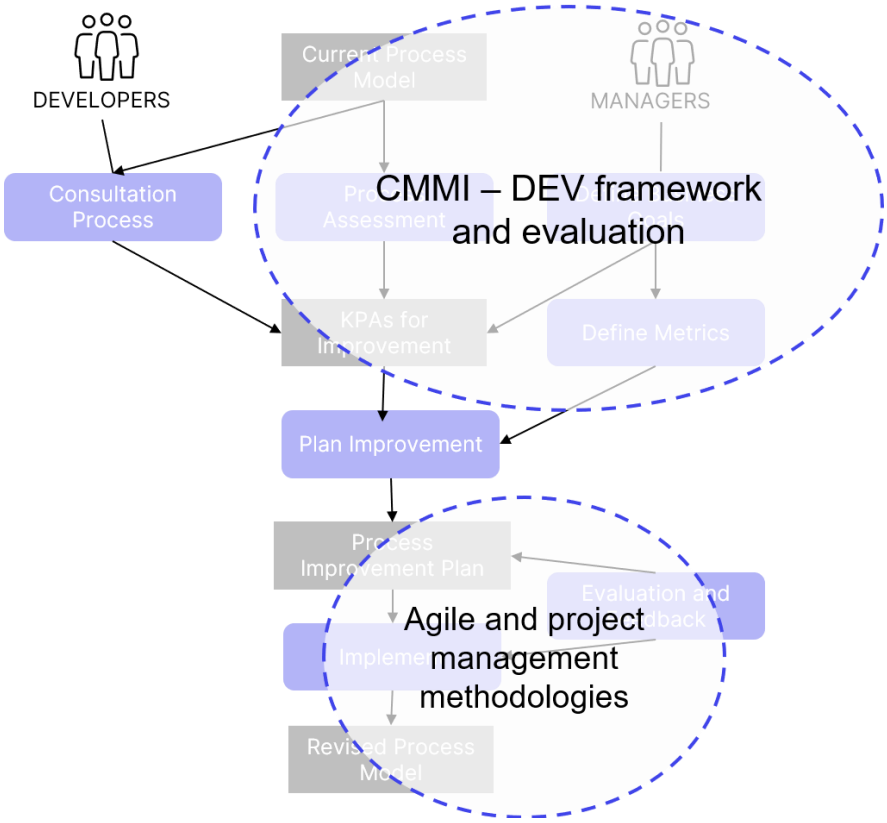
Once the current process is assessed and key process areas are defined and prioritized, there is a need to define an improvement plan for these processes. For improvement part, this research applies mix of project management and agile methodologies. Rationalization behind that is as follows. A traditional project management is an essential component to both SPI and CMMI-DEV. And that is where limitations of researched startup come into place as it cannot rely on waterfall, sequential model. As mentioned above, the startup is in a situation when the contracted customers eagerly await the product. With an ambitious product not nearly finished, a process streamline has to rely on agile methodologies and iterative model of development. Otherwise, startup would lose its customers. Agile development is recommended method when a product/system faces certain levels of uncertainty.[102] This condition leads us to the last partial hypothesis:

[102] Malakar, Sudipta. 2021. *Agile Methodologies In-Depth,* p. 56.

H3: A business model of researched startup defines which tools for software improvement can be used in the process.
R3: How do the contextual factors of an organization influence software processes in a company?

Next part of the thesis will attempt to prove these hypotheses using the empirical findings of the researcher during 14 months on a project. Figure below shows how did this thesis combined the described models and frameworks.

Figure 3 Visual representation of methodologies used in the thesis, based on PRISMS model, CMMI-DEV and Agile and project management meethods, created by the author



5 SPI IN A CHOSEN ORGANIZATION

The fifth chapter of this thesis moves on to the application of chosen SPI model on the startup. It is divided to subchapters based on the steps undertaken in the process.

5.1 Business goals definition

Business model of the startup was outlined in the previous chapters. In the beginning of the research, the startup has found itself committing to certain deadlines of the finished functionality which it was unable to fulfill. The contracted customers were not be able to pay for the services and the startup had to continue in the bootstrapping financing model.

The complexity of the product made in impossible to wait for the product to finish to count the on the sales profit from customers. The business decided to build the business model on partial sales deploying the product modularly.

Another management's question mark was how effective the process is and how utilized the human and financial resources in the software team are. The cost optimization is a must in startups operating under pressure of time and money variables. The ideal situation is to distribute the costs over time and not have any individual increases at once.

Considering this, the researcher defined business goals with the CEO as follows:

- Finish the core functionality of the product in 6 months
- Be able to set deadlines for finished functionalities for the customer
- Validate the software functionalities during its SDLC
- Monitor costs spent on software development
- Improve software quality
- Minimize costs spent on SPI

Important thing to point out is the last point. Even though the startup was in need of process improvement and optimalization, the costs used for that should be minimized from both development and business side. Continuous path of CMMI-DEV was therefore used, which can implement process improvements over time based on initial prioritization.

5.2 Process assessment and consultation process

Second step in the PRISMS SPI model is to undergo a process assessment. The existing process model of the startup lacked any definition and form. Rather than software process, the startup was performing a set of activities. The software team operated on an ad-hoc basis and was initially at the capability level 0 of CMMI-DEV framework.

5.2.1 Process outline

Through basic stages of SDLC, next part lays down how were the activities initially performed.

Planning had two layers. First one, high-level plan of the whole product based on the modular division. The high-level plan was conducted with the client to outline the implementation steps. However, no one in the team really knew how much time it would take to finish the specific modules. The complexity of functionalities was not broken down to smaller pieces and estimates were set incorrectly. Short-term planning was done every week in the composition of CEO, HoSW, senior frontend developer and senior backend developer. Planning was an unstructured meeting where there were discussed list of pending tasks across the system. On few occasions, planning was even cancelled, and the team was assigned tasks ad-hoc.

Defining (requirements) was being done only on the surface. CEO was the one giving the software team overall vision for the functionalities which were recorded in bullet points in the internal wiki. From these bullet points, set of development tasks were created in a task management online tool.

Design was not part of software activities. Outlined requirements on functionalities were then vaguely described in form of development tasks and developed. The software architect, who was part of the team, was independently analyzing technical solution of the next module to be implemented, outside of the SDLC of the team.

Coding was performed individually and in ad-hoc manner. There were several impediments during the process. Firstly, as the product has complex architecture, every new added feature could and did disrupt the existing functionality. Secondly, the creativity of a developer without undefined scope could be dangerous in creating inconsistencies in the product and in the system. Finally, as the scope of the task was not defined clearly, the process of coding was prolonged due to developer either asking about specifics or procrastinating.

There were no unit nor integral tests. The tester in the team conducted manual tests on the frontend part of the functionality and collected bugs found to the task management tool.

Deployment had no certain rules or practices. By the time research started, two already developed modules were in the process of deployment. Deployment was two dimensional – first to the testing environment for the customer, second to the production environment for the customer. Due to performance issues after deployment to the testing environment, unfinished functionalities or bugs found, the planned deployment of the modules had already been delayed on two occasions.

Maintenance bugs were reported only through the tester and had no prioritization. They were put to the task management tool without structure, where they were piling up together with some archaic or unfinished tasks.

As for the communication, the team communicated daily on daily standups used in Scrum model. The attendance on the standups was random, there were days where it was even under 50%. The structure of standups was ineffective as it was individuals talking in an unstructured manner, and the meeting was usually prolonged to 30 minutes.

The activities were also not producing desirable results and fell short on meeting business milestones.

5.2.2 Consultation process

Through various of collective and individual unstructured interviews, the team was asked on the current process and pain points they wished to change.

The general consensus on a pain point was the lack of specifications and clear architecture. The work that was assigned to them was only outlined, and it was up to developers to figure out the details. Some might consider it a positive room for creative process of each individual developers, but the consensus in the team was that in such a complex system and interconnected architecture, it was hard to effectively finish the tasks.

An important topic of the consultation was the planning. The team felt unnecessary pressure as the deadlines for functionalities were set by the business without consulting the development team or without properly estimating the complexity of the module. The development team was then pressured to finish the functionalities on set deadline which were unable to be finished in a defined time scope.

The ineffectiveness on the process was implied by the other two points. First one was ineffective time spent on standups. The team was comprised of 15 people. There were standups, when the discussion between two-three people exceeded 15 minutes and scope of what is supposed to be said on standups. The whole team of 15 people then spent time listening to overextended discussions which could have been avoided had the discussion participants took it off call.

Second activity pointing to ineffectiveness was work distribution. As it had no method, developers started the day without knowing on what tasks they will work on. As the startup held flat structure, there were clear no hierarchies set to distribute the work and no clear point of contact to ask for one.

Outside of the points discussed in subchapter 5.1, the business was also consulted on the pain points of the software process. The frustration from the team not meeting deadline was also voiced from the side of business. Additional points were lack of transparency of team's delivery and somewhat too creative process of functionalities deployment as it was full of inconsistencies and bugs. The business put emphasis on increasing software quality to minimize the number of bugs and performance issues.

Table 3 summarizes the pain points consulted with the software team and the business. Cells highlighted with green are the pain points which were identified by both parties. Cells in grey only with one of the asked.

Table 3 Comparison of pain points identified by the business and development team

Pain point	Development team	Business
Lack of product specifications	X	X
Lack of clearly structured technical architecture	X	
No planning	X	X
Impossible deadlines	X	
Deadlines not met		X
No validation of the product specifications with the customer		X
Lack of testing which led to cost increase		X
Ad hoc task distribution	X	
Unclear task management	X	
Inefficient standups	X	
Unclear time reporting of the team		X
Lack of accountability for the code		X

As seen from the table, there were only three points characterized by both the development team and business. Other points were either shared from business side or from development team side. It can be seen the business had sales and costs optimization on mind, while the development team focused on their daily operations and struggles. The summarized pain points had several business implications:

- Inconsistent software
- Zero integration of complex technical solutions
- Not stable solutions causing inconsistencies and bugs
- Confused, non-productive team with inability to fully utilized the team members
- Ever-delayed delivery
- Inability to make agreements with the clients
- NO PROFIT

5.3 KPAs for improvement

Based on the process assessment and consultations with the team, there were process areas from CMMI-DEV framework prioritized. As PRISMS and CMMI-DEV says itself, the process areas in continuous path should be chosen and prioritized based on individual needs of each organization. After determining the main pain points, they were put in their respective process area. Each of the KPAs was according to CMMI-DEV set with a target capability level and according to PRISMS model with a priority.

The pain points identified were subjected to the process area which should mitigate or eliminate the pain points as follows:

Table 4 Mapping of the pain points identified by the startup and the CMMI-DEV process area

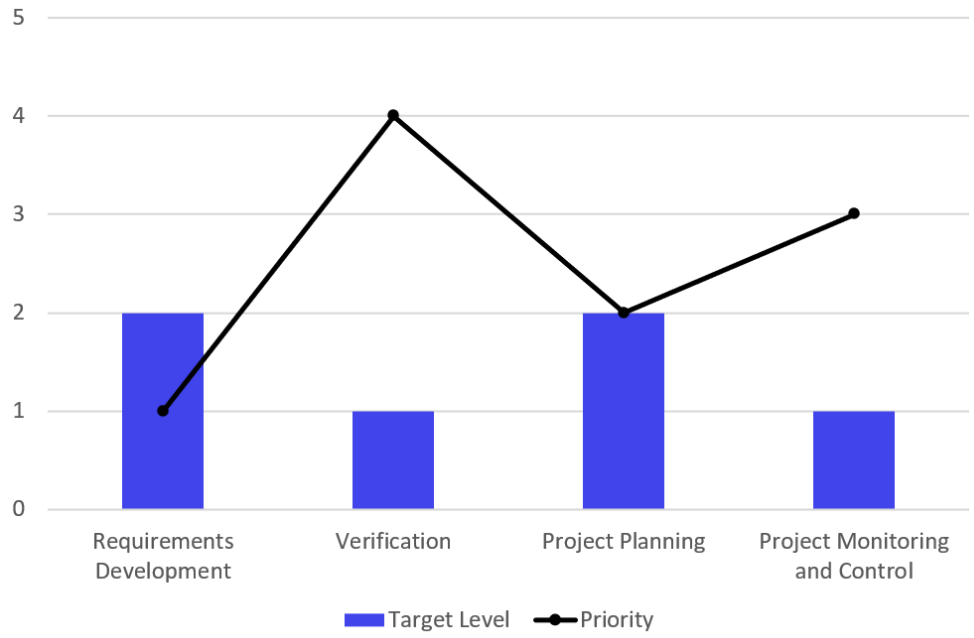
Pain points identified in the researched startup	Process area of CMMI-DEV framework
Lack of product specifications	Requirements Development
Lack of clearly structured technical architecture	Requirements Development
No planning	Project Planning
Impossible deadlines	Project Planning
Deadlines not met	Project Planning
No validation of the product specifications with the customer	Requirements Development
Lack of testing	Verification
Ad hoc task distribution	Project Planning
Unclear task management	Project Monitoring and Control
Inefficient standups	Project Monitoring and Control
Unclear time reporting of the team	Project Monitoring and Control
Lack of accountability for the code	Project Planning

The target level and priority were set considering these variables. The target level for capability model was set to Performed (Level 1) as the minimum improvement the startup had to undergo.

For the two areas both business and developers identified as required (see table 3), there was target capability level set to Managed (Level 2) to address this urgency.

As for priorities, the one tied directly tied to engineering process was preferred the most – Requirements Development. The reason for that is importance of business goals in this model. The business is product-oriented and prioritizes production process – software engineering to work as effective as possible. The project management KPAs followed as they were dependent on the first two process areas. An organization cannot plan, if the workload is not estimated. And the software team cannot estimate if it does not have clear requirements and technical architecture of the product.

Figure 4 List of KPAs identified for SPI of the researched startup, the target capability level of the SPI and priority

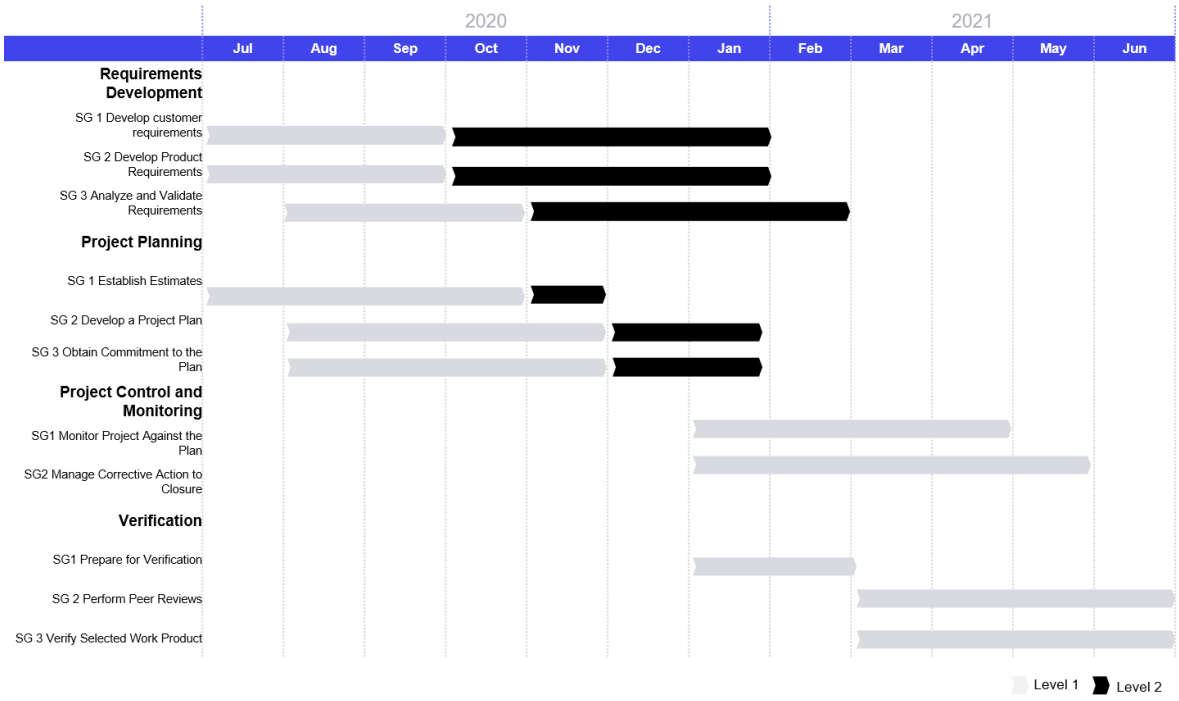


5.4 Improvement plan

To minimize the costs, activities to implement the improvements were dispersed into 12 months utilizing the continuous path of CMMI-DEV framework. The process areas were broken down based on the specific activities defined for each process area. For all process areas, level 1 was required, which meant all specific goals of the defined areas had to be achieved. This was prioritized in the improvement plan as well. Moving to Level 2 in Requirements Development and Project Planning was deprioritized and moved to the later parts of the plan.

The specific activities of process areas were performed based on the process area's determined priority.

Figure 5 Improvement plan for the researched startup based on PRISMS and CMMI-DEV



As the thesis follows agile and incremental development, the performed processes are also incremental with every SDLC phase. The specific activity is considered performed when it is regularly performed in the development process.

Due to importance of the Requirements Development and Project Planning area, their advancement to Level 2 was prioritized against Level 1 of Project Control and Monitoring. However, that does not mean the team was not conducting some activities from these process areas. It was only not considered to obligatory perform all of them on either regular, or institutional basis.

Moving the processes to Level 1 in Requirement Development and Project Planning areas was performed simultaneously due to their interdependence and same importance to the business objectives. Had they been done separately; the length might have been shorter. However, both relied on compliance from all software team members, including CEO. Among daily activities, and code production, the SPI activities also take up some portion of team's working hours. Thus, these progressions were laid out to 4-5 months considering these variables.

Level 2 for Project Planning was allocated much shorter time than for Requirements Development. The researcher assumed that specific activities in Project Planning were in competence on 1-2 people and their institutionalization could therefore be streamlined in comparison to Requirements Development which involved the whole software team and CEO.

The rest of the process areas were planned for the second half of the SPI process. The plan also counted on impediments in the process and reserved buffer time for first two process areas.

5.5 Plan implementation

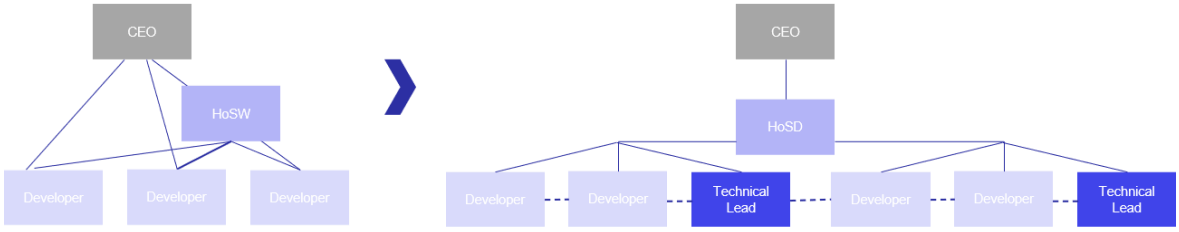
In order to monitor and manage the changes and improvements of the SPI, one of the integral parts was the team distribution. The software team was separated into micro teams. Firstly, the team composition suited the complex architecture of the microservices better. Each micro team could

focus on specific microservice and progress the module independently from the other parts. Secondly, the process of development had to be streamlined. While the modules were complex, the work on the modules could not be parallelized between 12 software developers. Sometimes three developers were working on their tasks and the rest was waiting for those tasks to be finished before they could start their own. The startup was losing one of the precious variables – time. Thirdly, by dividing the team to micro teams, two or three modules could be progressing simultaneously. Faster the production, faster the product launch and profit from the sales.

This restructure was supported by putting one more hierarchical level between developers and HoSW and PM – a technical lead who became responsible for the produced code. Another point to highlight was that team started to become more homogenous. During the 12 months, there were some members leaving, while some new developers arriving. The startup decided to hire only full-time workers and the ratio of full-time workers increased significantly. Pandemic of COVID resulted in full remote communication for all of the members regardless of their location. After HoSW left the startup, PM changed her role to Head of Software Delivery (HoSD) to be a sole bearer of knowledge of operations and process flows. This increased efficiency in decision making.

The organization then changed from flat to flatter structure as Figure 6 demonstrates (the number of people is only illustrational). This structure was chosen to accommodate to collective responsibility of XP model in developers. By creating a flat structure, but with a certain hierarchy, the startup could keep the team encouraged, interconnected and engaged, while also preventing confusions on reporting and accountability as in the flat structure.

Figure 6 Change of organizational structure in the startup during SPI



The process was, besides internal team, reviewed by two external consultants – one focused on the startup's business model and growth, and external consultant company – focused on the software internal processes.

Following parts lay down each of the defined key process areas and improvements that were implemented in the SPI process. Tools and best practices of agile methodology and project management were used to perform specific activities in the identified process areas. These tools were individually chosen for the startup's purposes by the researcher (who was a PM, later Head of Software Delivery), external consultant company, CEO, HosW and the development team itself.

5.5.1 Requirements Development

SG1 Develop Customer Requirements

In case of developing customer requirements, the requirements had been collected for years by the CEO. The first step was to utilize this experience and formalize the requirements, or requests, as was the used title for requirements in the startup. They started to be registered in a task management tool

They were given a customized form of registering them. The examples of a registered request can be seen in Appendix B. The required information a requirement had to have registered were:

- High-level overview of the requirement
- Who is requesting the functionality
- What are the users the requested functionality will affect
- Context on the requested functionality – why is the user requesting this functionality
- Business relevance to the customer
- Urgency
- Detail of the requested functionality e.g. screenshots
- Business relevance to the startup
- Note of the software team

Requests were then moved along the defined workflow from new, to consideration, to approved, to analysis, to development and to completed. Additional statuses like deferred and will not implement were also part of the defined workflow. The movement to next stages were conducted on Prioritization meetings which were attended by the CEO, HoSW and Project Manager (PM) on a biweekly basis. Besides moving the requirements to further stages, it was reviewed how many requirements proceeded to the development, how many are done, etc.

Later during the implementation, a new role entered this process area - Consulting manager, whose role was to map the current state of customer's processes and prepare the environment for the software launch. As he gathered significant understanding of customer's needs, part of his time was allocated to customer requirements identification. Clients and users were involved in the process through Consulting manager, or through feedback meetings reviewing the functionalities.

Another form of institutionalization was the guideline provided for request registration and identification of workflows as can be demonstrated in Appendix C.

SG2 Develop Product Requirements

After a request was moved to the status of Analysis, two parts of analysis occurred: User story and Technical architecture. Both the development team and the CEO identified the need to have specific definition of the task, how it should function and how it fits to the system architecture.

Concept of user stories and technical architecture was applied from XP framework User stories also followed a customized format of:

- Basic format – uses standard format of a user story
- Business case – provides business reasoning behind the functionality
- Prerequisites – what are functionalities that needs to be implemented before
- Acceptance Criteria – how should the functionality look like and against what will be tested
- Connected user stories
- Appendices – usually link to a prototype
- Architecture – usually link to the part of architecture in the internal wiki

User stories were formed into larger functionalities in a card view format. Every user story was then prioritized based on its scope into releases of certain module. The coherent card view of the partial modules in the system was included in Appendix D.

User stories were then subject of technical analysis which integrated newly requested functionalities into the system architecture. This step was implemented in order to accommodate to the complex architecture of the startup's product and to modular business development plan. Technical architecture consisted of defined data models, API definitions (Application Programming Interface), communication with other modules, etc. A software architect would then break down the architecture into tasks for developer which were added to the task management tool. Both user stories and technical solution were estimated with involved stakeholders and put on the product plan. In the process, CEO and/or HW department of the startup consulted and verified the expected behavior of some functionalities.

Due to the delicate character of both user story and software architecture, it being an intellectual property of the startup, no example is available in this thesis.

After implementation of these methods, the team evaluated process to be time taxing, but rewarding. For the business analysis and user stories definition, there was a Product Manager hired three months after SPI process started. However, for such a small team to have five people (CEO, HoSW, Consulting Manager, Product Manager and Project Manager) included in the process, it was determined as ineffective after two months. Due also costs optimization reasons, the role of a Product Manager was abolished. PM was assigned to continue developing the user stories as she had before, and it became an official part of her workload.

For the technical solution, one of the senior backend developers was reallocated from development and HoSW was also redirected from business requirements to translating them into technical analysis and design integration to strengthen the area of software architecture.
Formal guidelines for both user stories and technical architecture were laid out in the later stages of the improvement plan. Guideline of a user story is attached in Appendix E.

SG3: Analyze and Validate the Requirements
Two methods were applied for the last specific goal. Customer requirements were defined by the business (CEO, Consulting Manager, PM), user stories created by the PM and technical architecture by the software architects. In order for the developers, who are the receivers of these work products to have a say in the process, all these materials are sent to them beforehand. They are provided with a platform to comment on the parts of the documents they do not understand or provide different perspective. The findings are then discussed on a grooming meeting – accommodated from Scrum methodology – and analysis is being verified.

Second method was verification of the requirements with the customer which was taken from the Prototype model of the SDLC. Together with the user stories and technical architecture, a prototype model of an analyzed product was created. The wireframe was created using the tool Figma and was

practically a copy of a system with basic functions which where templated to mimic the functionality. Prototype was then validated with the customer – CEO and/or HW department, in appropriate time before development of the prototyped functionality started. On this validation, CEO could express his comments so that the rest of the team could edit the user stories or architecture accordingly.

Prototypes were later used with the client as well, to showcase the planned functionality and gather their feedback before the functionality is finished to make final adjustments. The prototyping has also shown its benefits but due to limited capacities could not be conducted on every new set of functionalities. Five months after the process a Product Designer was hired to institutionalize the process. Both designer kit and UX components guidelines have been completed and added to the internal wikipedia of the startup.

5.5.2 Project Planning

SG1: Establish Estimates

The revised process of requirements definition allowed estimations to be set. As mentioned above, grooming meeting became part of the development process where the development team discusses newly specified features. The main aim of the grooming was to estimate the user stories which were broken down to development tasks. Format of the grooming adopted poker planning from XP. Developers used online software to vote how much do they think it will take to produce a certain task. Everyone votes individually, and only after each member votes, the points are revealed to the group. If the points differ significantly, there is a discussion held. Developers can change their votes until the final confirmation is set and average of all points is assigned to the groomed task.

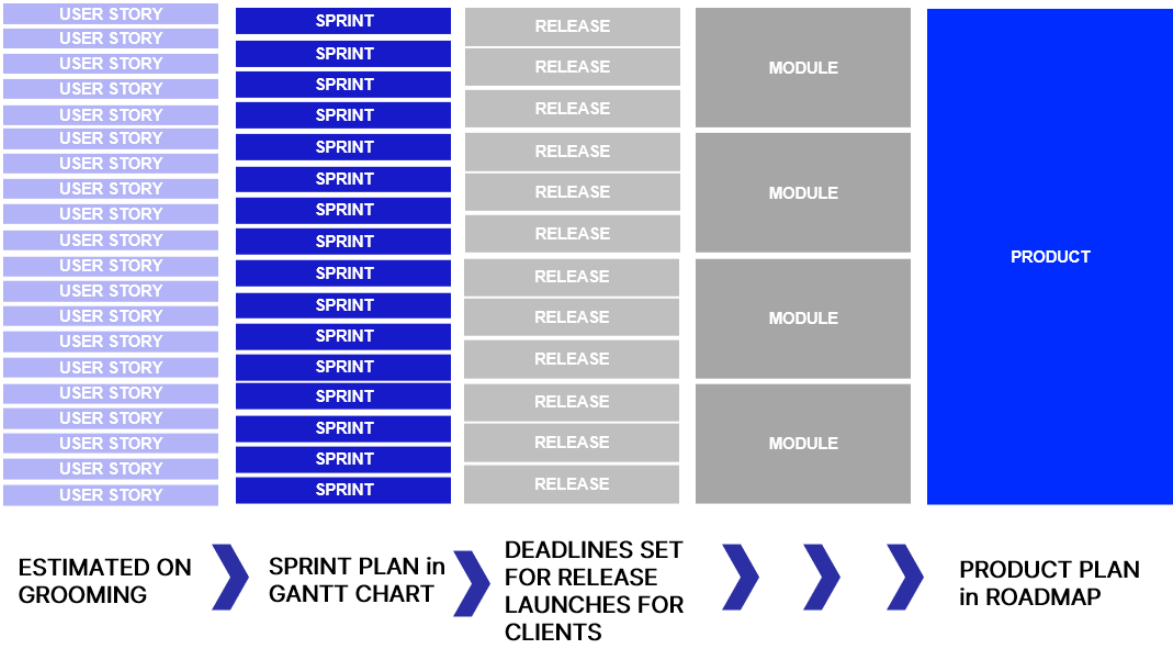
Grooming became a part of every new set of functionalities in the development process. The PM was moderating the sessions and recording the estimates in the task management tool. An example of a groomed task and the online software tool used can be seen in Appendix F.

SG2: Develop a Project Plan

CMMI-DEV uses sprint as a replacement for project in agile environments. The startup accommodated two weeks springs from Scrum. The modules were divided into releases. Releases were groomed and estimated. Tasks from releases were put in a Gantt chart based on their estimation and team assignation. Each developer was assessed based on their pace and capacity (as some were working part-time). A real example of a Gantt charted sprint is shown in Appendix G.

Once releases were estimated, the whole module was scheduled in the product plan. The product plan was a high-level roadmap of the product. Each module had three activities planned – business analysis (user stories and prototype validation), technical architecture and development (estimated development tasks). The whole process was in competence of the PM. The illustration of the divisions used in developing the Sprint and Product plan is shown on Figure 7:

Figure 7 Division of work structures used for Sprint plan and Product plan



SG3: Obtain Commitment to the Plan

A commitment to the plan is in sprint environments tasks assignation. This was constitutionalized by two tools. Firstly, directly on the grooming meeting, after their estimations, tasks were assigned to the developers. Secondly, if tasks were too big and not assigned on the grooming, technical lead was assigning the tasks to his micro team himself. The assignations were then recorded in the task management tool.

The commitments of both technical leads and developers in this activity was in the later stages confirmed by the detailed description of team roles responsibility which was conducted by the PM. The need for the clear definition of responsibilities was accelerated with the need to institutionalize this specific activity. Both formalizations are shown in Appendix H.

5.5.3 Project Monitoring and Control

SG1 Monitor the Project Against the Plan

There were several tools to perform and/or improve monitoring of the process, plan and team.

Firstly, task management were formalized adopting Kanban methods. Each sprint started to have its own Kanban board and a precise definition of moving the cards to the next status provided a visual support to the sprint monitoring. Appendix I showcases a diagram followed by the team to move their tasks. Appendix J is a real example of one of the sprints in Kanban board.

Secondly, answering one of the pain points of the developers, daily standups were revised and restructured. Guidelines on how to lead the standup, what to say on standups and how to prepare for standups were set (Appendix K). Technical leads were leading their standups to streamline the whole meeting as they knew about blockers of their teams beforehand and could swiftly address them

without prolonging the discussions. Lastly, standups started to have visual representation adopting Kanban boards.

Thirdly, new task management tool – Goodday – allowed each task to have its own time spent by a developer on a task. These partial time reports were then summarized reports & analytics tool of Goodday. A quick way to review reported time helped monitor the sprint and its progress. It also helped to compare the reported time to estimated time of the task.

All monitoring and control tools were then put in a weekly report of each module. The report used comparison method on estimated hours to develop the functionality vs. real hours it took to develop the functionality (Appendix L).

SG2 Manage Corrective Action to Closure
Due to incremental and dynamic environment of the startup, corrective actions were implemented daily and on three levels.

Firstly, on the sprint level. Comparing the estimated progress vs. the real progress as mentioned above was conducted during sprint. If sprint was behind, resource allocations were made during the sprint based on importance.

Secondly, on the product level. All changes to requirements or other unexpected events (e.g. refactoring the old code because of inconsistencies caused in the previous projects) were reflected in the product plan and deadlines immediately moved or reconsidered.

Finally, on the process level. Team adopted sprint retrospective method of Scrum and held retrospective discussions on the improved processes. Some of the corrections spurring from these retrospectives were for example clear list of competences in the team (e.g. Appendix H).

5.5.4 Verification

SG1 Prepare for Verification
Verification was conducted on every development task and part of the code. The preparation was therefore done during the process of grooming as that is when the tasks assigned to a spring were named.

SG2 Perform Peer Reviews
Implementing XP methods, peer reviews became part of the development process. Firstly, by formalizing technical lead role who had to review every part of the code his team members produced. In later stages it has also become practice for other developers in the team to review the code of their peer. This was however not enforced.

SG3 Verify Selected Work Product
The first level of verification was done by a developer, who produced the code as it was enforced by technical leads to perform unit tests on every new functionality. This was also reflected in the

grooming stage when the estimations of the functionalities increased because of considered time spent on writing unit tests. Unit tests on specific tasks took longer than the implementation itself.

The code then moved to the technical lead and other team members for the peer review and code review. Any comments of the technical lead had to be processed and corrected.

After the code review, the code moved to the testing environment where there were manual tests performed by the tester. The tester followed detailed acceptance criteria of a user story.

Final verification was conducted at the end of each functionality on so called DEMO meeting. On this, PM provided quick presentation of the functionality, each feature and review from the CEO and other stakeholders was recorded. This format was adopted from Scrum's ritual – Sprint review.

6 RESULTS OF THE SPI

The previous chapter described SPI process through their inputs and implementation. In the following one, the thesis proceeds with the results of this process. First, the revised process model is laid out with the impact it had on the startup. In the second subchapter, the process is evaluated against CMMI-DEV framework and their guidelines on process appraisals.

6.1 Revised process model

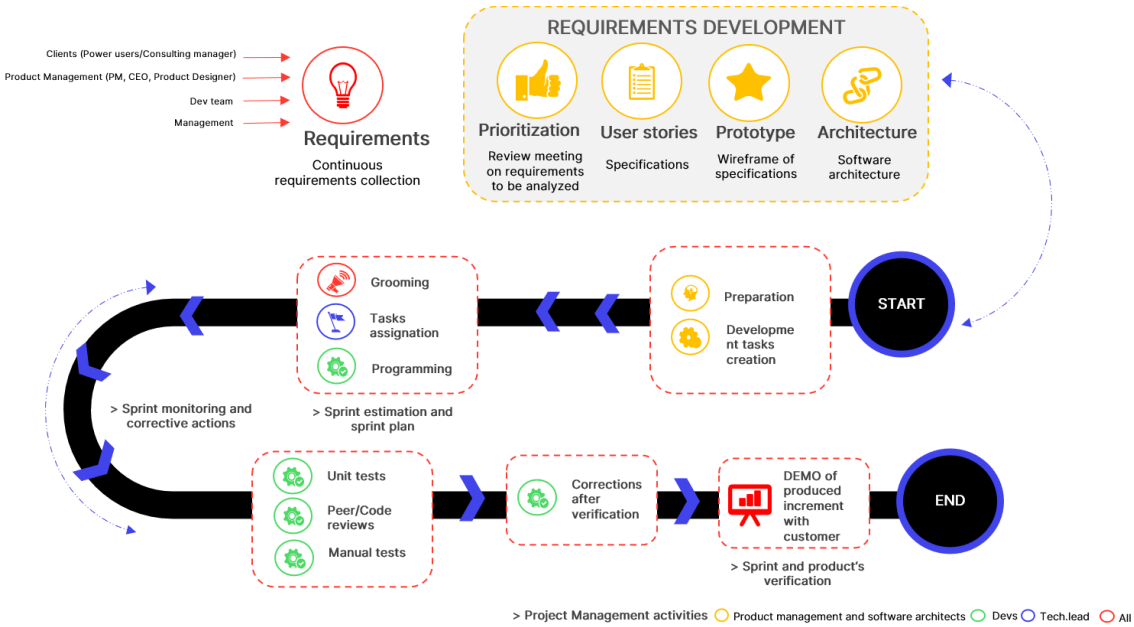
All improvements implemented had several positive impacts on the team and the process. The revised process model of software development can be seen on the Figure 8. This model is being followed by the startup since the improvements' implementation. It is also formalized in organizational policy and internal wiki.

Positive effects on the team is described below:

- Dividing the team to micro teams prevented this ineffectiveness and fully utilized each resource available
- Role of a technical lead enabled task management, code reviews and answered lack of responsibility or accountability for the produced code
- User stories resulted in no discrepancies between the produced functionality and customers' requirements
- This was supported by prototypes creation which served as a tool to visualize outcome of the functionality
- Technical architecture as part of the development process helped developers to have clear guidelines on the produced code. It drastically streamlined programming process.
- Groomings provided developers a platform to voice their opinions and be part of product shaping process
- Groomings allowed every task to be estimated based on developers' experience which led to detailed sprint planning and management

- Deadlines for the clients started to be set based on developers' estimations and PM's planning
- Kanban boards helped PM and the technical leads to have a visual representation on what is everyone working on, what is the amount of work left to be done and what was completed
- Unit tests, code reviews and peer reviews increased product's quality and decreased each functionality lifecycle

Figure 8 Revised process of software development in the researched startups after improvements implementation



To evaluate the business' satisfaction, we can go back to the business objectives set by CEO, majority of them were answered with the process improvement:

1. Finish the core functionality of the product in 6 months
   - Unfortunately, this objective could not be met as the complexity of the product turned out to be much more difficult to implement. A detailed list of specifications in form of user stories and detailed software architecture allowed the business to see how much time will be needed to complete the functionality. However, thanks to the SPI process, from 2 modules in three years, the software team implemented 4 modules in thirteen months. The velocity and tempo of production rapidly increased without significant increasements to development resources.
2. Be able to set deadlines for finished functionalities for the customer
   - Thanks to the estimations and sprint planning, the deadlines were more precise and reflected real complexity of the functionality.
3. Validate the software functionalities during its SDLC
   - Accommodating the prototype model, the business had a quick and effective framework to validate the analyzed functionality before it went to production. The discrepancies could therefore be discussed and corrected before it was too late and before time and financial resources were burnt.
4. Monitor costs spent on software development

- By requiring all developers to report their time under every development task, the management could monitor costs spend on functionalities and optimize them accordingly. This was not possible in the past as developers reported their time very vaguely and incoherently.

5. Improve software quality
   - Hard to measure improvement. However, multiple layers of tests – unit tests, code and peer reviews, manual tests – at least minimized number of returned tickets. Software quality got better also thanks to the specifications and software architecture which minimized inconsistencies in the software.
6. Minimize costs spent on SPI
   - A detailed comparison could not be made as there were no estimations on the SPI process and tailored SPI process beforehand. The researcher did not have access to all financial and costs data of the startup. SPI process not dealing a huge constraint on the financial stability of the startup is however a showcase of a acceptable costs/benefit ratio of the SPI.

6.2 KPAs

As the CMMI-DEV framework suggests, the performance of specific areas should be measured with adjusted activities in agile operating organizations. The following SPI evaluation, whether the specific activity is performed, are based on their own recommendations on how they should be adjusted for agile environments.

Tables 5 to 8 summarize improved process areas using the CMMI-DEV guideline. Each process area's specific have defined whether they achieved a generic goal to move to next capability level or not. Level 2 generic goals are redundant for process areas, which did not target this level. All activities included were described in the previous chapter.

Table 5 Evaluated specific and general goals of CMMI – DEV model in Requirements Development process area

		LEVEL 1	LEVEL 2								
	Specific goals	**Perform specific goals**	**Establish org. policy**	**Plan the Process**	**Provide Resources**	**Assign Responsibility**	**Train People**	**Control Work Products**	**Identify and Involve Stakeholders**	**Monitor and Control the Process**	**Objectively Evaluate Adherence**	**Review Status with Mngmnt**
Requirements Development	**SG 1 Develop customer requirements**	Yes	Part of org. guidelines and wiki	Continuous process with scheduled meetings	Several roles dedicated – Consulting Manager, part of PM, Product Designer	Every role has specified responsible area in internal wiki	New internal stakeholders and clients receive guidelines and process description	Requirements have org. guidelines	Clients and users – several platforms to engage them	Ratio of requirements developed vs. not is reviewed biweekly	External consultant company periodically reviewing	Weekly review status
	SG 2 Develop Product Requirements	Yes	Part of org guidelines and wiki	Always before sprint begins	Several roles dedicated – Software Architect, part of PM, Product Designer	Every role has specified responsible area in internal wiki	New internal stakeholders receive guidelines and process description	User stories and technical architecture have specific org. guidelines	CEO and HW department of org. consultations	User stories and technical activities have estimates and are controlled through product plan	External consultant company periodically reviewing	Weekly review status
	SG 3 Analyze and Validate Requirements	Yes	Part of org guidelines and wiki	Always before sprint begins after SG 2	Product Designer	Every role has specified responsible area in internal wiki	New internal stakeholders receive guidelines and process description	Prototypes have a defined designer kit and UX guidelines	CEO. HW department, Consulting manager part of validation if needed	Prototypes and validation also have estimates and are controlled through product plan	External consultant company periodically reviewing	Weekly review status

Table 6 Evaluated specific and general goals of CMMI – DEV model in Project Planning process area

		LEVEL 1	LEVEL 2								
	Specific goals	**Perform specific goals**	**Establish org. policy**	**Plan the Process**	**Provide Resources**	**Assign Responsibility**	**Train People**	**Control Work Products**	**Identify and Involve Stakeholders**	**Monitor and Control the Process**	**Objectively Evaluate Adherence**	**Review Status with Mngmnt**
Product Planning	**SG 1 Establish Estimates**	Yes	Part of org. guidelines and wiki	Always before sprint begins after Requirements Dev. activities	Part of PM and developers' allocations	Every role has specified responsible area in internal wiki	New internal developers receive guidelines and process description	Estimations are recorded directly on grooming to respective task	Development team reviews each estimation on poker planning	Grooming is moderated by the PM	External consultant company periodically reviewing	Weekly review status
	SG 2 Develop Project Plan	Yes	Part of the required process on org wiki	Always before sprint begins after establishment of estimations	PM	Every role has specified responsible area in internal wiki	New internal developers receive guidelines and process description	Project plan uses Gantt chart structure	CEO, Product Designer, technical leads are consulted on product and sprint plan creation	Project plans are revised with reality – estimated vs. real developed hours	-	Weekly review status
	SG 3 Obtain Commitment to the Plan	Yes	Part of roles responsibilities on org wiki	Always before spring begins on grooming meeting	Part of PM and technical leads' allocations	Every role has specified responsible area in internal wiki	New internal developers receive guidelines and process description	Commitments assigned based on developers' skillset	Developers, technical leads and PM	Tasks assignation is monitored and controlled daily on daily standups	-	Weekly review status

Table 7 Evaluated specific and general goals of CMMI – DEV model in Project Monitoring and Control process area

	LEVEL 1	
	Specific goals	**Perform specific goals**
Project Monitoring and Control	**SG 1 Monitor the Project Against the Plan**	Yes
	SG 2 Manage Corrective Actions to Closure	Yes

Table 8 Evaluated specific and general goals of CMMI – DEV model in Verification process area

	LEVEL 1	
	Specific goals	**Perform specific goals**
Verification	**SG 1 Prepare for Verification**	Yes
	SG 2 Perform Peer Reviews	Yes
	SG 3 Verify Selected Work Products	Yes

All process areas were able to reach targeted level 1 capability level - Performed. Requirements development was institutionalized and achieved generic goals of level 2 capability level - Managed. However, Project Planning does not fulfill two out of 10 generic goals of level 2 capability level and is still in the process of reaching the Managed level.

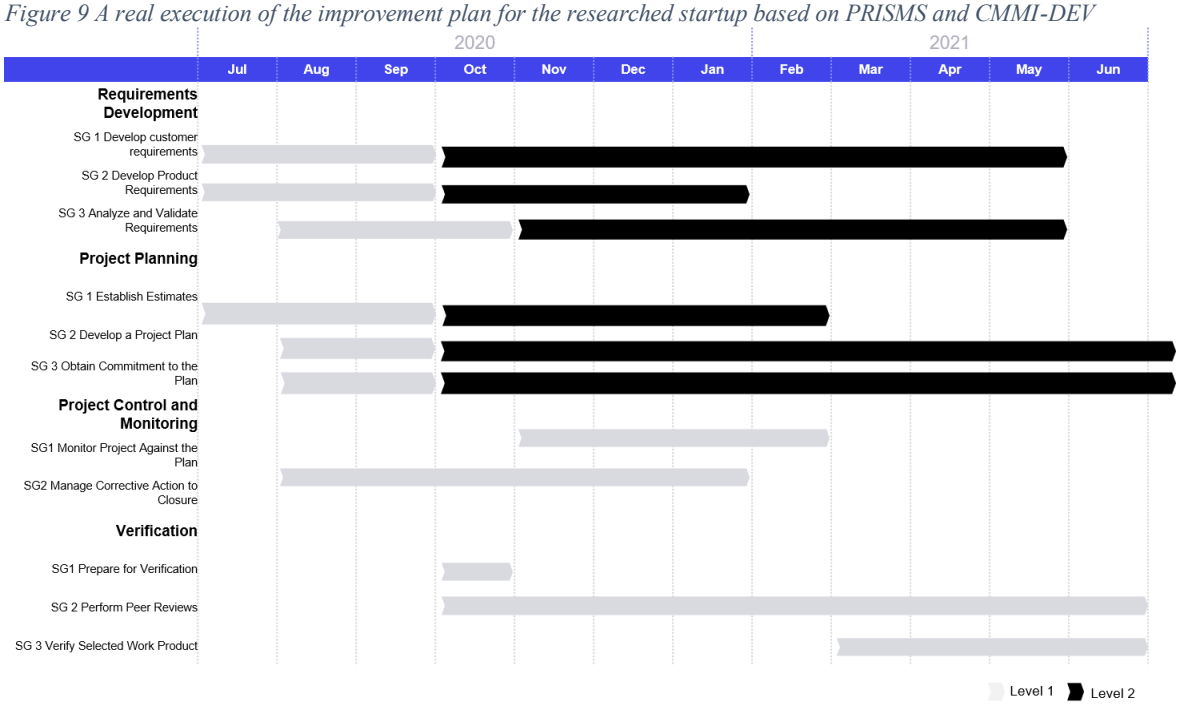
Overall, in comparison to the plan, the performing of the activities was much faster than expected. It was the startup's dynamic and flexible environment that allowed new models to be

implemented almost immediately. Some of the activities from process areas were connected and therefore, they were performed earlier than originally planned.

What showed an unexpected turn was the institutionalization of these processes based on CMMI-DEV framework guidelines. Even though the process was performing, the generic goals of Managed Level needed more time to be applied. The success of performing activities also hindered this process. As the startup and the business felt the improvements of the performing activities, the motivation to institutionalize them and move them to the next capability level decreased. It also required extra time resources which were due to financial situation of the startup dispersed over time. That is why level 2 progress took longer than originally estimated in the improvement plan.

Figure 9 showcases the real improvement plan based on its implementation and when the respective process areas reached the targeted levels.

Figure 9 A real execution of the improvement plan for the researched startup based on PRISMS and CMMI-DEV



7 INTERPREATATION

After presenting the empirical data of the research, this chapter will address the validity of the formulated hypotheses. PRISMS model recommends involving developers in the key process areas definition as their insight of the process is on higher level of details. *Hypothesis 1* concerning startup developers involvement applied this recommendation. To answer this hypothesis *Research Question 1* asks what the differences in software process evaluation between developers and business are. Looking back at the Consultation and Process Assessment process it could be seen there were several differences between pain points identified by the developers and by the business. Only two pain points were identified by both parties (no planning, lack of specifications), while the majority of the technical points were identified only by the developers. Improvements that streamlined the development process such as software architecture or unit tests would then be ignored. Overall, it can be concluded that while business focused on their business objectives and goals, it was actually

developers who focused on the procedural changes. The reason behind is that the flat structure of startup engaged developers in all parts of processes. As developers operate in the process on a daily basis and therefore know exactly what needs to change or what are the things that can be improved.

Hypothesis 2 can be argued. The limitations were clear during the startup. Answering the *Research Question 2* on what the tools to minimize financial investment are, the proposed thesis used prioritization and compliance with the business goals. Firstly, it needed to tailor PRISMS model and apply CMMI-DEV framework instead of CMM. Continuous path of CMMI-DEV allowed distribution of costs investment of SPI to various months. Secondly, it could not implement all process areas at once and had to prioritize which area to focus on and on which level. The lack of human resources to lead the process resulted in delays in Improvement plan, especially in process areas moving to capability Level 2. As a result, the improved process and majority of activities displayed positive effects on the software team and achievement of majority of the defined business objectives. An inability for one process area to reach Level 2 can however not fully confirm this partial hypothesis. Financial constrains allowed to perform identified activities. However, their full institutionalization and advancement to Level 2 was deprioritized due to limited financial resources of the startup.

Concerning *Hypothesis 3*, the contextual factors of the startup were present during various steps of the process. The business model of the startup, where it was necessary to generate partial sales profit from launched modules, structured the process into sprints which ensured regular increments. This incremental methodology had to accommodate agile tools which fulfill this methodology in the most efficient way. *Research Question 3* of how the contextual factors of an organization influence software processes in a company was answered by constant mix of methods used from different frameworks and not just accommodating set of recommended tools of one method, e.g. Scrum.

The final hypothesis of the thesis can be concluded as partially confirmed as Hypothesis 2 could not be fully proved.

H: A SPI model can be used to streamline processes in a startup company if used with appropriate and tailored agile and project management methods.

Conclusion

The proposed thesis focused on the software process streamlining in a selected startup company. More specifically, the thesis aimed to prove that various models of software process improvement, SDLC and agile and project management can coexist to accommodate specific needs and contextual facts of a startup. The thesis applied the concepts and theory on software process, software teams, agile methodologies, software process improvement and startup specifications to implement a customized, but efficient model of software process improvement.

In regard to that, thesis analyzed established models of software process improvements and their benefits and shortcomings. Through variety of other models and versions, there was identified two frameworks that could be applied to the startup's needs – combination of PRISMS SPI framework which customizes SPI methods to small business and CMMI – DEV framework which is an adjusted CMMI framework. After analysis of the startup's contextual framework, agile methodology was used as a tool to implement theoretical frameworks of the model. This led to formulation of three hypotheses. The first dealt with the importance of developers in assessing the process areas to improve. The second assumed that the limited time and financial resources of the startup could be compensated with tailored models and tools used in the software process improvement. Finally, the third one suggested, that contextual and business model variables are integral to decision making on which tools of implementation should be used.

The main hypothesis of this thesis expected a connection between these three hypotheses and summarized it in the assumption that the established SPI frameworks can be successful with tailored models and agile tools in the startup environment. In order to prove these hypotheses, the research continued with answering three research questions. Data researched to confirm them provided a summary of each steps taken in the SPI model – from process areas identification, through prioritization, plan definition and its final implementation. Results of the software improvement were then briefly described and the whole process was evaluated based on CMMI-DEV framework. The process improvement showed beneficial results in the researched startup. Three out of fours defined process areas were able to reach their targeted level. This process has been accompanied by list of real artefacts and work products in the appendices to demonstrate the real process and activites during the research.

Certain limitations of the research have to be considered. Firstly, all data were empirically obtained and could have been subjected to subjectivity. This is something most of qualitative studies suffer from. To compensate for that methodology frameworks are used to conduct a research. The proposed thesis therefore used an established model in the field of SPI - CMMI. The framework also provided guidelines and measures on evaluation the SPI process which addressed lack of quantitative variables in the research. Second limitation is time. As with any other subjects of studies, in order to fully assess results of the SPI, more time would be recommended before conducting conclusions. Last one, case studies are usually limited in generalizations of conclusions.

Nevertheless, this thesis should be used as an example of how to deal with complex and difficult environments of software development. The thesis should also provide inspiration on navigating the steps to consider when choosing SPI model and which development methodology to use in different setups for other small companies or startups. Further studies can be conducted from this thesis as

well. Either by applying the model on other companies with similar conditions or by continuation of the research in the startup with more process areas included or even higher target levels.

8 List of used literature

Al-Tarawneh, Mejhem Yousef, Mohd Syazwan Abdullah, and Jasem Alostad. 2010. "Software Development Process Improvement Framework (SDPIF) for Small Software Development Firms (SSDFs)." *IJCSI International Journal of Computer Science Issues* 10 (1): 475-486, doi: 10.1.1.698.4176.

Allen, P., M. Ramachandran and H. Abushama. 2003. "PRISMS: an approach to software process improvement for small to medium enterprises." *Third International Conference on Quality Software, 2003. Proceedings*: 211-214. https://www.semanticscholar.org/paper/PRISMS%3A-an-approach-to-software-process-improvement-Allen-Ramachandran/54e62c1a6c2c4c77d3a1aebd59648b13ceecfe1a

Alshamrani, Adel, and Abdullah Bahatab. 2015. "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model." *IJCSI International Journal of Computer Science* 12 (1): 106-111, ISSN: 1694-0784.

Anderson, David J., and Andy Carmichael. 2016. *Essential Kanban Condensed*. Lean Kanban University Press.

Baldridge, Rebecca, and Benjamin Carry. 2021. "What is a Startup?" Forbes. https://www.forbes.com/advisor/investing/what-is-a-startup/#544a2a9a4c63.

Batista, J., and A. Dias de Figueiredo. 2000. "SPI in a Very SmallTeam: a Case with CMM." *SOFTWARE PROCESS IMPROVEMENT AND PRACTICE* 5. https://www.academia.edu/4324846/SPI_in_a_very_small_team_a_case_with_CMM.

Blank, Steve, and Bob Dorf. 2012. *The Startup Owner's Manual: The Step-By-Step Guide for Building a Great Company*. California: K & S Ranch, Inc.

Brennecke, Andreas, and Reinhard Keil-Slawik. 1996. "History of Software Engineering." Position Papers for Dagstuhl Seminar 9635. https://www.dagstuhl.de/Reports/96/9635.pdf

Card, David. 2004. "Research directions in software process improvement." *IEEE Xplore: Computer Software and Applications Conference, 2004*. https://doi.org/http://dx.doi.org/10.1109/CMPSAC.2004.1342834.

CMMI Product Team. *CMMI for Development, Version 1.3* (CMU/SEI-2010-TR-033). 2010. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661

CMMI Product Development Team. *CMMI for Systems Engineering/Software Engineering, Version 1.02, Continuous Representation (CMMI-SE/SW, V1.02, Continuous)* (CMU/SEI-2000-TR-019). 2000. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5267

Coallier, Francois, and Motoei Azuma. 1997. “Introduction to Software Engineering Standards.” In SPICE: The Theory and Practice of Software Process Improvement and Capability Determination, Khaled El Emam, Jean-Normand Druoin, and Walcélio Melo, 1-18. Wiley-IEEE Computer Society Pr.

Di Tullio, Dany and Bouchaïb Bahli. 2013. "The Impact of Software Process Maturity on Software Project Performance: The Contingent Role of Software Development Risk." *Systèmes d'Information Et Management* 18 (3) (09): 85-116,147. https://www.proquest.com/scholarly-journals/impact-software-process-maturity-on-project/docview/1491961460/se-2?accountid=17203.

Dora, Sumit Kojar, and Pushkar Dubey. 2013. “SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) ANALYTICAL COMPARISON AND SURVEY ON TRADITIONAL AND AGILE METHODOLOGY.” *Abhinav institute of technology and management* 2 (8).

Forourharfar, Amir. 2014. “Entrepreneurial Timing Theory : Time Entrepreneurship and Time Strategy.” *Asian Journal of Research in Business Economics and Management* 4 (11): 1-27. https://www.researchgate.net/publication/268031860_Entrepreneurial_Timing_Theory_Time_Entrepreneurship_and_Time_Strategy.

Ganga, Deianira, and Sam Scott. 2006. “Cultural "Insiders" and the Issue of Positionality in Qualitative Migration Research: Moving "Across" and Moving "Along" Researcher-Participant Divides.” *FORUM:QUALITATIVE SOCIAL RESEARCH SOZIALFORSCHUNG* 7 (3). https://www.researchgate.net/publication/49913614_Cultural_Insiders_and_the_Issue_of_Positionality_in_Qualitative_Migration_Research_Moving_Across_and_Moving_Along_Researcher-Participant_Divides.

Ghemawat, Pankaj. 1985. “Building Strategy on the Experience Curve.” Harvard Business Review. https://hbr.org/1985/03/building-strategy-on-the-experience-curve.

Grenning, James. 2002. “Planning Poker: How to avoid analysis paralysis while release planning.” http://renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf.

Harroch, Richard D., and Mike Sullivan. 2019. “Startup Financing: 5 Key Funding Options For Your Company.” Forbes. https://www.forbes.com/sites/allbusiness/2019/12/22/startup-financing-key-options/.

Hayes, Adam. 2021. “Venture capital.” Investopedia. https://www.investopedia.com/terms/v/venturecapital.asp.

Ho-Won Jung, & Dennis Goldenson. 2002. *The Internal Consistency of Key Process Areas in the Capability Maturity Model (CMM) for Software (SW-CMM)* (CMU/SEI-2002-TR-037). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=6279

Humphrey, Watts. 1989. *Managing the Software Process*. I. Addison-Wesley Professional.

Juhola, Tomi, Sami Hyrynsalmi, Ville Leppänen, and Tuomas Mäkilä. *Agile Software Development and Innovation: A Systematic Literature Review*. Manchester: The International Society for Professional Innovation Management (ISPIM), 2013. doi: 10.13140/2.1.1046.3049

Kazim, Ali. 2017. "A Study of Software Development Life Cycle Process Models." *International Journal of Advanced Research in Computer Science* 8 (1) (01). https://www.proquest.com/scholarly-journals/study-software-development-life-cycle-process/docview/1901446145/se-2?accountid=17203.

Kenton, Will. 2020. “Bootstrapping.” Investopedia. https://www.investopedia.com/terms/b/bootstrap.asp.

Keshta, Ismail. 2019. “Approaches to Software Process Improvement: A State-of-the-Art Review.” *Journal of Software*: 519-529. https://doi.org/10.17706/jsw.14.11.519-529.

Krasner, Herb. 2001. “Accumulating the Body of Evidence forThe Payoff of Software Process Improvement.” *Software Process Improvement*: 519-539. https://www.researchgate.net/publication/245582411_Accumulating_the_Body_of_Evidence_for_The_Payoff_of_Software_Process_Improvement.

Konrad, Michael D., Mark C. Paulk, and Allan W. Graydon. 1995. “An Overview of SPICE’s Model for Process Management.” *Proceedings of the Fifth International Conference on Software Quality*. : https://www.researchgate.net/publication/243777006.

Kuhrmann, Marco, Philipp Diebold, and Jürgen Münch. 2016. "Software Process Improvement: A Systematic Mapping Study on the State of the Art." *PeerJ Computer Science*. doi:http://dx.doi.org/10.7717/peerj-cs.62.

Lee, Seiyoung, and Hwan-Seung Yong. 2013. “Agile Software Development Framework in a Small Project Environment.” *J Inf Process Syst* 9 (1). https://doi.org/http://dx.doi.org/10.3745/JIPS.2013.9.1.069.

Liaupsin, C. J. (2003). The comprehensive evaluation of professional development software: A critique of methodology. *Journal of Special Education Technology, 18*(1), 29-37. doi: 10.1177/016264340301800103

Masters, Steve, and Carol Bothwell. 1995. “CMM Appraisal Framework, Version 1.0 (CMU/SEI-95-TR-001).” *Software Engineering Institute, Carnegie Mellon University*. https://doi.org/10.1184/R1/6585155.v1.

Malakar, Sudipta. 2021. *Agile Methodologies In-Depth: Delivering Proven Agile, SCRUM and Kanban Practices for High-Quality Business Demands*. BPB Publications, India.

“Manifesto for Agile Software Development.” 2021. https://agilemanifesto.org/.

Mansour, Shefir. 2021. “Product Manager: The role and best practices for beginners.” Scrum. https://www.atlassian.com/agile/product-management/product-manager.

Marum Simão Filho, Plácido R. Pinheiro, Adriano B. Albuquerque, and J. P. C. Rodrigues Joel. "Task Allocation in Distributed Software Development: A Systematic Literature Review." *Complexity* 2018, (2018): 13. doi: 10.1155/2018/6071718

McAvoy, John, and Tom Butler. 2009. “A Failure to Learn by Software Developers: Inhibiting the Adoption of an Agile Software Development Methodology.” *Journal of Information Technology Case and Application Research* 11 (1): 23-46. https://doi.org/https://doi.org/10.1080/15228053.2009.10856152.

McAvoy, John, and Tom Butler. 2017. “The Role Of Project Management In Ineffective Decision Making Within Agile Software Development Projects”. Online. *European Journal Of Information Systems* 18 (4): 372-383. https://doi.org/10.1057/ejis.2009.22.

McCormick, Mike. 2012. “Waterfall vs. Agile Methodology.” *MPCS, Inc.* http://mccormick-pcs.com/images/Waterfall_vs_Agile_Methodology.pdf.

Misra, Subhas. "Agile Software Development Practices: Evolution, Principles, and Criticisms." *The International Journal of Quality & Reliability Management* 29, no. 9 (2012): 972-980. doi:10.1108/02656711211272863

Morgan, Jacob. 2015. “The 5 Types Of Organizational Structures: Part 2, 'Flatter' Organizations.” Forbes. https://www.forbes.com/sites/jacobmorgan/2015/07/08/the-5-types-of-organizational-structures-part-2-flatter-organizations/.

Murray, Anna P.. 2016. *The Complete Software Project Manager : Mastering Technology from Planning to Launch and Beyond*. Hoboken: John Wiley & Sons, Incorporated.

Naur, Peter, and Brian Randell. “SOFTWARE ENGINEERING: Report on a conference sponsored by the NATO SCIENCE COMMITTEE Garmisch, Germany, 7th to 11th October 1968.” *NATO SCIENCE COMMITTEE* 1969. https://www.scrummanager.net/files/nato1968e.pdf.

Paulk, Mark C. 2009. “A History of the Capability Maturity Model for Software.” *Software Quality Professional* 12 (1). https://www.researchgate.net/publication/229023237_A_History_of_the_Capability_Maturity_ModelR_for_Software.

Richardson, Ita. 2002. "SPI Models: What Characteristics are Required for Small Software Development Companies?" *Software Quality Journal* 10 (2) (09): 101-114. doi:http://dx.doi.org/10.1023/A:1020519822806.

Rico, David F. “Software Process Improvement: Impacting the Bottom Line by using Powerful “Solutions”.” 1997. http://davidfrico.com/spipaperpdf.htm.

Royce, Walker. 2002. “CMM vs. CMMI: From Conventional to Modern Software Management.” *The Rational Edge*. https://www.csc.kth.se/~karlm/CMMI.pdf.

“SDLC (Software Development Life Cycle) Phases, Process, Models.” 2021. Software Testing Help. https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/.

Schwaber, Ken, and Jeff Sutherland. 2020. “Scrum Guide.” https://scrumguides.org/index.html.

“Software Architecture.” 2021. Software Engineering Institute, Carnegie Mellon University. https://www.sei.cmu.edu/our-work/software-architecture/.

Sommerville, Ian. 2011. *Software engineering*. 9 ed. Boston, US: Pearson Education.

Steiger, Jen S., Khalid Ait Hammou, and Md Hasan Galib. 2014. “An Examination of the Influence of Organizational Structure Types and Management Levels on Knowledge Management Practices in Organizations.” *International Journal of Business and Management* 9 (6). https://doi.org/10.5539/ijbm.v9n6p43.

Sutton, Stanley M. 2000. “The Role of Process in a Software Start-up.” *IEEE Softw.* 17: 33-39. doi:10.1109/52.854066

Unterkalmsteiner, Michael, Tony Gorschek, A. K. M. Islam, Chow Kian Cheng, Rahadian Bayu Permadi, and Robert Feldt. "Evaluation and Measurement of Software Process Improvement-A Systematic Literature Review." IEEE Transactions on Software Engineering 38, no. 2 (Mar, 2012): 398-424. doi: 10.1109/TSE.2011.26

"Upgrading from SW-CMM® to CMMI®." 2004. *Software Engineering Institute, Carnegie Mellon University*. https://resources.sei.cmu.edu/asset_files/WhitePaper/2004_019_001_29417.pdf.

Wang, Lingfeng. 2007. "Agility counts in developing small-size software." *IEEE Potentials* 26 (6): 16-23. https://doi.org/10.1109/MPOT.2007.906114.

"What are Microservices?" 2021. Microservices Architecture. https://microservices.io/.

What is a Developer in Scrum?" 2021. Scrum. https://www.scrum.org/resources/what-is-a-scrum-developer.

"What is a Product Owner?" 2021. Scrum. https://www.scrum.org/resources/what-is-a-product-owner.

Williams, Shanon. 2021. "7 types of organizational structures (+ org charts for implementation)." Lucid Chart. https://www.lucidchart.com/blog/types-of-organizational-structures.

List of pictures

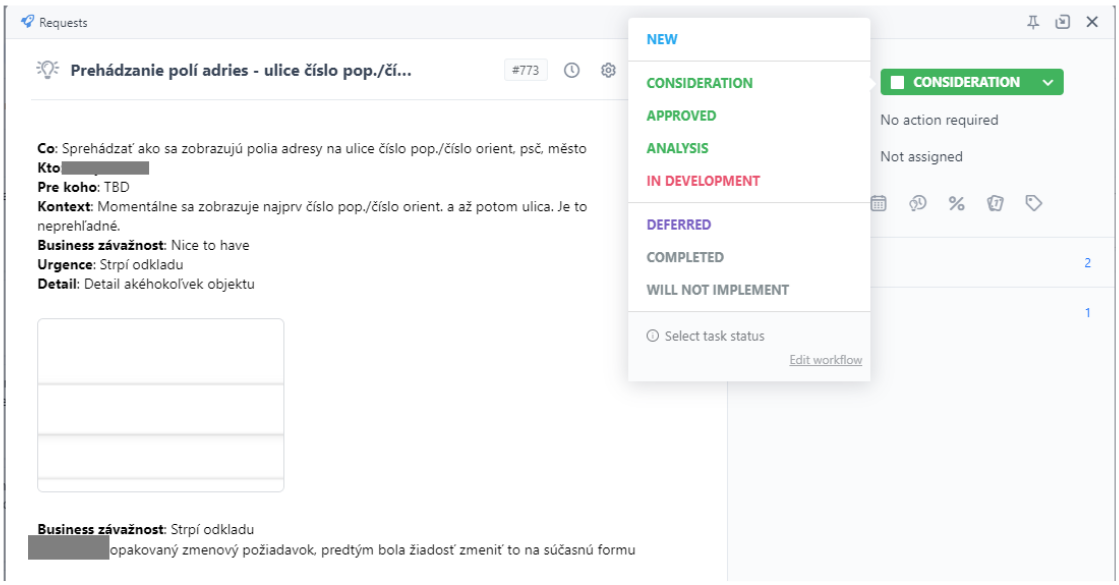
List of tables

Appendices

Appendix A: List of abbreviations used in the thesis

ADS – Agile Software Development
CEO – Chief Executive Officer
CMM – Capability Maturity Model
CMMI – Capability Maturity Model Integration
CMMI-DEV – Capability Maturity Model Integration for Development
DevOps – Development and Operations
HoSD – Head of Software Delivery
HoSW – Head of Software Department
KPA – Key Process Area
PM – Project Manager
QDF – Quality Function Development
SDLC – Software Development Lifecycle
SEI – Software Engineering Institute
SG – Specific Goal
SPI – Software Process Improvement
UX – User Experience
WIP – Work in Progress
XP – Extreme Programming

Appendix B: A real example of requirement registering in the revised process of the startup



Appendix C: A real example guideline defined by the startup for requirements definition in the internal wikipedia

Requests

- **Čo:** požadavky od užívateľov, zmenové návrhy, návrhy na zmenu/prepracovanie funkcionality
- **Prečo:** Aby sme mali miesto, kde zhromažďovať požiadavky a tie potom prioritizovať
- **Kto vytvára:** ALL
- **Kto posúva:** [redacted]
- **Stavy:**
  - New - novo vytvorený request
  - Consideration- request, o ktorom je treba sa ešte pobaviť
    - napr. ak zákazník navrhne nejakú zmenu a my sa rozhodneme, či pôjde do stavu approved až vtedy, až dodiskutujeme, v akom rozsahu tá zmena má byť prevedená
  - Approved - requests, ktoré po prioritizácii boli odsúhlasené a budú sa implementovať
    - z týchto requests sú vytvorené a prepojené vývojové tickety vo workflow Delivery
  - Analysis - requests, ktoré boli odsúhlasené sa analyzujú z businessového a technického hľadiska
  - In development - requests, ktoré sú momentálne vo vývoji
  - Deferred- requests, ktoré sme nateraz odložili ale môžu byť neskôr zapojené opäť do Consideration statusu
  - Deployed - requests, ktoré sú nasadené v prostredí
  - Will not implement - requests, ktorými sa nebudeme zaoberať a nebudeme ich implementovať

Komplexnost

- Requests by mali zahŕňať jednu funkcionalitu, prípadne viac
- Ich komplexnosť nemusí byť limitovaná, keďže sa potom aj tak rozpadajú na vývojové tasky samostatne a po ich prioritizácii

Formát popisu

Requests by mali mať tento formát aj s príkladom:

Co: O akú funkcionalitu sa jedná

príklad: V prípade, že je uvedená iba jedna adresa (TRV - doručovacia), automaticky to mať prepísanú ako trvalú adresu (ATP - trvalá).

Kto: Kto to navrhuje/požaduje

príklad: [redacted]

Pre koho: Pre koho to bude potrebné, užívatelia

príklad: učetní oddělení SBD, inštalatér, správca BK, BeiT

Kontext: Čo je za požadovanou funkcionalitou a prečo sa požaduje jej vývoj/zmena

príklad: U subjektoch, ktoré majú uvedenú len jednu adresu (v importnom súbore TRV - dorurčovacia adresa), sa automaticky považuje táto adresa ako trvalá [redacted] sa momentálne prepisuje len ako doručovacia - tj. mať možnosť túto adresu duplikovať v prípade, že je osamotená ako trvalú. V prípade, že u subjektu evidujeme trvalú (ATP) aj doručovaciu (TRV) tak uvádzať obe.

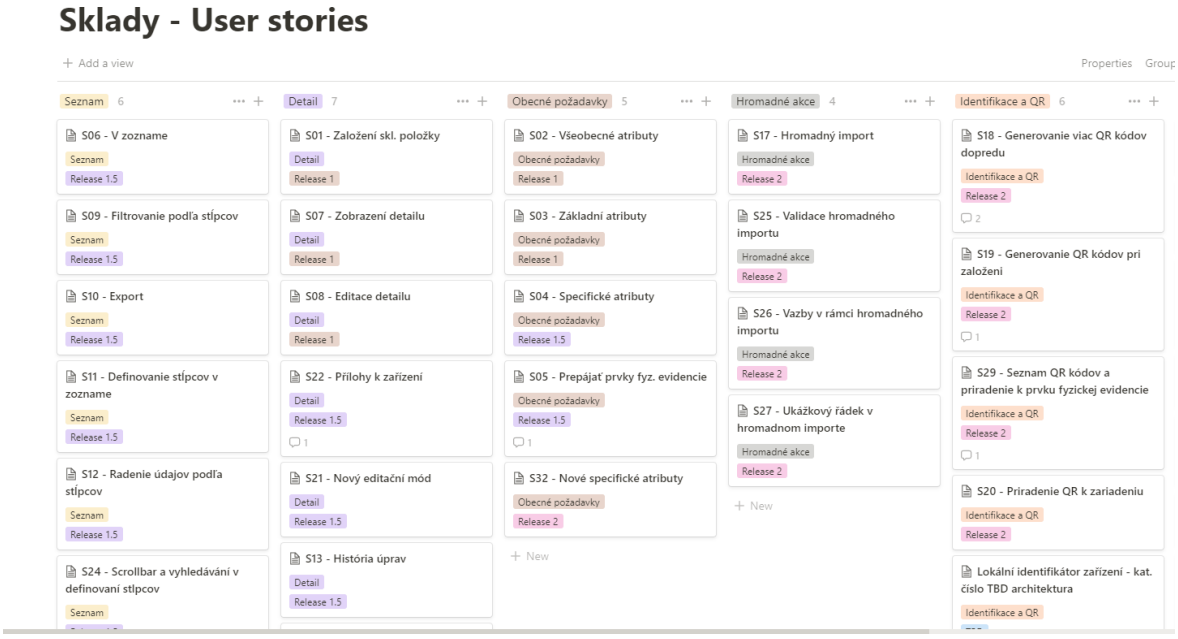
Business závažnost: Vyberáme medzi Nice to have/Must have

Urgence: Vyberáme medzi Strpí odkladu/Nestrpí odkladu

Detail: špecifikácia, kde je funkcionalita požadovaná

príklad: (screenshot, kde je treba pridať tlačítko, zmeniť farbu, apod.)

Appendix D: A real life example of card view of user stories defined for one of the submodels in the startup



Appendix E: A real example guideline defined by the startup for user stories in the internal Wikipedia

User stories guideline

User story vs. epic

Ak je user story veľká, a dá sa rozpadnúť do menších, hovoríme o nej ako o epicu.

Príklad: Ako správca nemovitosti chcem evidovať fyzické zariadenia s ich špecifickými atribútmi, aby som o nich mal prehľad a dokázal ich prepájať s nameranými hodnotami po ich inštalácii.

—> toto je epic, resp. je to veľká user story, ktorá sa dá rozpadnúť na desiatky ďalších.

Epic rozpadnutý na user stories:

- Ako správca nemovitostí chcem definovať atribúty prvku fyzickej evidencie podľa jej typu.
- Ako správca nemovitostí chcem filtrovať prvky fyzickej evidencie podľa ich typu, ID, druhu,...

Je na nás, na akom levely to budeme držať

- buď môžeme urobiť user story na úrovni epicu, tj. rozpracovať veľký dokument s atribútmi, ktoré špecifikujem nižšie a z toho sa budú rozpadať na jednotlivé tasky v cloubhouse
- alebo môžeme rozpadať epicy na menšie user storky, ktoré budú na úrovni zadania

Základný formát

As a ____, - pre koho je user story určená, typ uživateľa

I want ___, - čo je cieľom

so that ____ - benefit alebo hodnota pre uživateľa, čo mu to pridáva

Náš príklad: Ako správca nemovitosti chcem evidovať fyzické zariadenia s ich špecifickými atribútmi, aby som o nich mal prehľad a dokázal ich prepájať s nameranými hodnotami po ich inštalácii.

Business case

Teória začína s bullet points ako high level overview čo má funkcionalita robiť, akú má pridanú hodnotu a ako zapadá do big picture. Prečo je dôležité ju mať.

V našom prípade by to bolo niečo ako:

- [redacted]

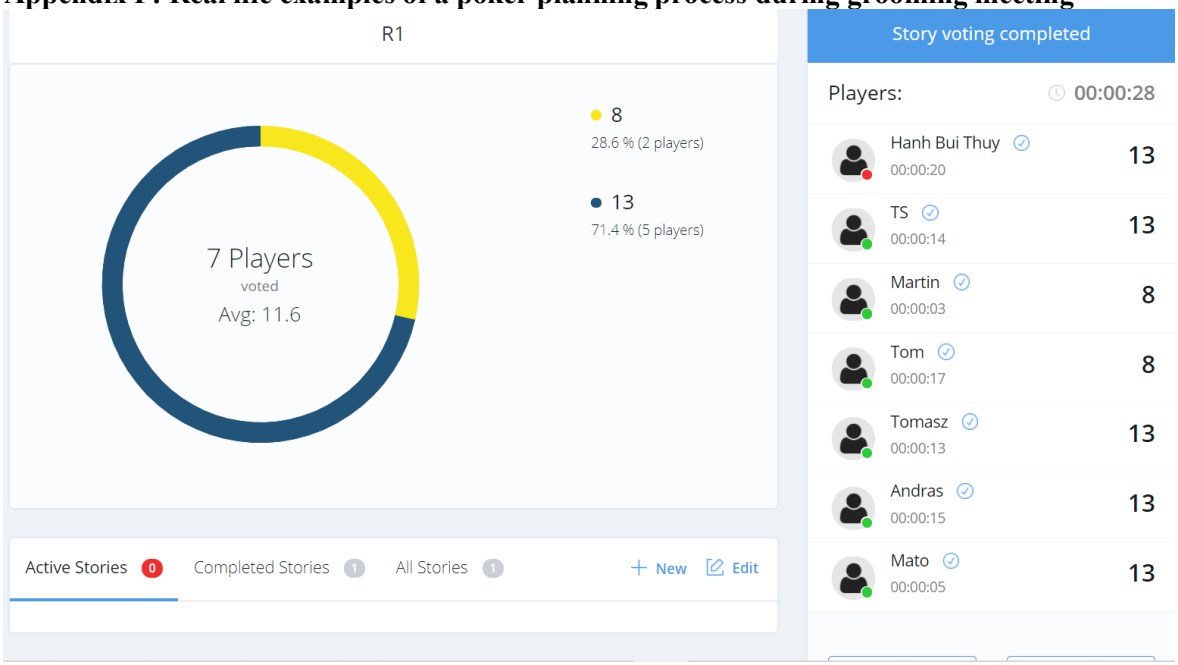
Akceptačné kritéria

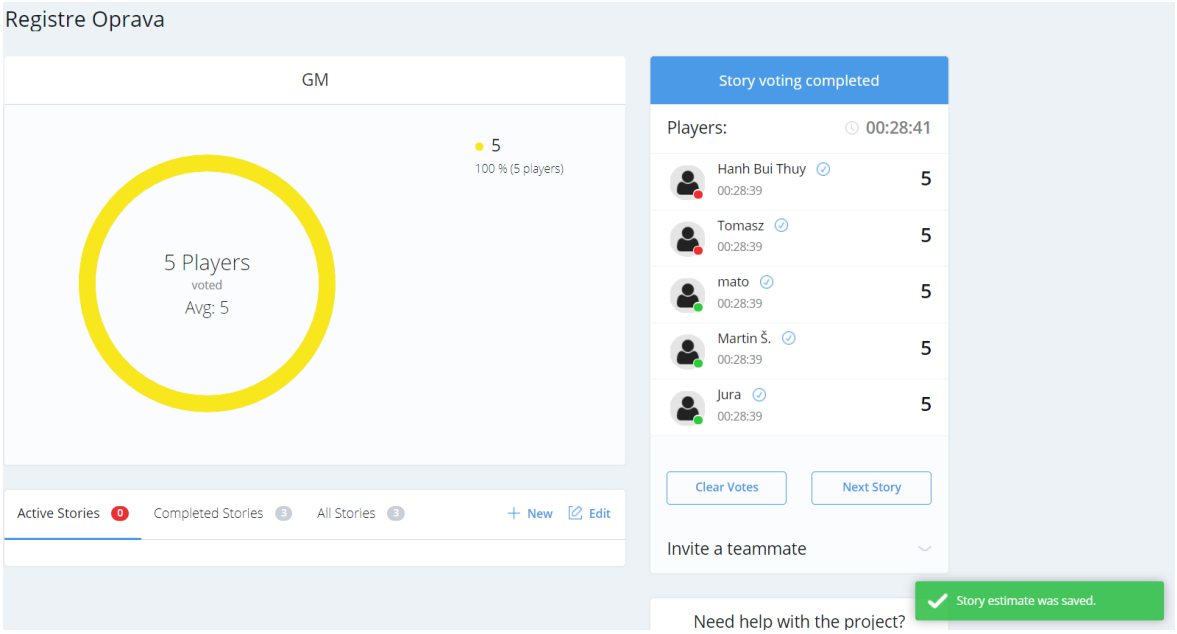
Každá user story musí mať popísané akceptačné kritéria - tj. čo by funkcionalita mala robiť

Čo by mali akceptačné kritéria zahrňovať (nie nutne všetko, ale toto tam patrí)

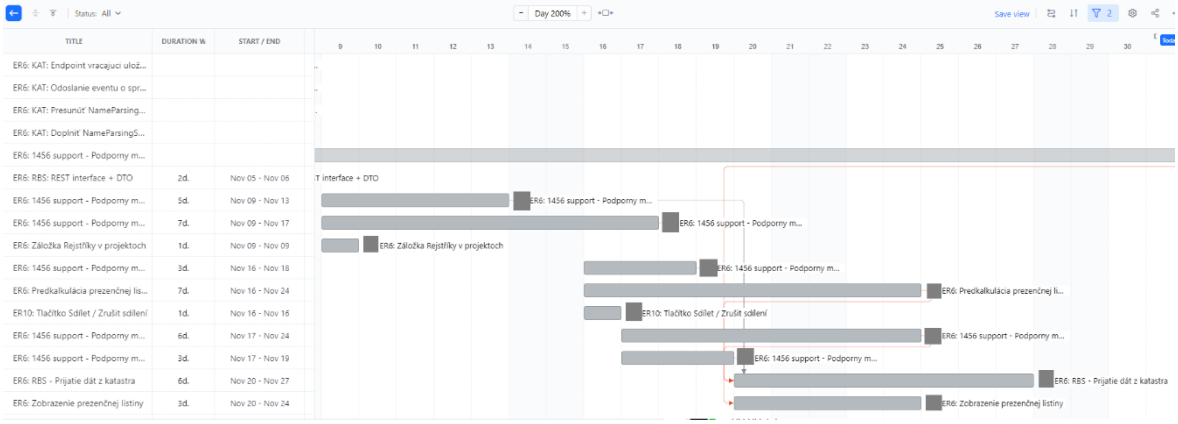
- **negatívne scenáre funkcionality** - [redacted]
- **impakt user storky na ostatné featury** - [redacted]
- **UX detaily**, ak je potreba na niečo brať ohľad - napr. hlavný zoznam by mal byť viditeľný celý na defaultnom rozlíšení obrazovky
- **Funkčné use cases** - funkčné sú tie, ktoré opisujú čo by sa malo robiť napr. schopnosť [redacted]
- **Nefunkčné use cases** - nefunkčné sú tie, ktoré opisujú ako by to systém mal vykonať [redacted]
- **End-to-end user flow** - buď procesné, alebo flow chart cez systémy, cez dáta.

Appendix F: Real life examples of a poker planning process during grooming meeting





Appendix G: A real example of a gantt chart (Sprint plan) of a revised planning process



Appendix H: A real example of a formalized role of technical lead and software developer

Software Developer

Held by [redacted]

Reports to Hanh Bui Thuy

Department Software Development

Add a property

Add a comment...

A truly good Software Engineer thinks about the integrity of his and others code, understand business and users' need for the code and is mindful of the industry the code is developer for.

Main responsibilities

Code Delivery

- Produce clean, efficient code based on specifications
- Troubleshoot, debug and upgrade existing software
- Ensure project deliverables meet business requirements
- Perform code reviews to identify basic technical and logical errors
- Perform unit and automated tests
- Utilize programming principles, tools, and techniques to write application codes
- Multi-task and change from one task to another without loss of efficiency or composure

Product Building

- Be familiar with business analysis and technical architecture in order to deliver code based on specifications
- Recommend and execute improvements
- Understand how delivered code impacts company as whole and creates business value
- Be aware of a business context and industry the software is being developer for

Team Collaboration

- Be mindful of other parts of a function delivered by other team members
- Collaborate with the team on the tasks
- Clearly and regularly communicate with management
- Respect the prioritization and utililization defined by Teachnical Lead/PM/DM

Reporting & Documentation

- Attend stanups and report on your progress
- Report time spent on each task
- Create technical documentation for reference and reporting
- Provide input in determining time and cost estimates for assigned projects

SUMMARY: The main duties and responsibilities of software engineers include directing and participating in programming activities, monitoring, and evaluating system performance, and designing and implementing new programs and features.

Success Metrics

How can a role recognize they are doing a good job? These are the success metrics to help them and their managers identify if the responsibilities are fulfilled as they should.

- ✅ A code is delivered in desirable quality
- ✅ Code fulfills all business and technical requirements set by the User Stories and Architecture
- ✅ Code is delivered within estimated hours defined on Grooming

Technical Lead

Held by [redacted]
Reports to Hanh Bui Thuy
Department Software Development
Add a property

Add a comment...

Outstanding Technical Leads are accountable, trustworthy, and able to build lasting relationships with their teams.

Main responsibilities

Delivery

- Take up the responsibility of the team's delivery
- Ensure the delivery functions in a required quality and scope
- Assure code's quality
- Undergo the process of analysis in order to fulfil the requirements
- Put together separated tasks of team members into functioning code
- Achieve daily, weekly, and monthly goals
- Has clear picture of the function, what is left of backlog and what issues have been reported

Team Management & Work Organization

- Lead Stand ups and ensure their efficiency and time scope
- Delegate tasks
- Liaise with team members and to ensure projects are completed to standard
- Ensures the team is productive, fully utilized and efficient
- Ensures communication and collaboration between FE and BE
- Collaborate with their team to identify and fix technical problems

Technical Support & Consultation

- Review team member's codes
- Has the knowledge about the implemented area and technologies used
- Guide their team through technical issues and challenges
- Discuss confusions or discrepencies in tha architecture or analysis with BA/Architect

Gateway For Non-Technicals

- Serve as a technical translator between developer and management
- Assist and guide the PM in project coordination
- Assist and guide testers in quality assurance process
- Reports on emergencies and potential risks to the PM and DM
- Reports on the risks of delivery's target not being met

SUMMARY: Technical Lead is the one who actually creates a technical vision in order to turn it into reality with the help of the team.

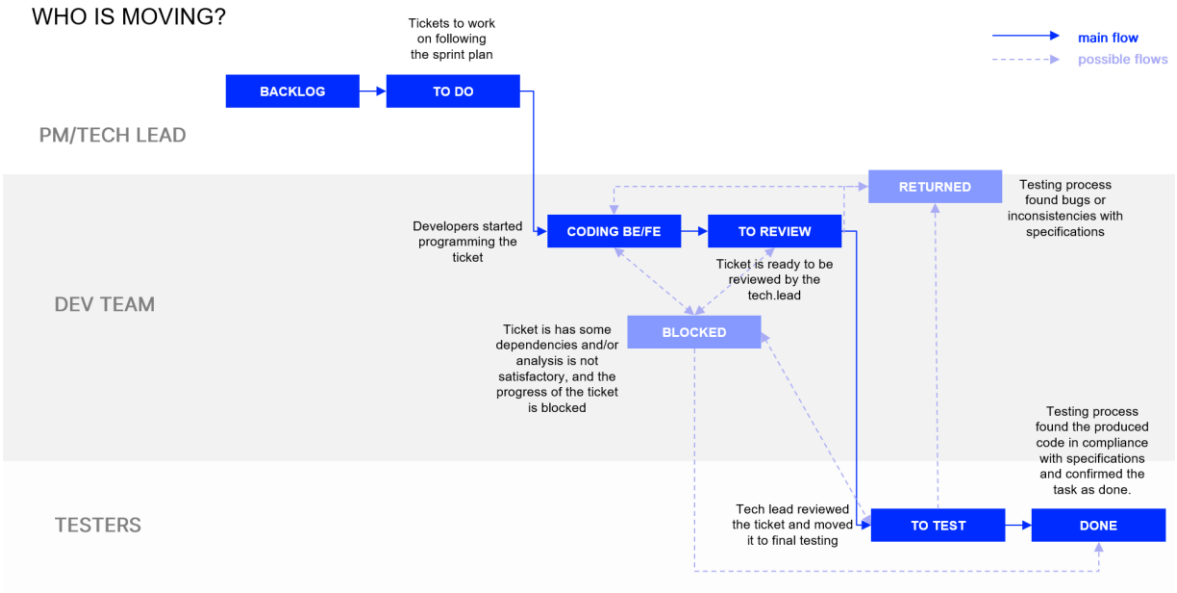
Success Metrics

How can a role recognize they are doing a good job? These are the success metrics to help them and their managers identify if the responsibilities are fulfilled as they should.

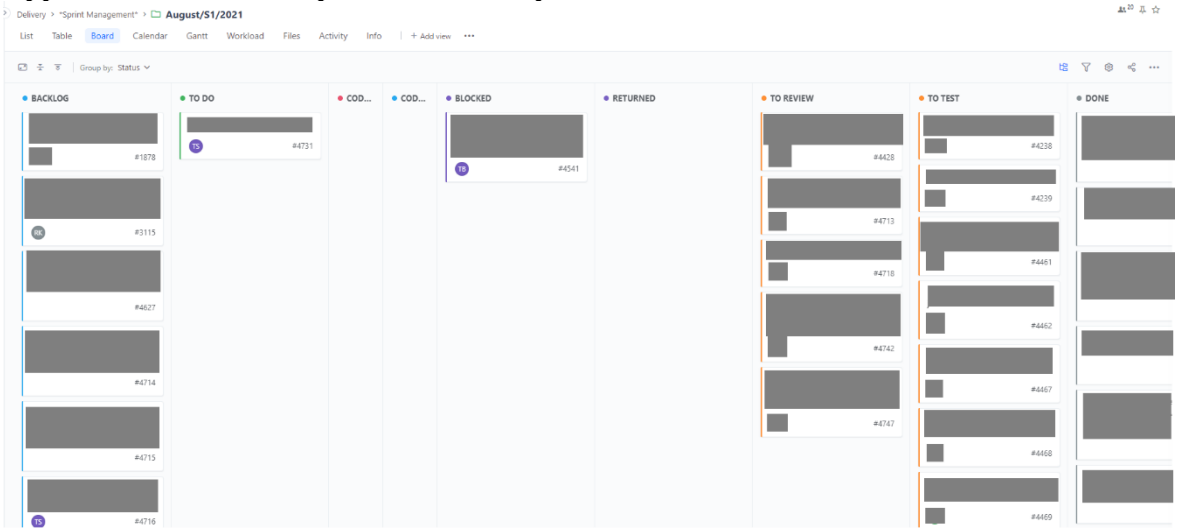
- ✅ A coherent function is delivered with required quality and within defined time scope
- ✅ Every team member knows exactly what task he should work on
- ✅ PM understand and has a clear picture on the release/sprint progress
- ✅ Kanban board of a project is being updated on daily basis

IMPORTANT: A Technical Lead is not responsible for people allocation in a team and is not held responsible if such allocations prevent fulfilling project's target.

Appendix I: A guideline diagram for moving the cards in sprint Kanban boards



Appendix J: A real example of one of the sprint Kanban boards



Appendix K: A real example of guidelines of formalized daily standups

Standups

Why do we have a Daily Standup:

1. Helps us stay in touch with each other daily
2. Checks the progress and who is working on a specific task
3. Provides a clear picture for the team if we're on track to complete the sprint's goal
4. Serves as a platform to ask for help/provide help to each other
5. Makes it possible to reprioritize and assign tasks

When and where do we have a Daily Standup:

- Daily from 9am or 9:15am based on the team assignation
- via Skype sharing a Kanban board with specific iteration

How do we have a Daily Standup:

- tech lead starts sharing a screen and gives a status himself
- the status is given answering three questions:
  - What did I accomplish yesterday?
  - What will I do today?
  - What obstacles are preventing me in progressing?
- following these questions, the status should be given while describing the tickets on the shared screen from right to left- starting with Completed/Ready for Test/Ready for Review, etc.
- tech lead asks every one individually to give status based on the formula above

What to do during a Daily Standup

#1: Parking Topics

If a certain problem, topic or an obstacle require a lengthy discussion, it should be parked and followed up on after the call. This way, you don't consume everyone's time, as some topics are only related to few of us and keep the standup short.

#2: Offering Help

If a teammate faces an obstacle or a problem you have been working on, tell them that you're open to offering input/help if needed even if they don't necessarily ask for help or advice.

#4: Sharing Status With Tickets Seen On The Screen

A screen sharing is an effective tool how to make sure the work being done has a ticket in Clubhouse so it does not happen someone is working on a task and have nowhere to log their time. It also helps give an overview of what tasks are we talking about exactly, rather than just saying abstract definitions non-technical people do not understand.

What not to do during a Daily Standup

#1: Problem-Solving During The Standup

Obstacles in our work are voiced on a Stand Up and teamates provide help or advice, but the problem should not be solved on a Stand Up. Teammates should agree to discuss/call earch other after the Stand Up or agree on a followup meeting to discuss the problem.

#2: Not Attending The Standup

Sometimes it is not possible to attend the meeting everyday. However, due to the reasons specified in Why do we have a Daily Standup? it is very important for every team member to be present. If any teammate cannot attend due to urgent reasons, they should always give a quick answer to the three questions in a Skype group before a Standup starts.

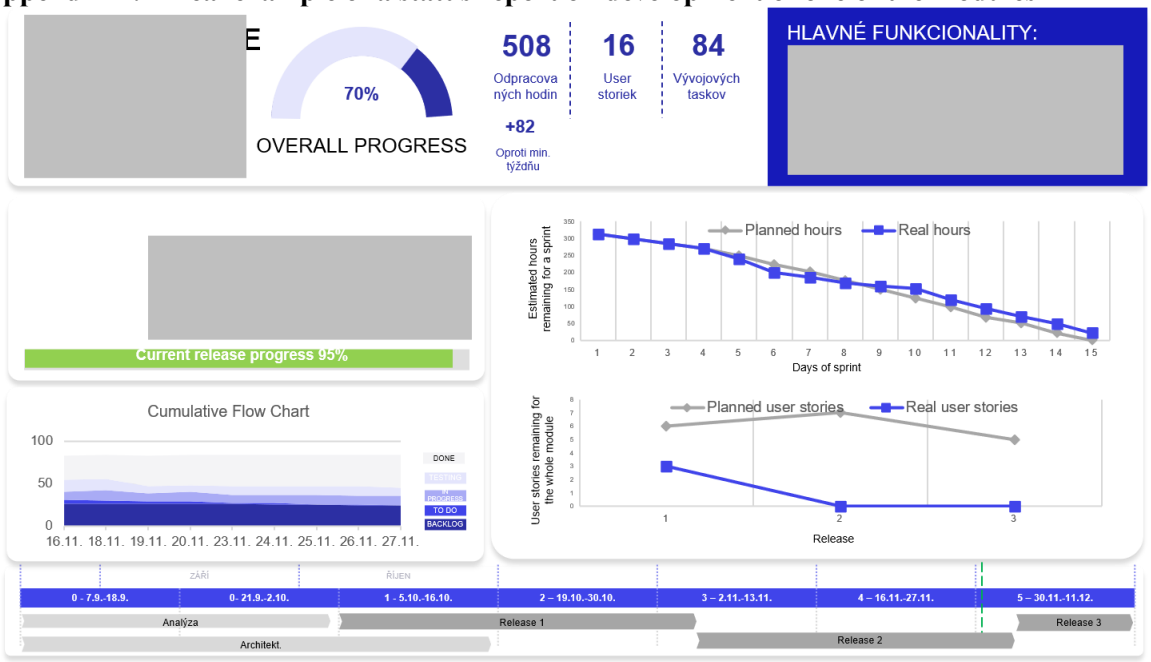
#3: Code Review During The Standup

As in the case of a problem solving, a code review should happen outside of a Standup.

#4: Talking About How I Solved Something

A Standup should be an overview of work done, not how we did it or how we debugged it.

Appendix L: A real example of a status report on development of one of the modules



Records of Loans

Declaration:
I give permission to lend this master's thesis. The user confirms with their signature that they will properly cite this work in the list of used literature.

Name and surname: Hanh Bui Thuy
In Prague 20. 08. 2021 Signature:

Name	Department	Data	Signature