LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
School of Engineering Science
Master's Programme in Software Engineering and Digital Transformation

PETER THE GREAT ST. PETERSBURG POLYTECHNIC UNIVERSITY
Graduate School of Business and Management of Institute of Industrial Management, Economics and Trade
Master's Programme in Business-Engineering Technologies

Denis Surkov

PERSONAL SOFTWARE PROCESS ADAPTATION IN AGILE SOFTWARE DEVELOPMENT FOR IMPROVING DEVELOPER SKILLS

1st Supervisor/Examiner: Assoc. Prof., Uolevi Nikula, LUT

2nd Supervisor/Examiner: Prof., Dr. Sc., Igor V. Ilin, Peter the Great St. Petersburg Polytechnic University

Lappeenranta, Finland – Saint Petersburg, Russia

2019

ABSTRACT

Author:	Denis Surkov
Title:	Personal software process adaptation in Agile software development for improving developer skills
Department:	LUT School of Engineering Science, Software Engineering
	Peter the Great St. Petersburg Polytechnic University, Graduate School of Business and Management of Institute of Industrial Management, Economics and Trade
Master's Programme:	
	Double Degree Programme between LUT Software Engineering and Peter the Great St. Petersburg Polytechnic University
Year:	2019
Master's thesis:	Lappeenranta University of Technology, Finland
	Peter the Great St. Petersburg Polytechnic University, Russia
	63 pages, 5 tables, 7 figures
Examiners:	Assoc. Prof., Uolevi Nikula, LUT Prof., Dr. Sc., Igor V. Ilin, Peter the Great St. Petersburg Polytechnic University
Keywords:	personal software process, software development, software project, Agile, developer skills

This work examines the options for the development of professional skills of software developers, performed according to Agile methodologies. As part of this study, data from fifteen IT projects implemented by two companies in the city of St. Petersburg was collected and analyzed. Comparison and description of each phase of the project throughout the entire development lifecycle were made, which revealed the differences in project management and their weaknesses, which lead to the appearance of defects in the developed software. Based on the analyzed projects, recommendations were made for improving the qualifications of developers and expanding their skills in business analysis. The results that were obtained in the course of this study can be useful to all developers whose responsibilities go beyond the writing of code, as well as to the management staff in the IT field for developing their team.

РЕЗЮМЕ

Автор: Сурков Денис Игоревич

Заглавие: Адаптация индивидуального процесса разработки при применении методологии Agile для улучшения профессиональных навыков разработчика

Факультет: ЛТУ Факультет Инженерных наук

Санкт-Петербургский политехнический университет Петра Великого, Институт промышленного менеджмента, экономики и торговли, Высшая школа управления и бизнеса

Магистратура: Технологии бизнес-инжиниринга

Год: 2019

Диссертация: Лаппеенрантский Технологический Университет, Финляндия

Санкт-Петербургский политехнический университет Петра Великого, Россия

63 страниц, 5 таблиц, 7 рисунков

Экзаменаторы: Профессор Уолеви Никула, Лаппеенрантский Технологический Университет
Профессор, д.э.н., Игорь В. Ильин, Санкт-Петербургский политехнический университет Петра Великого

Ключевые слова: персональный процесс разработки, разработка программного обеспечения, ИТ-проект, Agile, профессиональные навыки разработчика

Данная работа исследует варианты развития профессиональных навыков разработчиков программного обеспечения, выполняемого по методологиям Agile. В рамках данного исследования были собраны и проанализированы данные пятнадцати ИТ-проектов, реализованных двумя компаниями в городе Санкт-Петербург. Было произведено сравнение каждого этапа проекта на протяжении всего жизненного цикла разработки, что позволило выявить разницу в ведении проектов и их слабые стороны, которые приводят к появлению дефектов в разрабатываемом программном обеспечении. Были предложены рекомендации по повышению квалификации разработчиков в сфере анализа бизнеса. Результаты, которые были получены в ходе данного исследования, могут быть полезны всем разработчикам, чьи обязанности выходят за рамки написания кода, а также руководящему персоналу в ИТ-сфере для развития своей команды.

Acknowledgments

First of all, I would like to thank Assoc. Prof. Uolevi Nikula of the School of Engineering Science at the Lappeenranta University of Technology. His valuable comments and feedback during my work on my master's thesis helped me greatly structure my thoughts and workflow as a whole. Under his leadership, for the first time, I applied the approaches of Agile in conditions that were especially unusual for me - writing a scientific work. Nevertheless, the most crucial thing that Assoc. Prof. Uolevi Nikula taught me was all the time to look for space for improvements, which gave enhancements not only to my educational process but also to the worker. For all this, I am very grateful to him.

In addition, I would like to thank my research team, which helped me with the work in this study. Their invaluable contributions and helpful advice helped me cope with the implementation of this study.

Finally, I want to express my gratitude to those in the companies in which I was lucky to work and in one of which I continue to work to this day, for the opportunity to gain practical experience, analyze this experience and grow as a specialist.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

CMM	– Capability Maturity Model
PSP	– Personal Software Process
SPI	– Software Process Improvement
TSP	– Team Software Process

1 INTRODUCTION

In the last 25 years, huge amounts of different methodologies and techniques have been presented in software development. Only some of them have passed the test of time and development processes and are used nowadays (Abrahamsson, 2002).

The introduction of the Agile Manifesto in 2001 has driven software engineering area with groundbreaking changes. The revolution that the manifesto has caused is rather extraordinary. It is hard to imagine another period of creating so many software methodologies, techniques, and practices. While many engineers have quickly recognized this unprecedented growth, many challenges have still to be tacked to carry coherence to the current reasoning on the flexibility of the methods (Dingsøyr et al., 2012).

Companies that have made a great attempt on improving their processes based on the Capability Maturity Model (CMM) have ascertained that Agile methods can ensure improvements as well. Over the past years, the CMM has been widely utilized for estimating organizational maturity and process capability throughout the world. Companies are confident in CMM use because of its extensive descriptions of how the different well practices correspond to each other. Lately, a new method for software development has caused a deep interest in software development organizations. Agile software development methods, procedures, and techniques ensure to enhance customer satisfaction, to build high-quality products and to decrease development times (Pikkarainen et al., 2006).

Even though CMM provides a well-built basis for improvement, its attention is undoubtedly focused on “what” organizations should do, and not on “how” they should do it. One of the challenges of using CMM is that in small companies it is usually not possible to hire process experts, so every engineer should be involved in process improvement at least part-time (Paulk, 1998). To demonstrate how to apply the principles of the CMM process and make it effective, it is imperative to explain to software engineers what they should do and to answer their questions and eliminate misunderstandings during explanations and practical actions. This meant not only the need for much greater detail of the process, but also required a clearer understanding of the proper practices of design engineers that requires changes and improvements in the behavior of each engineer and an introduction to the personal software process (Humphrey, 1999).

Personal Software Process (PSP) is a software process for software developers that was introduced by Humphrey (1997). PSP and Agile methods, such as Scrum, are usually considered to be opposite to each other, because PSP drives by plan and emphasizes discipline, while Agile promotes flexibility. PSP consists of exact stages, applications, norms, and scenarios. It introduces an analysis and measurement approach to track and guide the engineer's personal work. Defects, size and time are the three main indicators in the PSP, which also explains the process improvement model, which presents comprehensive practices at different levels of maturity and leads to the improvement of developer skills. A typical PSP process includes steps such as planning, designing, coding, compiling, testing, and post-opening. A script accompanies each stage. Also, the PSP is a continuously developing software process that consists of seven primary levels. Each level provides new instructions that is based on its previous levels. These seven levels consist of a separate model for improving the software development process and guide a mature software developer.

Present software development involves adaptability, which is better encouraged by Agile methods, and predictability that is encouraged by plan-driven processes. According to Rong et al. (2010), after a thorough analysis of the PSP and Agile practices, it can be argued that they do not conflict with each other. PSP is a process with separate but related levels that focuses on upgrading the skills of a software engineer, while Agile is a process model focused on team collaboration and its adaptation to project conditions.

In this paper, I examine 15 software projects, which were gathered from two companies, that were done with Agile methodologies and propose the way of PSP adaptation in Agile projects for a software engineer, which merges the strengths of PSP and Agile methods. Compared to traditional Agile, the suggested adaptation provides specific instructions that lead and support the engineer to reach better task estimation, quality assurance and workflow management. In this case, this work tackles problems as an optimal way of development time using, maintenance issues and, as a result, eliminating weak points of developer skills during the software development lifecycle. To explore the issues mentioned before, the following research questions (RQ) are addressed:

1. How to improve software quality and reduce software development defects?
2. How to improve personal software development workflow?

2.1. What are the most important developer skills needed to be improved during Agile software development?

2.2. How can developer maturity be defined?

Selecting these research questions will also help point out those missing aspects that every software developer should pay attention to develop their professional qualities for useful work in a team and organization as a whole.

This work has the following structure. Second chapter discusses previous works on the topic of the PSP and TSP, as well as their relationship with the models of skills acquisition, software development using Agile methodologies, and also describes the relationship with the methods of Software Process Improvement. Third chapter describes the method by which this study was carried out, the main phases of the research process, including the structure of this process. Chapter four demonstrates the results of the research and their analysis, answering the research questions. Chapter five gives a discussion of this study using works from previous years. Final chapter, chapter six, presents the results of the research and the goals achieved.

2 LITERATURE REVIEW

2.1 Personal Software Process

Personal Software Process is a structured, sequential work model that is designed to enable software engineers to plan their tasks, track them and deliver a high-quality product to end users. PSP provides engineers with the data, which helps to follow their plans, meet all user requirements and, as a result, ensure the quality of developed product (Ferguson et al., 1997). This model can be utilized during several stages in the software development process (requirements engineering, system testing, writing of documentation, and maintenance) and for software with different scope from the small mobile application to large enterprise systems (Humphrey, 1994).

PSP was introduced to reduce defects in developed products what is achieved by improving workflow planning and tasks estimating skills. Engineers, who studied PSP techniques, become with encouraging improvements in productivity around 20 percent and defects reducing in five times (Johnson et al., 2003). PSP includes methods that help to improve quality, prevent bugs and weak spots in the development process, expeditiously find them and fix. Such defect tracking allows engineers to concentrate on their personal mistakes, pay higher attention to work and think in advance. PSP makes it possible by defining templates for data recording and techniques for managing code measurements, size metrics and allocation of defect types. Within PSP methodology an engineer plans and documents his or her work. Then, developing a product, the engineer must track and record in a template each defect that occurs. When the project ends, the engineer analyzes finished work and makes a summary report for the project. As a result, each engineer who follows PSP, can obtain a clear understanding of defect removed expenses and then use effective practices for defect-controlling and correcting (Johnson et al., 1998). PSP consists of seven levels that are shown in Figure 1.

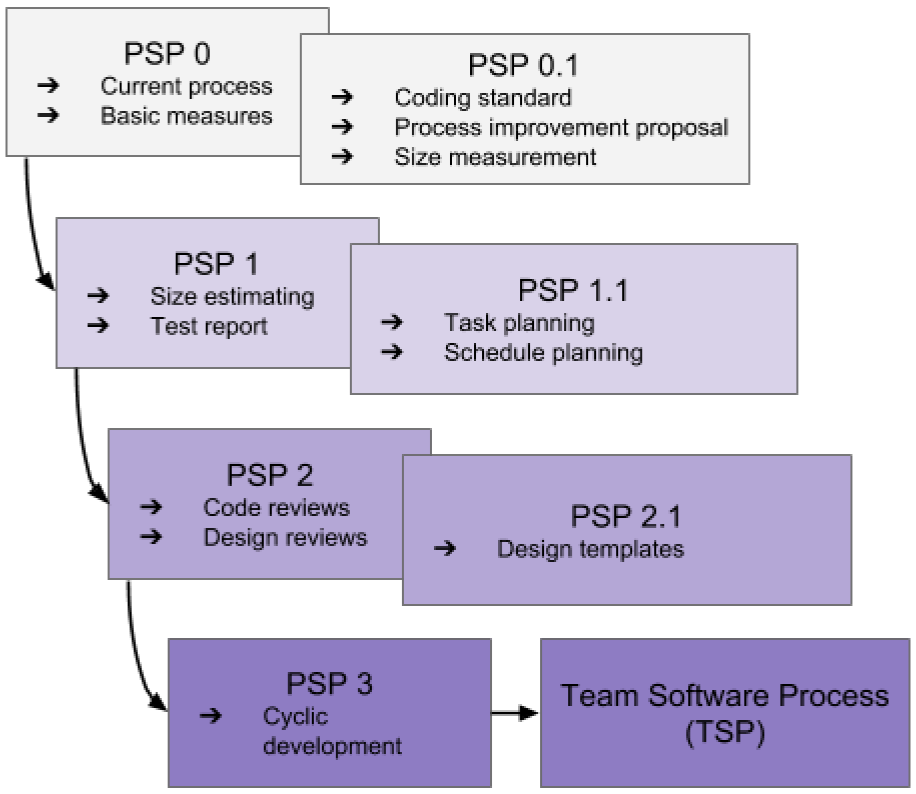


Figure 1. PSP levels (Humphrey, 1997)

The goal of each engineer is to reach the highest level – PSP 3. All the levels can be described as follows (Humphrey, 1997):

1. PSP 0 holds instructions for collecting measurements (development time and defects logs) that are used in process planning and as a base for improvement evaluation. At this level, engineers start to learn PSP basics.
2. PSP 0.1 collects metrics for size, suggests improvements for the process and a template that engineers utilize to note the process issues.
3. PSP 1 introduces a PSP analysis method that is used to evaluate the size of each task and define scopes of the evaluation based on historical data.
4. PSP 1.1 complements evaluation for schedule and recourse and earned-value tracking. This tracking approach helps engineers to assign the priority and to analyze the deadlines of each task.
5. PSP 2 reports quality measurements and estimation, design and code review checklist. The checklists may differ among engineers because each of them creates most appropriate based on defect data from the previous experiences.
6. PSP 2.1 introduces approaches to prevent defects and teaches engineers design specification methods.

7. PSP 3 trains design verification methods and approaches for adapting PSP to engineers' working conditions. At this level, the engineers become entirely familiar with PSP techniques.
8. TSP extends the PSP methods to the software team conditions.

The Team Software Process (TSP) process complements and expands the PSP and CMM to enable developers to work in development teams. The TSP demonstrates to engineers how to create an independent team and how to act as a useful part of a team (Humphrey 1999). Also, the TSP shows how to manage and support these groups of people, as well as provide an environment conducive to high efficiency. TSP has the following goals (Humphrey, 2000):

- Create independent and motivated teams that organize and monitor their work, evaluate goals and control their plans;
- Demonstrate to managers how to train and stimulate their teams and how to maintain their consistently high performance;
- Deliver improvement guidelines for companies with a high level of maturity;
- Accelerate your software development process by ensuring the correct and expected behavior of the CMM level 5.

The main advantage of TSP is that it demonstrates to engineers the way to produce high-quality software with estimated costs and on schedule. TSP provides engineers with information on how to track their workflow and how to monitor their plans (Tadayon, 2004).

2.2 Software Process Improvement

In the early days, Software Process Improvement and PSP were referred to each other (Ferguson et al., 1997). The most commonly used improvement models are the CMM as the reference process model, and IDEAL as the model to guide improvement. CMM assesses the performance of software companies across five levels of maturity and eighteen critical process areas and seeks to improve the development process. CMM addresses management practices; PSP defines the orderly approach for engineers to build software. In the scope of PSP, engineers train twelve of eighteen crucial process areas from CMM. As a result, personal software process training helps to amplify the CMM process improvement model. This can be achieved because engineers become more disciplined in their work estimation, a project tasks planning, tracking issues and improving the quality of the software they produce. According to the Krishnan and Kellner (1999) study, the level of professional skills of an engineer and the team as a whole, along with the consistent implementation of CMM practices, greatly facilitates reducing the number of defects in software development. Thus, it provides a direct link between the sequential improvement of software processes, personal growth, and the reduction of local defects in the final product. The Krishnan and Kellner (1999) research results show that projects can improve the quality of the supplied software by more consistently enhancing the qualifications of engineers in the CMM practice.

Another model, where PSP takes place, is IDEAL. The model proposes methods to control the Software Process Improvement (SPI) process. It is a technical approach that explains what to implement and less how to do it. IDEAL complements by Change Management, which focuses on employees and the company aspects, proposes methods to lead practitioners, persuades them to use the new processes, encourages them to change and improve the company's operations (Ferreira et al., 2010). It consists of five stages: initiating (justify to employees the real need for software process improvements), diagnosing (determine the future condition of the company with the SPI implemented), establishing (accumulate a focus group for SPI initiative, connect SPI methods to the practitioners), acting (contribution in extension for employees for comprehensive actions, SPI achieves short-run wins in the SPI initiative), leveraging (integrate achievements and creating of more changes, set the SPI methods in the company corporate culture). One of the issues for models such as CMM or IDEAL is that they concentrate more on technological aspects than on human aspects.

According to Beecham et al. (2003), these models have several weak points. High maturity organizations are more connected to organizational issues than organizations with low maturity. In the same time, low maturity organizations are closely connected to issues relating straight forward to projects such as quality, technologies, scheduling, and techniques. From an employee's viewpoint, top management of the company faces challenges with setting goals, realizing the company's politics and culture. Project managers suffer problems with budget estimating, scheduling and change management. Software engineers refer to issues with communicating within the team and the company, requirement management, testing, workflow documenting and using of auxiliary tools.

To succeed in adapting improvement models, Agile methodologies such as XP or Scrum can be used (Pino et al., 2008). In this case, organizations should initiate the improvement as soon as possible with a straightforward version of the model. Besides, an organization needs to be in enough stable state. When one of the SPI models is implemented, guided it systematically and coherently by specific technique and particular methods, in this case, all improvements need to be prioritized. Then, these improvements should be executed with the iterative and incremental approach, which leads to continued adoption of improved methods. Involving as many employees as possible, holding personal pieces of training, providing rapid feedback and delivering significant effect improvements in a short time frame helps to accelerate the model stages. In addition, one of the significant obstacles to overcome is the communication of employees within the team and the company, which improve the interaction process and improve the consistency of actions (Beecham et al., 2003).

2.3 Engineers' skills acquisition

Modern workers must be flexible in their skills, able to interact within a team, build communication with people of different mentality from countries around the world, and also understand customer and business requirements, and be open to constant innovation. Employers, in turn, expect each newly hired employee to have relevant knowledge and will be able to continuously develop their professional skills. Finding employees with such qualities is not an easy task (Cockburn et al., 2001). Thus, there is a need for an alternative approach to managing current employees, which will support organizational and potentially dramatic changes in times of active technological development.

The boundaries of the qualities and skills of each employee can be described using the "shape." If an employee has in-depth and extensive knowledge in only one single area, then, in that case, such a form called an I-Shaped. If an employee has the same in-depth experience in more than one field, then this is a Pi-Shaped (or H-Shaped). Dash-form describes those specialists who possess skills and knowledge in many areas, while not having an understanding of at least one of them. Finally, T-Shaped is maintained by people who are always and for a long time engaged in the study of various fields. Such people are always in the study of multiple tests, still open to communication with the outside world and, including, with the people with whom they work (Demirkan et al., 2015). These people have astute and critical thinking and imagination, quickly and constructively learn from their mistakes.

T-shaped specialists have the necessary breadth and volume of knowledge and experience, which contribute to rapid adaptation in new conditions, work in an ever-changing environment, as well as high communication skills in a team that includes many experts of different functionality, as well as specialists of different cultures and mentality. The increased flexibility of such specialists gives them innovative potential, with the help of which they are always ready to apply new methods, techniques, and tools in their work, and it also allows them to offer modern solutions in their field of work. In addition, their high communicative skills allow actively including in the work of customers, which gives a significant contribution to the workflow and the development of advanced services (Karjalainen et al., 2009).

Some companies are already taking steps to develop cultures that encourage T-shaped specialists. Selection, recruitment, and empowerment of employees are central issues in the transition to the service sector. A crucial part of the cultural and structural evolution is to attract the right people to support it; cultural and structural changes must be built to create the right environment for the formation of the necessary thinking and approaches. This means how to determine where the current employees are located and to determine where different types of people are most needed in the company where and they can function best (Hansen, 2001). Internal resources should be allocated to conduct this assessment and develop plans to attract new T-shaped talents where necessary. The key to bringing this needed talent is building a culture and a set of incentives that support the qualities they value. This can mean rethinking basic personnel practices such as accountability and compensation (Boehm et al., 2015). First of all, culture should appreciate the breadth of knowledge and intellectual flexibility that are the hallmarks of T-shaped people and encourage the creativity and cooperation on which they thrive. Companies may need, for example, to develop a range of support for various types of collaboration and creativity, ranging from unstructured spaces, where employees can gather for creative work and exchange information and ideas with structured processes using computer tools and project planning programs. The combination of two types of collaboration is essential because unstructured collaboration contributes to creativity, while structured cooperation contributes to efficiency.

The higher the qualifications and professional skills of a developer, the greater the likelihood that the improvement process will be executed better and in a shorter time, what makes a developer a part of SPI (Perry et al., 1994). The developer can acquire the necessary skills using the Dreyfus model or Bloom's taxonomy.

Bloom's taxonomy is a cognitive skills taxonomy that is used in many areas of education, including computer science (Azuma et al., 2003). Bloom outlined six categories of the cognitive field: knowledge, understanding, application, analysis, synthesis and evaluation. The sequence is organized from the lowest level that is a simple form of identification to the highest level that is the most abstract and complex form of cognitive skill. Based on a study that was done by Khairuddin and Hashim in 2008 Bloom's taxonomy focuses on identifying assessed learning goals in the computer science field, so it can be stated that integrating Bloom's taxonomy with computer science curricula made communication of teachers more effective. Other research works that have been carried out for specific areas of computer science in education using the Bloom taxonomy show the effectiveness of the tested automatic evaluation procedure for programming, the Bloom taxonomy levels for the three software developer sections and the Bloom taxonomy for human-computer interaction (Bourque et al., 2003).

The Dreyfus model provides a valuable topology for creating and improving the skills of an engineer. The idea of the Dreyfus Model of Skill Acquisition was firstly introduced in 1980, and it suggests that an engineer acquires a skill based on external training and, in general, proceeds through five phases from beginner to expert (Shinkle, 2009):

1. Beginner. Performing at this phase, an engineer, or junior engineer wants to know just enough to realize a project task. He or she works well with solid principles, and are not able to handle uncertainties. As a result, there is no place for discussions of policies. Also, it does not make much sense to show and prove benefits, since it will not have the necessary effect on awareness.
2. Advanced Beginner. Performing at this phase, engineers are still focused on the methods to execute tasks but are starting to realize some of the elements behind the principles. They require details as fast as possible to support their expanding insight. They desire to understand the advantages of learning and applying the principles and methodologies.

3. Competent. Performing at this phase, engineers are utilizing the principles to form their use of the practices. They are capable of realizing the long-distance and identifying when things are running quite well and when they are not. They are aware of the advantages expected, wishing to combine principles and methodologies to the advantages.
4. Proficient. Performing at this phase, engineers are considering the complete framework, considering ideological models to structure their thinking. They realize the interconnection of principles, methodologies, and advantages so well that they may not be capable of dividing them.
5. Expert. Performing at this phase, engineers operate from immediate personal understanding and having adopted principles so accurately, and they may have a problem interpreting their argumentation to colleagues. They are regularly considering better practices, and are specific, and defined frameworks and rules burden them. They may spot advantages as so evident that they lose sight of addressing them in discussions with others (Hall-Ellis et al., 2013).

One of the most effective ways to utilize the Dreyfus Model is to implement it in Agile software development. It helps to measure team maturity and plan coherent steps in improving team member skills (Weyrauch, 2006).

2.4 Combination of CMM, Agile and Dreyfus model

CMM can use any model of software development process, not just the “waterfall” but also Agile models, or a combination of the “waterfall” approach and iterative approaches inherent in Agile methods (Boehm et al., 2003). Combining traditional planning for an extended period at a high level of management and, for instance, daily Scrum meetings in small groups working on specific tasks usually always happens in large companies (Paulk, 2002). CMM can add more discipline to the development carried out by Agile methodology. For example, if the number of engineers working with one repository has increased, and the repository has become unstable, then a more rigorous procedure for delivering code to the repository can help correct the situation. This procedure falls into the CMM's “Configuration Management” and “Product Integration” process areas, thus following the recommendations described for these process areas can help in Agile project. In any project, from the very beginning there are project management, planning, and tracking, if take up a formal description of how this happens, it may turn out that the processes are entirely consistent with the CMM model. Finally, CMM's rejection by Agile methodologies supporters can be evident of a not very good understanding of this area. The combination of Agile methods and CMM occurs in many projects, CMM formally fixes many practices that are already used in developed and mature Agile-projects. An organization that follows only Agile methods may easily reach the third level of the CMM model (Pikkarainen et al., 2006).

Agile methods perform successfully where enough number of Competent and Proficient, according to the Dreyfus model classification, software developers are available. Once a developer becomes Competent he or she is ready and willing to take ownership of his work (Shinkle, 2009). Based on the study of Rong et al. in 2010 one of the ways to grow is PSP. It is argued that the PSP extends Agile using reliable methods that will provide a more valuable and productive direction for software developers. Furthermore, manageability and predictability, which are well established by the PSP, will profit software developers with more responsibility and planning. When the work of each software developer can be predicted, hence, the team's work starts to be more predictable.

Each Agile-team must have a critical mass of Experts. This does not guarantee a positive result, it is only a necessary requirement (Hunt, 2008). In addition, Agile team should have Experts in all critical areas of development. The team needs someone who can put together the requirements and analyze them; otherwise, the developer depends on the owner of the product. Besides, someone should be able to design software architecture. Moreover, the team needs someone who has suitable qualifications in testing to make sure that the product meets the requirements. A team that complies with these criteria and takes into account the above information may succeed in their project. Of course, such a team cannot overcome all obstacles, such as a very difficult client or inadequate management, but the chances are greater.

Bass, Allison and Banerjee in their article "Agile method tailoring in a CMMI level 5 organization" (2013) suggested the connection between CMM, Agile and Dreyfus model, which is presented in Figure 2. Also, the proposed connection will be used further in the results part, which is why the CMM will be considered further from the SPI point of view, and the Dreyfus model will be considered from the point of improving and expanding developer professional skills.

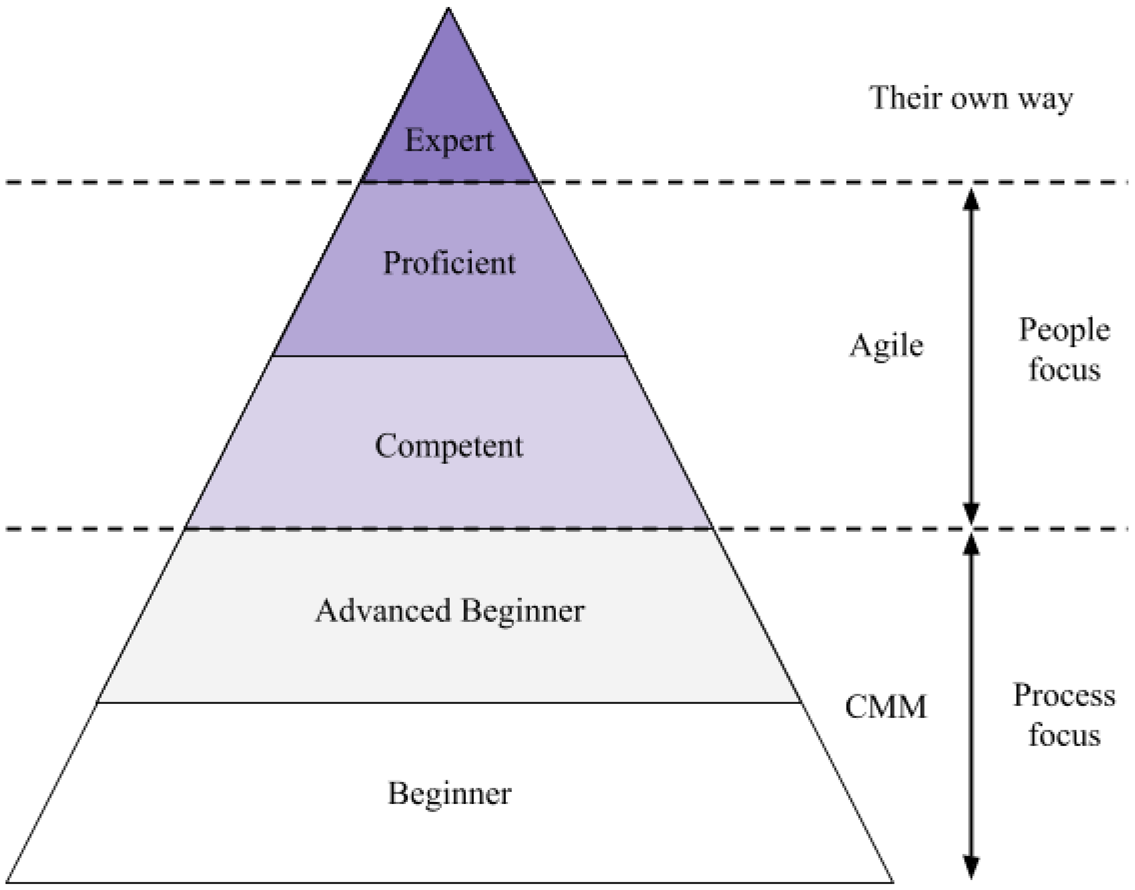


Figure 2. CMM, Agile and Dreyfus model (Bass, Allison and Banerjee, 2013)

Agile development is based on feedback, but the possibility of self-correction based on previous experience is feasible only if there are employees with a higher level of development. Agile development is a handy tool, but it does not work in teams created entirely from Beginners. Therefore, it is essential for each developer to acquire the necessary skills on the way from the Beginner to the Expert.

3 METHODOLOGY

To conduct the current research, the research paradigm is addressed. It was stated that the human aspects, or people factors, of software development, have a significant influence on software productivity, as well as the growing complexity of nowadays software products, the collaboration expected to develop software efficiently has become as essential as the human aspects of software creation (de Souza et al., 2009). This research paper is highly dependent on different human behavior, and how companies deal with it in daily workflow needs to be studied.

Qualitative research methods have been developed and are commonly used to manage the complexity of issues, including human behavior. Qualitative research methods were created to analyze the complexities of human behavior (for example, motivation, communication, and understanding). In software engineering, the combination of technological and social behavioral aspects allows the combination of qualitative and quantitative approaches to take advantage of both (Seaman, 1999).

A case study includes the study of a problem considered in one or several cases within a particular system. Case study research is a qualitative method in which the researcher examines a specific system (case) or various specific systems (cases) over time, in particular, through careful data collection, including multiple sources of information (e.g., measurements, interviews, audio and video materials, as well as documents and reports), and also describes the case and topics based on the case. For example, a number of programs can be selected (a multi-site study) or one program (on-site study) (Creswell et al., 2017). This data collection reveals a detailed description of the case, where the researcher describes in detail the history of the case, the chronology of events or the daily presentation of the actions of the case (Stake, 1995). Based on Kähkönen (2011) study, research process steps were adopted for this research and presented in Figure 3.

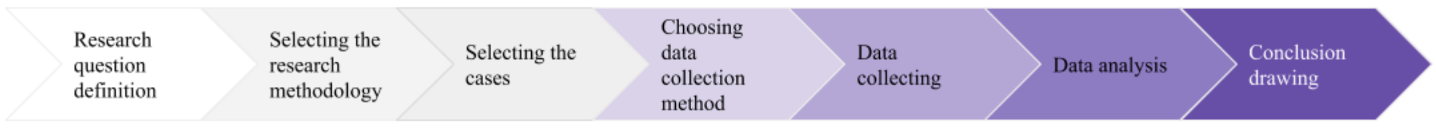

Research question definition
Selecting the research methodology
Selecting the cases
Choosing data collection method
Data collecting
Data analysis
Conclusion drawing

Figure 3. Case studies research process

3.1 Data Gathering

Data collection in case studies usually uses various sources of information. These sources include archival data such as documents or records, observations, and interviews (Yin, 2006). Therefore, a table for gathering project data was drawn up taking into account the research questions and aims to emphasize the specifics of the implementation of software projects related to the organization of the software development process and the management of project results, to clarify the relationship between developer skills and project results.

The design of the data collection table of the project was organized iteratively, and the table fields were updated several times after filling in projects. The fundamental approach to the formation of table fields is based on practical and research experience. These fields were made based on the listed aspects, derived from the project type and the project's sequence of actions. The literature was studied mainly after collecting and analyzing the results to form scientifically based research results.

All fields of the table were formed based on historical and current project documentation (Creswell et al., 2017). The first impulse to this decision was an interest in receiving an exact amount of information about the projects of companies and, if possible, a stimulating assessment.

The project data collection procedure was done in the following stages:

1. Collect historical data from task tracking applications;
2. Meeting with a business analyst from company A, discussing and collecting project data from an online task tracking system;
3. Data collection of current projects in company B from online task tracking system;
4. Filling the project template;
5. Analysis of the results, the formation of the analysis document;
6. Isolation of all problems and concerns to further improve the collection of project data.

The table was used to receive more structured data on the personal software process understanding among the companies. The process of working in some projects could change with time in company B since the organizational structure of the company, and the

IT department in particular changed. Therefore, it was decided to collect additional data on current projects as follows:

- Collect data from task tracking applications;
- Contact a representative;
- Sending questionnaires to representatives;
- Collection and analysis of data.

All collected data on 15 projects of two companies is presented in Appendix 1 and Appendix 2. Each table includes a description of each project, which includes the duration of the project, the requirements for the project carried out (as they were provided initially), the project start and end dates, the reasons for postponing the project’s deadline, the roles in the company that are responsible for the project, the duration of testing of each project and the number of errors detected during testing and developing new functionality.

The basic information about each case company:

Company A is an IT company. It provides a complete solution for the development of the company from consulting to the implementation of information systems. The company introduces modern automation methods and uses our experience to make customers’ company more competitive. All the project data was gathered based on work in the IT department, where roles of developers and business analysts are strictly divided.

Company B is a stone trading company. It was established in 2003. The company sells natural stone from around the world. These are granite, marble, onyx, travertine, quartzite, and many others. This is a slab stone for further processing and ready-to-use tile, and these are finished stone products. Company B partners include leading companies from Italy, Spain, Greece, Germany, China, India, Oman, Saudi Arabia, Namibia, Angola, Brazil, the USA, and the Russian Federation. All the project data was gathered based on work in the IT department, where IT developers combine roles of business analysts and coders.

3.2 Data Analysis

When several cases are selected, a typical analysis format is first detailed description of each case and topic within the case, or an in-situ analysis, followed by a case-by-case analysis - cross-analysis of the case, and approval or interpretation of the case value (Gioia et al., 2013).

The study used inductive method. Empirical data collected from projects have been studied continuously from historical documents and reports at a more general level to propose a solution for improvement. This method allows research to be more innovative and create innovative concepts that will provide theory and practice in software development.

3.3 Addressing validity

In a qualitative study, there is always possibility for different explanations, depending on the viewpoint of each party involved in the study, including readers. The following viewpoints should be recognized based on previous research:

- Reliability is associated with whether the results are convincing in terms of the issues studied;
- Portability is associated with whether the findings of the research unit can be assigned to a general theoretical statements;
- Reliability is associated with whether it is reasonable to repeat the study and whether it will be directed to the appropriate results;
- Confirmation is associated with verifying the discussions and conclusions of the study by others.

These viewpoints that need to be carefully discussed to ensure the accuracy and correctness of the conclusions.

4 RESULTS

Empirical data was gathered from 15 projects, executed in two companies. All the collected data is presented in two appendices of this paper. Table 1 summarizes the projects.

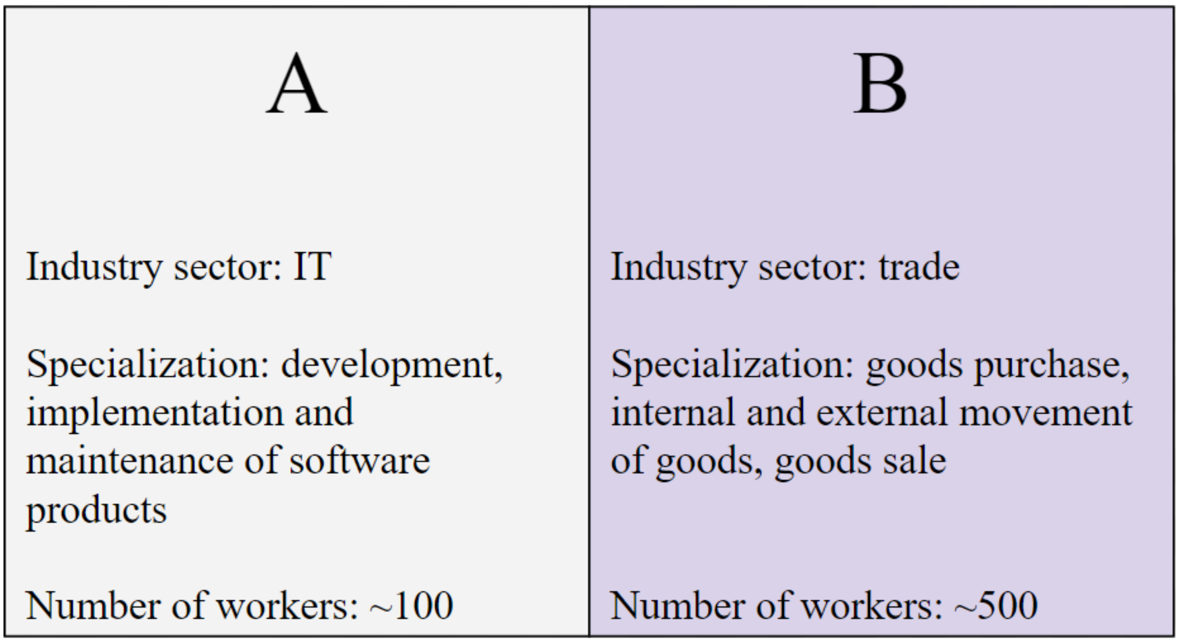
Table 1. Summary project statistics

Company	Number of projects	Total number of defects	Average number of defects per project	Number of projects with delays
A	6	14	**2**	2
B	9	58	**6**	6

Table 1 shows the difference in the number of defects that were made during the software development projects. Summary data reveals that the number of defects in company A projects was one third less than in company B. Also, more than half of the projects in company B were implemented with deadline delays. The reasons that led to such a difference in the quality of the developing software are reported in the section that analyzes the companies' workflows. Describes the structure of teams that are engaged in software development, which makes it possible to see the difference in the approach to the divide work. After that, the team's actions at each development stage are compared according to Agile methodologies. As a result, recommendations and improvements for the development of professional skills of the developer are listed.

4.1 Analyzing the company workflows

A brief overview of the companies in which this IT project data was collected is presented in Figure 4.



A	B
Industry sector: IT	Industry sector: trade
Specialization: development, implementation and maintenance of software products	Specialization: goods purchase, internal and external movement of goods, goods sale
Number of workers: ~100	Number of workers: ~500

Figure 4. Company overview

4.1.1 Company A workflow

Company A is an IT company that develops software to automate accounting in an enterprise or organization. All implemented and developed solutions are based on Russian software called «1C». Using standard products of the 1C company, company A builds, modifies or adjusts modules of the program, each of which is responsible for a particular part of accounting: accounting, sales, warehouse logistics, finance, and production. Each type of 1C product used can be a separate program that automates a particular type of accounting (for example, accounting software) or a complex structure of modules representing software of the Enterprise Resource Planning (ERP) class.

Each project at Company A is carried out strictly through the following stages:

Launch stage. At this stage, the project initiative is rejected or becomes a project. The primary mechanism for solving this problem is the project control process. In its framework, the following is done:

- Coordination of the project initiative with the sponsor and the customer;
- Identification of sources of economic results;
- Determination of the source (s) of financing and the marginal budget of the project.

Identification of key IT services required to implement a project idea. The essential IT services of the project initiative need to be compared with the existing portfolio of services: perhaps the services of similar content in the organization are already provided. The service level manager or his subordinates perform this work.

According to the results of the launch stage, the customer decides to launch the project initiative as a project, either to suspend the initiative or cancel it. In the first case, the customer sends a change request to the Service Desk of the information service, which is subsequently processed by the change manager. In the second case, the project initiative is suspended, for example, until the issue of funding is resolved. In the third - the project initiative is closed.

Selection stage. At this stage, a technological platform is determined that optimally implements the project initiative. This task is achieved through two processes: project control and change management. First of all, as part of the change management process, the information necessary to make a decision is collected:

- Expected benefits of the project will be implemented (in whole or part);
- Identification of possible options for the technical implementation of the project;
- Identification of the resources of the budget, employees, suppliers, and IT infrastructure;
- Estimation of the total cost of change for the options considered;
- Estimation of risks for each of the options.

Based on this collected information, the project team classifies the options for project implementation according to their effectiveness, considering benefits, costs, and risks. The project team (including its management), the customer and the project sponsor approve the selected choice, under the project management process. After that, the option is approved as part of the change management process: change manager (all projects), change committee (medium and large projects), organizational board (large projects). The definition of the technical platform allows the project team to determine the necessary project resources, what helps to create a project plan and budget.

Design stage. At the design stage, a “to-be” business process model is created that implements the project initiative. An important part of the business process "as it should be" is newly created IT services. These new services should be coordinated with the service level manager as part of the discussion of the conceptual design of the information system being implemented. Decisions on the conceptual project and based on the stage as a whole are made in the process of project control.

Implementation stage. From the project management point of view (as opposed to the control of considering projects), this is not one stage, but two - the development of a solution and its deployment. The development of the solution includes the settings and programming of the system being implemented, the creation of an array of reference data, and finally, a necessary test of the system, evaluating all the newly created functionality in the aggregate. As a result, the development of solutions should be obtained:

- Settings and code that implement the conceptual design;
- The results of the integral test of the system and its evaluation by the project team and the customer of the project;
- Updated estimates of IT infrastructure requirements, taking into account additional information received during the design and execution stages.

Based on this information, the change manager decides on the industrial deployment of the system. In the case of a favorable decision, deployment measures are included in the schedule of future changes. The actual deployment of the system is coordinated as part of the release management process. The project team performs preparation of the industrial exploitation environment and testing of the latter. Willingness to adopt a new information system for commercial operation is determined as part of the project control process. Information on IT infrastructure readiness is transferred from the change management process.

Operation stage. At this stage, two tasks are accomplished: evaluation of project implementation and planning for the further realization of benefits. As part of the project control process, the conformity assessment of the actual benefits realized with the executed benefits is ensured. In case of significant differences, the fulfilled and unfulfilled prerequisites, as well as the foreseen and unforeseen risks of the project are considered. At the same time, the service level management process provides information on the compliance of the received services with the expected. In parallel, a post-project analysis is conducted as part of the change management process, evaluating the planning and deployment of the IT infrastructure.

Implementing a project on time and within the allocated budget requires inevitable trade-offs between the customer’s needs and the resources of the project team and the existing IT infrastructure. As a result, the functional scope of the project, as a rule, reports, forms, requests and other means of presenting information to the end user, as a rule, suffer. Therefore, according to the results of the analysis, it is planned to implement the benefits outside the project, including the development of forms and reports, the installation of additional jobs, etc. These measures often do not require registration as a project or are implemented as a separate project. Anyway, it is necessary to plan IT resources and budget for them, which is ensured in the change management process.

4.1.2 Company B workflow

Company B lacks a defined and phased sequence of work on projects. Each employee receives requests directly from users, keeps a list of tasks and requirements personally in a convenient task management system for each and has the following responsibilities. The IT department of the company performs tasks as follows: implementing IT projects, ensuring the operability of information systems, providing information about new IT capabilities and technologies for managing them, clerical work for the department, maintaining the IT budget, accounting for IT assets, and providing IT staff. This department consists the following roles:

- The network technician identifies problems encountered during the operation of the network; analyzes user requirements; coordinates the process of setting up and maintaining network equipment; Ensures software and hardware network compatibility prepares the budget in the accounting field and ensures efficient use of resources; supervises less qualified technical staff.
- The programmer solves complex programming issues related to upgrading, modifying existing code or creating new code; establishes the sequence of operations for the input and computer processing of data; monitors testing and debugging software.
- The system administrator installs the software and hardware; monitors and optimizes the operation of computer operating systems; identifies software problems; analyzes user requirements, assesses additional opportunities for improving software performance.

- The head of the IT department manages all activities related to the maintenance of computing equipment; controls the process of selecting, installing, supporting software and hardware; controls the company's communications with IT service partners; manage the process of selection, training of specialists of the department, analyzes the results of their activities; leads the process of training employees.

The weak point in Company B structure is that there are no business analysts, who analyze user requirements, determine the software and hardware configuration. Lack of such specialists has led to lack of technical specifications, technical reports on software and hardware support. In addition, lack of employees with business analyst skills has led to low coordination in the process of testing and commissioning IT support and analysis of the complex issues of programming regarding the modification of the code of existing programs and the creation of code for new applications.

4.2 Companies projects analysis

Since projects in companies A and B are carried out according to Agile methodologies, the phases of Agile software development lifecycle shown in Figure 5 will be used to compare the projects.

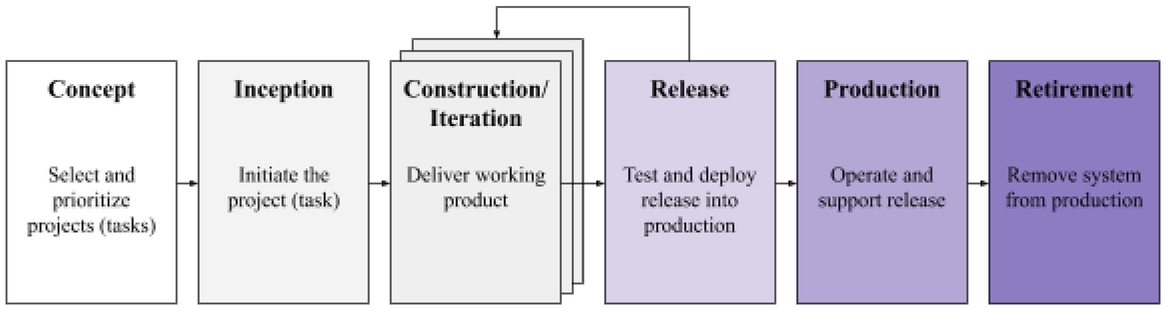

Concept
Select and prioritize projects (tasks)
Inception
Initiate the project (task)
Construction/ Iteration
Deliver working product
Release
Test and deploy release into production
Production
Operate and support release
Retirement
Remove system from production

Figure 5. Agile software development lifecycle (Ambler, 2002)

Based on Agile software lifecycle, each project phase can be compared between companies A and B, which will emphasize the differences between project management styles and further, based on this comparison, suggest development aspects of each company B developer to improve the quality of the software they develop. Table 2 presents the comparison of the project's phases according to Agile software development lifecycle (Ambler, 2002).

Table 2. Comparison of project phases between companies A and B

Project phase	Company A	Company B
Concept	Responsible person: project manager and business analyst. Activities: identifying and documenting business requirements and determining how they will be met, determining which products will be delivered, the necessary resources and skills required to complete the project. Tools: electronic document management system, project task tracking system.	Responsible person: stakeholder and developer. Activities: The head of one of the departments (the top management, sales, marketing, warehouse, procurement, logistics, accounting, finance) sends the agreed technical task in the form of business requirements to change the company's business process, the developer analyzes the requirements and lists the business requirements. Tools: emails, project task tracking system.
Inception	Responsible person: system architect, business analyst, developer. Activities: identifying and documenting elements that will be included in products or systems, analyzing business requirements and determining technical solutions, choosing the most effective and efficient solution. Tools: electronic document management system, project task tracking system.	Responsible person: developer. Activities: analysis (without any documentation) of the current system architecture to determine the presence of the necessary elements or modules that can be used to implement business requirements. Tools: project task tracking system.
Construction/Iteration	Responsible person: system administrator, developer. Activities: documenting how the components of products or systems technically work, how they interact with other elements and how they interact with hardware and software (high-level	Responsible person: system administrator, developer. Activities: analyzing (without documentation) requirements impact on hardware, analyzing of interaction with external modules or systems (for example, a

Project phase	Company A	Company B
	design), identifying and documenting modules in each element of a product or system and describing the function of each module, how it works and how each module interacts with other modules (detailed design). Tools: electronic document management system.	website). Tools: none.
Release	Responsible person: developer. Activities: developing the code, building the components. Tools: project task tracking system.	Responsible person: developer. Activities: developing the code, building the components. Tools: project task tracking system.
Production	Responsible person: business analyst, developer. Activities: combining all of the modules and components into a functioning product or system, testing. Tools: project task tracking system.	Responsible person: developer. Activities: testing of implemented requirements in terms of functionality for defects, at least of all testing from the business process point of view. Tools: project task tracking system.
Retirement	Responsible person: business analyst, developer. Activities: integrating and delivering the working iteration into production. Tools: electronic document management system.	Responsible person: users, developer. Activities: testing of realized requirements by users, adjustment of fulfilled requirements from the business logic point of view. Tools: project task tracking system.

Despite the fact that both companies use Agile methodologies in software development, their workflow and allocation of project resources are significantly different. As can be seen from the table of comparison of projects of two companies, in company A throughout the entire project implementation cycle, the IT team is participating in its entirety (architect, business analyst, developer, system administrator), where each specialist performs according to the role. While in company B at each project stage, the main work is performed by the developer from requirements gathering to testing. In particular, due to this approach to work in company B, there are more defects in the developed software than in company A.

Most of the defects and misunderstandings in projects of company B arise immediately at the first stage of the project - the collection and analysis of requirements. In company B, business representatives send their requirements directly to the developers, which requires a clear understanding of the developer of the company's business processes. In company A, business requirements and the primary collection of information about the project are engaged in specially designed specialists for this - business analysts and project managers. This approach helps to understand the business, analyze initial requirements and their compliance with the current IT solution, formulate and further adapt requirements for developers.

The next and significant difference in project management approaches is to documenting at each stage. In company A, each stage is recorded and stored in the corporate electronic document management system, which subsequently allows for a clearer follow up of changes in the project, monitoring the fulfillment of the requirement and keeping track of the terms and resources of the project. It is essential to note here that the documentation is not exhaustive since the development is carried out according to Agile methodologies. In company B, at the stage of analyzing the project architecture, as at all other stages, no documentation is performed, except for tracking tasks using a special tool that allows a developer to track tasks from creating a project task in the backlog to its execution as a Kanban board.

At the next project stage - at the Design stage - in company A each time a detailed analysis of the impact of current requirements on an existing IT solution, its modules or components is performed. The analysis is performed from a technical point of view (for example, to avoid duplication of code) and from a business point of view, so as not to disrupt any of the counts and to eliminate any conflict situations of software blocks or its components. Most often, this detailed analysis is carried out and documented in the form of building models and business processes of current software, which allows a developer to get a holistic assessment of project management. In company B, on the contrary, the analysis is performed at the level of consulting with other developers, users or based on their understanding of the system, which can later lead to certain misunderstandings.

According to the actions performed, the resources involved and the roles, Construction stage is no different in companies A and B. At the same time, there is a significant difference: programmers in company A begin to write code based on well-defined development requirements with well-built models and full awareness of how implemented requirements will work. The programmers of the company B begin to write the code, having the created tasks with a brief description and deadlines in the backlog and only understanding, not documented or modeled, in their mind. This approach in company B may lead to a loss of logical connections in the implementation of, unusually large projects since it is impossible to keep in mind all the conditions and their influence on other components.

Company A integrates new requirements into the client's software test environment by creating a new product release with all relevant changes and documents them. Further, the testing of realized requirements by developers and business analysts takes place: developers test the code for syntax errors, business analysts test the new functionality in terms of meeting the stated requirements by the business. In company B, developers test the code for syntax errors and perform surface testing for compliance of the implemented functionality with the business requirements.

At Implementation stage, company A specialists transfer thoroughly tested changes that meet the requirements from the test environment to the work environment, thus avoiding a large number of errors and misunderstandings. In company B, this stage of the project is essential for testing new functions from a business point of view, since it is at the moment that users themselves are testing new functionality and providing developers with feedback on the work done. With this company B approach, testing by users periodically leads to a complete shutdown of the software if the implemented functionality was not sufficiently tested or the requirements were initially misunderstood. This situation is especially critical for those developers who are just starting to work in company B, without knowing the specifics of accounting and other aspects of the company’s work.

4.3 Addressing the improvement of developer skills

4.3.1 Defining improvement fields

Effective communication is critical (Brodbeck, 2001). Without this, many problems arise: wasted time and possible money loses, poor quality of written code and inefficient software development, delays and unrealized expectations (Mockus et al., 2001). Effective communication saves money, time and labor, and this happens when the following requirements are met (Seaman et al., 1997):

- Message threads should be easily identifiable (not only email but any communication);
- The content of the message should be understood as quickly as possible;
- Describe events clearly in your message;
- Drive efficiently;
- Only involve the resources (people, tools, etc.) that are needed to complete the task;
- The rhythm of the project is smooth and measurable;
- Team leaders monitor progress on their project sites;
- People with different levels of responsibility are better involved in the project.

According to Paasivaara (2003), effective communication in IT projects has three important components such as legibility, traceability, and readability. Email is the main communication in many companies (Jackson et al., 2002), including companies A and B. In company A, there are standards for business communication within the company and communication with customers. In company B, especially in the IT department, there are no standards for formalizing business communication, what leads to disagreements and misunderstandings when communicating by email within the company between the IT department and all other departments of the company. This is because employees of IT department in company B write letters using technical terms, do not follow the letter structure, and do not even follow any standards when writing the letter subject. In company A, business communication standards clearly define aspects that company employees must follow when writing emails: the subject of the letter should be short and understandable with the project or task number, a body of the letter should be clearly structured with the responsible persons and a description of the current discussion problem.

One of the fundamental principles of software engineering is to ensure interaction between users and engineers (Leonard-Barton et al., 1993). Before software development begins, it is specialists “on demand” - analysts throw the “bridge” between customers and performers, which sets the level of communication and understanding between them, which is necessary to solve project problems. It is necessary to identify all possible sources of requirements relevant to the solution of project tasks. Only then can their influence on the project be determined. The prioritization, the uniqueness of the requirements transferred to the engineers, the connection between the requirements and their mutual impact on each other are all a result of a clear and unambiguous understanding of the sources of the requirements. There are many practices and approaches of requirements elicitation (Zowghi et al., 2005). Among them are the following:

- Interviewing - the traditional approach of extracting requirements; information must be analyzed and transformed into requirements; this information from the user is an “entry” into the requirements collection process, and the requirements themselves are an “output” of these processes;
- Scenarios - a context for gathering user requirements that define answers to the questions “what if” and “how it is done” concerning business processes implemented by users;
- Prototypes are an excellent tool for refining and/or detailing requirements (Zhang, 2007); there are different approaches to prototyping - from “paper” models to pilot subsystems, implemented as independent (in terms of resource management) projects or beta versions of products; often prototypes are gradually transformed into project results and are used to verify and approve requirements;
- “Clarifying meetings” is a term derived from the general practice of management and based on the ideas of cooperation of interested parties for a joint analysis of ways to solve problems, identify and prevent risks, etc. This is a particular form of meetings between project participants and interested parties on the part of the customer, which is devoted to discussing those questions that cannot be answered as a result of regular interviews and which require the involvement of a more significant number of people than just the user-analyst pair (De Lucia et al., 2010);
- Observation (observation) - implies the immediate presence of analysts and engineers next to the user in the process of the latter performing his work to ensure the functioning of business processes.

Design phase is one of the critical parts of software development. This phase includes the modeling of a future product (Van Lamsweerde, 2009). Most modern modeling techniques are based on the use of the Unified Modeling Language (UML) (Gomaa, 2005). UML is a language for defining, representing, designing and recording software systems, organizational and economic systems, technical systems and other systems of different nature (Lange et al., 2006). UML contains a standard set of diagrams and notations of multiple types. The main objectives in the development with UML are (Van Moll et al., 2002):

- Provide users with a ready-to-use visual modeling language that allows creating meaningful models;
- Provide mechanisms for extensibility and specialization to expand the basic concepts;
- Ensure independence from concrete development methods and programming languages;
- Implement a modeling language that must be both accurate and understandable, without too much formalism;
- Integrate best practices.

In company B, a developer focuses only on software development that leads to losing sight of the big picture. A company B developer must understand how the business works and be oriented in business processes, go beyond creating applications. In company A, business analysts are responsible for understanding business. A business-oriented developer, as a business analyst in company B, can suggest ideas for new applications that will subsequently improve performance. This is necessary so that the specialist can offer his insights that will help get the best result or profit of the company. A developer can find out about the presence of this quality by asking whether the respondent participated in improving the organization's business.

4.3.2 PSP extension and adaptation

Considering all the above characteristics, the expansion of the professional skills of the developer is presented in the form of a diagram in Figure 6.

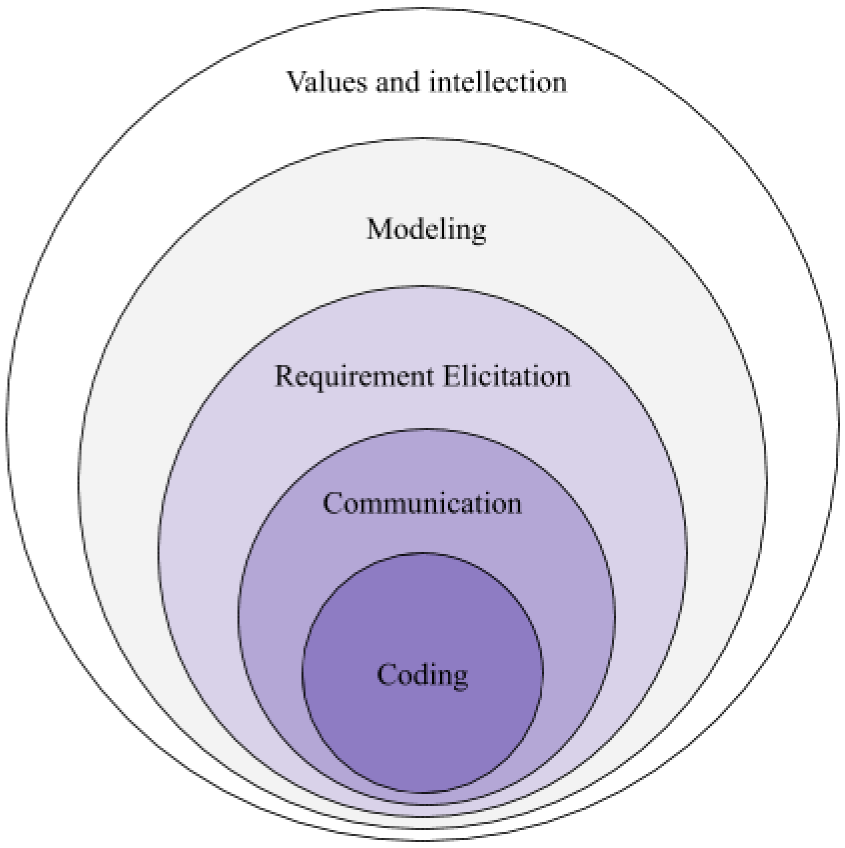


Figure 6. Expanded developer skills

Table 3 suggests improvements and additions for each level of the PSP according to specific characteristics. These suggestions allow adapting PSP-approach to work according to Agile methodology that will allow each developer to increase the level of communication with users and customers. In addition, more clearly extract and analyze the requirements for the developing software, based on requirements to build models of the future product and, as a result, deliver to the business exactly the product that is really needed and which will work as expected. Especially these additions will be valuable in those cases when no business intelligence in the team structure develops a software product, as described above using the example in company B.

Table 3. Additions and improvements for each level of PSP

PSP level	Improvements
PSP 0 holds instructions for collecting measurements (development time and defects logs) that are used in process planning and as a base for improvement evaluation.	Communication: defining a “personal user” for each project developer who will explain professional terms and business concepts. Requirements elicitation: studying aspects of the client's domain and business, the terminology, and purpose of the developing product. Modeling: creating a simple analysis model (or context diagram) that displays the place of the new system in the appropriate environment.
PSP 0.1 collects metrics for size, suggests improvements for the process and a template that engineers utilize to note the process issues.	Communication: find out from users what tasks they need to perform using software; discuss how the client should interact with the system for each such task. Requirements elicitation: creating patterns for identifying, analyzing, defining and validating requirements. Modeling: additions of the context diagram, definition of boundaries between the developing system and entities such as information systems, devices, and users.
PSP 1 introduces a PSP analysis method that is used to evaluate the size and establish the accuracy of the evaluation based on historical data.	Communication: if there is no clear and mutual understanding of requirements between developers and users, then create a prototype that will show the product concept more tangible. Requirements elicitation: analyzing the influence of requirements on each other. Modeling: a study of requirements modeling, which will allow identifying incorrect, inconsistent,

PSP level	Improvements
	missing and redundant requirements.
PSP 1.1 complements evaluation for schedule and recourse and earned-value tracking (to assign the priority and to analyze the deadlines of each task).	Communication: find out from the client the expected performance, efficiency, reliability, usability. Requirements elicitation: analyze the implementation of each requirement in terms of costs and acceptable performance in the intended environment; consider the risks of meeting each requirement, as well as conflicts with other requirements, dependence on external aspects and technical limitations. Modeling: construction of data flow and entity-relationship diagrams using the UML standard.
PSP 2 reports quality measurements and estimation, design and code review checklist.	Communication: define the process of submitting, analyzing and approving or rejecting changes. Requirements elicitation: based on user requirements, create functional test scripts and document the expected behavior of the product under specific conditions. Modeling: building a class diagram, sequence diagram, interaction diagram using the UML standard.
PSP 2.1 introduces approaches to prevent defects and teaches engineers with design specification methods.	Communication: record the dates of changes in the specifications of the requirements, the amendments themselves, their reasons, as well as the persons who made the changes. Requirements elicitation: track how developers regulatory actions affect the overall project that will allow a developer to evaluate the impact of this work. Modeling: a study of constructed models for flaws, correction of flaws.
PSP 3 trains design verification methods and approaches for adapting PSP to engineers’ working conditions.	Communication: coordination of models with stakeholders. Requirements elicitation: to conduct a retrospective of projects, also called the study of completed projects, familiarization with experience in the field of problems and ways of creating requirements accumulated during the work on previous projects.

PSP level	Improvements
	Modeling: creating test scenarios for models built based on previously collected requirements.
TSP extends the PSP methods to the software team conditions.	Communication: conducting surveys of potential users and discussions with them. Requirements elicitation: Capture team’s requirements development and project management efforts. This data will allow a team to assess compliance with the plans and more effectively plan the necessary resources for future projects. Modeling: creation, verification, and improvement of models by the whole team.

Following the steps of the PSP, taking into account the proposed additions in the areas of communication, extraction of requirements and modeling, will lead not only to a possible improvement in the quality of the written code but also to the regular participation of the developer in the company's business processes (Ahmed, 2013; Hall et al., 2002; Saiedian et al., 2000). Consequently, an understanding of how these business processes work will begin to form, with the result that the developer’s knowledge of the company's values will be formed. That, in turn, can lead not only to clear communication, to a specific collection of requirements and modeling but also can lead to proposals for improving the business process by the developer. A developer may offer innovative product features, new market opportunities, and business opportunities and think about how to satisfy customers.

According to the Dreyfus model, a developer who possesses such knowledge, having passed all the PSP levels with the proposed additions, can be classified as an Expert. And the effectiveness of Agile-team depends on the presence of experts in it, that is, a specialist who will not only be able to write code, but also collect and analyze user requirements, create an architecture design for the developing software, and also have the necessary knowledge and experience in testing, allow the product to meet customer requirements and expectations. Moreover, the expansion and dissemination of experts' knowledge and experience on the development team will have an impact on the pace of software development by ensuring the proper and anticipated behavior of the fifth CMM level.

4.3.3 Defining a developer maturity

Comparing the workflows and projects between companies A and B, as well as determining the areas of developer skills improvement fields, the final result is that the developer of company B needs to possess the qualities and skills of company A business analyst. Based on the analyzed data, the following aspects of developer growth in the field of software development using Agile methodologies were highlighted. The aspects proposed below will expand the professional skills of each developer, developing his or her competence in the analytical environment for working with requirements and users, customers.

Addressing acquisition of communication skills, a developer must be able to use different means of collecting information and present this information in various ways in a normal and understandable language, as Wiegers and Beatty (2013) highlighted. A professional in this field possess simultaneously developed communication skills, knowledge of the psychology of interpersonal communication, technical knowledge, knowledge of the subject area of business and personal qualities suitable for this job. Critical success factors are patience, a sincere desire to work with people and listening to them. Active listening involves eliminating interference, maintaining a proper posture and eye contact, as well as repeating key points to reinforce their understanding. A developer needs to instantly grasp what people are saying and be able to read between the lines to discover things that they are embarrassed to say. Learn how colleagues prefer to communicate and avoid imposing developers own filter of understanding on customer statements. Look for assumptions that would emphasize the thoughts of others or their interpretation and create a comfortable communication environment. The ability to organize a friendly atmosphere at meetings or seminars to clarify requirements is one of the necessary skills of a developer, based on De Lucia and Qusef (2010) study. A neutral observer who has the experience of interviewing, observing and creating comfortable conditions for communication will generate trust in the group and reduce tensions between businesspeople and IT staff. During active and sometimes non-constructive communication, a developer may deal with a large amount of erratic data. To cope with the data and build a coherent whole, a developer will need exceptional organizational skills, as well as patience and perseverance to isolate the main ideas from chaos.

Requirements elicitation skills include the ability to interview and ask questions, what also follows from Zowghi and Coulin study (2005). People retrieve much of the information about the requirements in the course of the conversation, and therefore the analyst must be able to communicate with different people and groups - only this way he will be able to identify their needs. It may be challenging to work with senior managers or overly self-confident or aggressive people. To clarify the essential requirements of users, a developer must ask the right questions. For example, users usually focus on the expected behavior of the system. However, a significant part of the code will handle exceptions, so a developer should determine the possible error conditions and the system response to them. As a developer gains experience, he or she will learn to ask questions that reveal and clarify uncertainties, differences of opinion, assumptions and unspoken expectations. Hence, a competent developer is capable of thinking at several levels of abstraction. Sometimes a developer needs to go from the higher information to the details. In some cases, based on the need for one of the users to formulate a set of requirements that will satisfy the majority of users of this class. Critically evaluate information obtained from different sources to resolve conflicts, separate users' passing wishes from their real needs and distinguish solutions from requirements. It can be achieved by improving observation skills. A careful developer will remember comments made in passing that may be valuable (Beyer and Holtzblatt, 1997). Observing how the user performs his or her duties or works with an existing application, an experienced analyst will identify moments that the user has not even mentioned. Observation sometimes helps to direct the discussion to a new direction to identify additional requirements that no one said anything about (Wiegers and Beatty, 2013). The main result of the requirements creation process is a written specification with information for customers, the marketing department and technical staff. A developer must be able to read critically and efficiently, since he or she has to look through many materials and need to understand their essence quickly, and should be fluent in the language and clearly express complex ideas.

Improving modeling skills, a developer should be able to work with a variety of tools, starting with ancient flowcharts and structured analysis models (information flow diagrams, “essence - connection” diagrams, etc.) and ending with the modern UML (Unified Modeling Language). Some of these tools are useful in communicating with users, others with developers (Van Lamsweerde, 2009). The developer should explain to other project participants the value of using these methods and how to work with their data.

Expanding his or her professional skills in field of communications, requirements elicitations and modeling leads to better understanding of company values and intellection, what creates confidence in conversations with employees holding various positions in the organization and helps to explain to customers the essence of the requirements creation and software development processes.

5 DISCUSSION

5.1 Addressing research questions

Based on the results of this work, proposed improvements for customizing the personal software process, which was formulated based on project data collected in two companies using Agile methodologies, are aimed at improving the developer’s skills in analyzing the company's business and its processes. These ideas lead to the formation of a developer with flexible and broad professional skills, to create a closer and more direct connection between the developer and the customer, as well as to a deeper understanding of the customer's requirements. Analyzed data emphasizes that company B has more defects in developed software than company A. Firstly, Company B does not pay attention to software testing. Secondly, the developers of company B interact with end users directly, which leads to misunderstandings of the requirements between two parties.

The first problem of lack of testing can be solved and automated using TDD that is answering RQ 1 (How to improve software quality and reduce software development defects?). Since company B lacks a well-defined approach to testing developing software products, one of the ways to improve the quality of the final product and reduce defects during its implementation is a creation of software products based on the repetition of short development cycles, starting with the initial writing of tests and after developing the functionality of the program during one implementation cycle. This technology is called Test Driven Development. At the beginning after writing the test, this test will not be successful, since the corresponding code has not yet been written. To write a test, the developer must clearly understand the system requirements. This distinguishes development by testing from methods when the code is already written. This forces the developer to focus on requirements before writing code (Beck, 2004). A development cycle following the TDD methodology is presented in Figure 7.

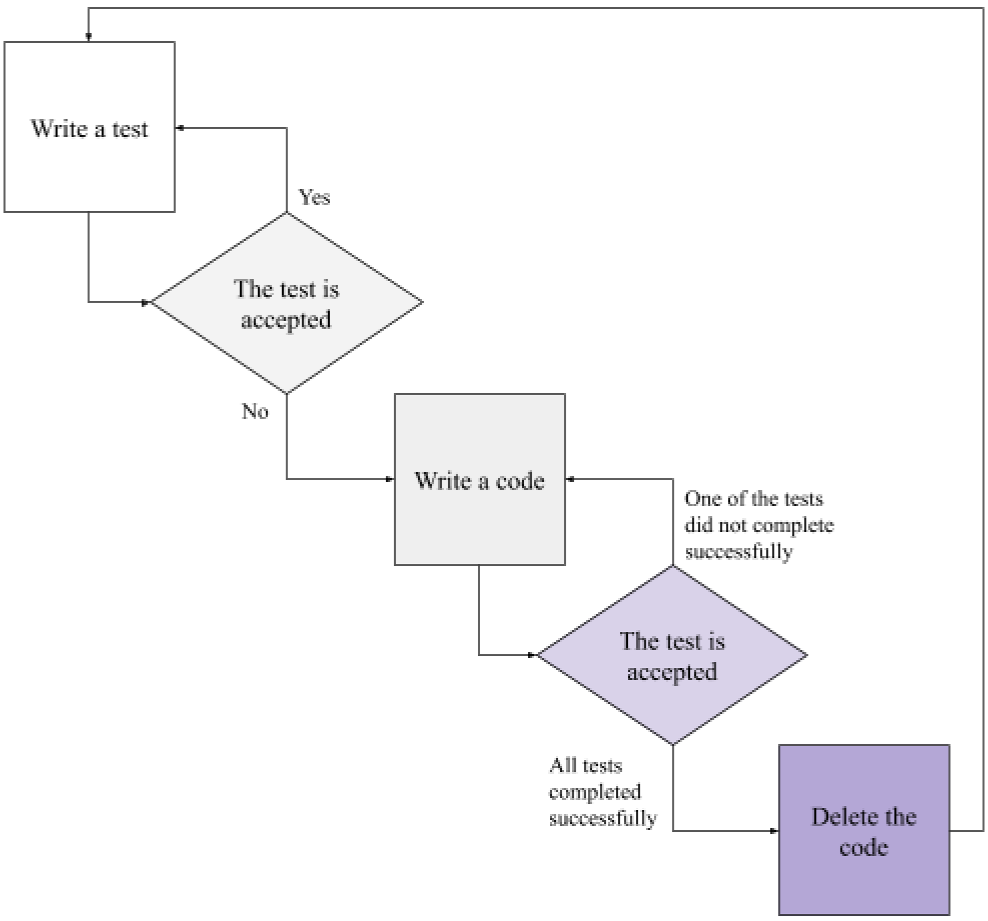


Figure 7. Test Driven Development schema (Beck, 2004)

Following the TDD methodology, the code will be completely covered with tests, as it implies the initial writing of the test (Beck, 2004). Two basic rules must be followed when developing on TDD methodology:

1. New program code is implemented only after a unit test is written that fails;
2. It is written precisely such a volume of newly implemented code that is necessary for this test to pass (Kaner et al., 2000).

Creating tests before writing the code of a class, method, or anything else, the programmer ponders in advance how it can be used and will have a positive effect on the quality of the implemented module, project architecture as a whole (McConnell, 2004).

The second problem, the problem of misunderstandings between developers and users, can be solved by developing the skills of the company's developers B. Study by Ferguson et al. (1997) states that it is possible to improve the qualifications of developers using the PSP approach. In this case, as can be seen from this work, the PSP develops technical skills of the developer. Therefore, in order to apply PSP approach to company B developers, it is necessary to adapt and expand it. According to studies of Ahmed (2013), Hall, Beecham & Rainer (2002) and Saiedian & Dale (2000), when working with users, it is necessary to possess analytical skills in the areas of communication, requirements gathering, modeling and understanding of the company's business values. Based on these fields, improvements and additions were suggested to each of PSP level, what is answering RQ 2.2 (How can developer maturity be defined?).

Since Agile's approach to development is used in companies A and B, the developers themselves must also be flexible in their skills. To avoid downtime and increase delivery speed, a developer should be a T-shaped specialist. These are people who have their own deeply studied specialization, as well as the desire to develop in related areas. They can easily substitute colleagues, thereby preventing the loss of production speed. At any time, the team can be rearranged in such a way as to solve almost any task within the predicted period (Boehm et al., 2015). Therefore, answering RQ 2.1 (What are the most important developer skills needed to be improved during Agile software development?) and taking into account the results of the Karjalainen et al. (2009) research, we can distinguish the following skills for development:

- Autonomy. The person or team should be able to make his or her own decisions. It is required that everyone understands their responsibility for what they do. In turn, the management should allow the person to make mistakes within acceptable limits if something goes wrong.
- Motivating goal. Each member of the team must understand the common goal and be aware of the contribution that his part of the work carries. If not, then people usually come to work for a salary. They know that they will make their piece, throw them "over the fence" to the next stage and go home.
- Mastery. That allows people to show their best qualities, learn something new, and be the best in something. And in this plus T-shape - no one is forced to develop in one direction, a developer is free to study what he or she likes as part of the team.

These three conditions are supported by an Agile framework called Scrum (Martin, 2002). In it, the work is divided into sprints, and each set a goal that brings the team one step closer to achieving the final result. The team has no management, so its members make their own decisions and are responsible for the outcome. This stimulates the search for ideas to improve the product. So that the speed of work does not slow down, if someone from the team members is on vacation or is sick, the participants should be able to substitute each other. Therefore, they have to learn about related professions.

Summarizing the answers to research questions 2.1 and 2.2, I can answer to RQ 2 - How to improve personal software development workflow? Expanding their competence in the analytical field, each developer will be able to more optimally use their working time, better plan and better manage user expectations.

All described aspects of the development of developer’s professional skills are additions to the PSP methodology in the areas of communication, working with user requirements, modeling and understanding the features and values of a business. These aspects of development are described in detail for each stage of the PSP in the results section of this work. These additions to the PSP are designed for software development projects when working on Agile methodologies. These statements of improvements and influences on the development of professional skills of developers and business require verification in the future, and may also have limitations, which are described below in the subsection of the limitations of this study.

5.2 Limitations and scope of further research

In this study, specific aspects were identified and described for improving and adapting the personal software development process when developing according to Agile methodology, and possible ways for developing the developer's skills were investigated. The study has several limitations that should be overcome in the following studies. First, the number of reviewed projects and companies is quite small. For more accurate results of the study of human factors and determine their trends in software development, it is necessary to collect more data. That also allows applying the proposed ideas and approaches in practice.

Secondly, there is a restriction on a geographical basis, since all the companies that collected project data are located on the territory of one city. As the human factor is studied, this study needs to be expanded geographically and collect data on projects in companies that are located in other countries and on different continents, which will allow future research to take into account aspects such as mentality, climate, and features of the work schedule.

Thirdly, recommended ideas for improving the developer's professional skills should not only be tested on a more significant number of projects and in different companies, but also in a longer time perspective. Since the human factor is being studied in the software sphere, it takes time to develop the necessary professional skills and their application, which reveals the need for observation for a long time.

6 CONCLUSION

In scopes of this research, the topic of adapting the personal software development process according to Agile methodology was considered. The study was aimed at identifying the professional skills of the developer to improve the quality of the developing product.

Data on 15 Agile projects that were collected in two companies were collected and analyzed, which allowed to develop improvements to each level of the personal software process, as well as to identify potential developer professional skills to improve.

Analysis and comparison of the collected data showed that in company A, where a team with a defined distribution of roles is clearly built when working with projects, projects are completed with almost no delays, and the errors developed in the products themselves are minimal. In the second company, company B, where the software developer not only writes the code but also collects and analyzes the requirements and communicates directly with the customer, in each project, there are delays mainly because of misunderstandings during requirements elicitation process, which leads to a high number of defects compared with projects in company A.

To improve the developed software quality, improve the process of extracting and analyzing requirements, as well as reducing the number of defects, the following areas of development of professional competencies of developers in company B were highlighted: communication, requirements extraction, modeling, awareness of company business values. To improve and expand the skills in each of these areas, appropriate development approaches have been proposed so that each developer becomes more flexible in his or her skills and knowledge. According to the identified aspects of the development of the programmer, improvements and recommendations were proposed for each level of the PSP, which allows a developer to adapt this methodology for developers who work in the conditions of development according to Agile methodologies. Each recommendation includes additions to a specific level for developing communication skills, collecting and analyzing customer requirements, modeling business processes and software development architecture, and understanding the values and goals of a business, which allows meeting customer expectations and offers improvements to business processes of the company.

If considering the contribution of this study from the point of view of the scientific community, this work examines the possibilities of improving the influence of the human factor in software development.

REFERENCES

Abrahamsson, P. 2002. Agile software development methods: Review and analysis. Espoo: Technical Research Centre of Finland.

Ahmed, F. 2013. Soft Skills and Software Development: A Reflection from Software Industry. International Journal of Information Processing and Management, 4(3), p. 171.

Ambler, S. 2002. Agile modeling: Effective practices for eXtreme programming and the unified process. New York: Wiley.

Azuma, M., Coallier, F., & Garbajosa, J. 2003. How to apply the Bloom taxonomy to software engineering. In Eleventh annual international workshop on Software Technology and Engineering Practice, IEEE, pp. 117-122.

Bass, J. M., Allison, I. K., & Banerjee, U. 2013. Agile method tailoring in a CMMI level 5 organization. Journal of International Technology and Information Management, 22(4), p. 5.

Beck, K. 2004. Test-driven development: By example. Computing Reviews, 45(5), p. 271.

Beecham, S., Hall, T., & Rainer, A. 2003. Software process improvement problems in twelve software companies: An empirical analysis. Empirical software engineering, 8(1), pp. 7-42.

Beyer, H., & Holtzblatt, K. 1997. Contextual design: defining customer-centered systems. Elsevier.

Boehm, B., & Mobasser, S. K. 2015. System thinking: Educating T-shaped software engineers. In Proceedings of the 37th International Conference on Software Engineering-Volume 2, IEEE Press, pp. 333-342.

Boehm, B., & Turner, R. 2003. People factors in software management: lessons from comparing agile and plan-driven methods. Crosstalk-The Journal of Defense Software Engineering.

Bourque, P., Buglione, L., Abran, A., & April, A. 2003. Bloom's taxonomy levels for three software engineer profiles. In Eleventh Annual International Workshop on Software Technology and Engineering Practice, IEEE, pp. 123-129.

Brodbeck, F. C. 2001. Communication and performance in software development projects. European Journal of Work and Organizational Psychology, 10(1), pp. 73-94.

Cockburn, A., & Highsmith, J. 2001. Agile software development: The people factor. Computer, (11), pp. 131-133.

Creswell, J. W., & Poth, C. N. 2017. Qualitative inquiry and research design: Choosing among five approaches. Sage publications.

De Lucia, A., & Qusef, A. 2010. Requirements engineering in agile software development. Journal of emerging technologies in web intelligence, 2(3), pp. 212-220.

de Souza, C. R., Sharp, H., Singer, J., Cheng, L. T., & Venolia, G. 2009. Guest Editors' Introduction: Cooperative and Human Aspects of Software Engineering. IEEE software, 26(6), pp. 17-19.

Demirkan, H., & Spohrer, J. 2015. T-shaped innovators: Identifying the right talent to support service innovation. Research-Technology Management, 58(5), pp. 12-15.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. 2012. A decade of agile methodologies: Towards explaining agile software development. The Journal of Systems & Software, 85(6), pp. 1213-1221.

Ferguson, P., Humphrey, W. S., Khajenoori, S., Macke, S., & Matvya, A. 1997. Results of applying the personal software process. Computer, 30(5), pp. 24-31.

Ferreira, M. G., & Wazlawick, R. S. 2010. Complementing the SEI-IDEAL Model with Deployers' Real Experiences: The need to address human factors in SPI initiatives.

Gioia, D. A., Corley, K. G., & Hamilton, A. L. 2013. Seeking qualitative rigor in inductive research: Notes on the Gioia methodology. Organizational research methods, 16(1), pp. 15-31.

Gomaa, H. 2005. Designing software product lines with UML, IEEE, pp. 160-216.

Hall, T., Beecham, S., & Rainer, A. 2002. Requirements problems in twelve software companies: an empirical analysis. IEE Proceedings-Software, 149(5), pp. 153-160.

Hall-Ellis, S. D., & Grealy, D. S. 2013. The Dreyfus model of skill acquisition: A career development framework for succession planning and management in academic libraries. College & Research Libraries, 74(6), pp. 587-603.

Hansen, M. T. 2001. Introducing T-shaped managers. Knowledge management's next generation. Harvard business review, 79(3), pp. 106-16.

Humphrey W. S. 1997. Introduction to the Personal Software Process, Addison Wesley Longman.

Humphrey, W. S. 1994. The personal process in software engineering. In Proceedings of the Third International Conference on the Software Process. Applying the Software Process, IEEE, pp. 69-77.

Humphrey, W. S. 1999. Pathways to process maturity: The personal software process and team software process. SEI Interactive, 2(2), pp. 1-17.

Humphrey, W. S. 2000. The Team Software Process (sm)(TSP (sm)). Carnegie Mellon University, Software Engineering Institute.

Hunt, A. 2008. Pragmatic thinking and learning: Refactor your Wetware. Pragmatic bookshelf.

Jackson, T. W., Dawson, R., & Wilson, D. 2002. The cost of email within organizations. In Strategies for eCommerce Success, IGI Global pp. 307-313.

Johnson, P. M., & Disney, A. M. 1998. The personal software process: A cautionary case study. IEEE Software, 15(6), pp. 85-88.

Johnson, P. M., Kou, H., Agustin, J., Chan, C., Moore, C., Miglani, J., and Doane, W. E. 2003. Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In 25th International Conference on Software Engineering, 2003. Proceedings, IEEE, pp. 641-646.

Kähkönen, A. K. 2011. Conducting a case study in supply management. Operations and supply chain management, 4(1), pp. 31-41.

Kaner, C., Falk, J., & Nguyen, H. Q. 2000. Testing Computer Software Second Edition. Dreamtech Press.

Karjalainen, T. M., Koria, M., & Salimäki, M. 2009. Educating T-shaped design, business and engineering professionals. In Proceedings of the 19th CIRP Design Conference– Competitive Design. Cranfield University Press.

Khairuddin, N. N., & Hashim, K. 2008. Application of Bloom's taxonomy in software engineering assessments. In Proceedings of the 8th WSEAS International Conference on Applied Computer Science, pp. 66-69.

Krishnan, M. S., & Kellner, M. I. 1999. Measuring process consistency: Implications for reducing software defects. IEEE Transactions on Software Engineering, 25(6), pp. 800-815.

Lange, C. F., Chaudron, M. R., & Muskens, J. 2006. In practice: UML software architecture and design description. IEEE software, 23(2), pp. 40-46.

Leonard-Barton, D., & Sinha, D. K. 1993. Developer-user interaction and user satisfaction in internal technology transfer. Academy Of Management Journal, 36(5), pp. 1125-1139.

Martin, R. C. 2002. Agile software development: principles, patterns, and practices. Prentice Hall.

McConnell, S. 2004. Code complete. 2nd ed. Redmond (WA): Microsoft Press.

Mockus, A., & Herbsleb, J. 2001. Challenges of global software development. In Proceedings seventh international software metrics symposium, IEEE, pp. 182-184.

Paasivaara, M. 2003. Communication needs, practices and supporting structures in global inter-organizational software development projects. In ICSE International Workshop on Global Software Development.

Paulk, M. C. 2002. Agile methodologies and process discipline. Crosstalk, pp. 15-18.

Paulk, M. C. 1998. Using the software CMM in small organizations. In The Joint 1998 Proceedings of the Pacific Northwest Software Quality Conference and the Eighth International Conference on Software Quality, Portland, OR, USA, pp. 350-361.

Perry, D. E., Staudenmayer, N. A., & Votta, L. G. 1994. People, organizations, and process improvement. IEEE Software, 11(4), pp. 36-45.

Pikkarainen, M., & Mantyniemi, A. 2006. An approach for using CMMI in agile software development assessments: experiences from three case studies.

Pino, F. J., García, F., & Piattini, M. 2008. Software process improvement in small and medium software enterprises: a systematic review. Software Quality Journal, 16(2), pp. 237-261.

Rong, G., Shao, D., & Zhang, H. 2010. SCRUM-PSP: Embracing process agility and discipline. In 2010 Asia Pacific Software Engineering Conference, IEEE, pp. 316-325.

Saiedian, H., & Dale, R. 2000. Requirements engineering: making the connection between the software developer and customer. Information and software technology, 42(6), pp. 419-428.

Seaman, C. B. 1999. Qualitative methods in empirical studies of software engineering. IEEE Transactions on software engineering, 25(4), pp. 557-572.

Seaman, C. B., & Basili, V. R. 1997. Communication and organization in software development: an empirical study. IBM Systems Journal, 36(4), pp. 550-563.

Shinkle, C. M. 2009. Applying the Dreyfus model of skill acquisition to the adoption of Kanban systems at software engineering professionals (SEP). In 2009 Agile Conference, IEEE, pp. 186-191.

Stake, R. E. 1995. The art of case study research. Sage.

Tadayon, N. 2004. Software engineering based on the team software process with a real world project. Journal of Computing Sciences in Colleges, 19(4), pp. 133-142.

Van Lamsweerde, A. 2009. Requirements engineering: From system goals to UML models to software (Vol. 10). Chichester, UK: John Wiley & Sons.

Van Moll, J. H., Jacobs, J. C., Freimut, B., & Trienekens, J. J. M. 2002. The importance of life cycle modeling to defect detection and prevention. In 10th International Workshop on Software Technology and Engineering Practice, IEEE, pp. 144-155.

Weyrauch, K. 2006. What are we arguing about? A framework for defining agile in our organization. In AGILE 2006 (AGILE'06), IEEE, p. 8.

Wiegers, K., & Beatty, J. 2013. Software requirements. Pearson Education.

Yin, R. K. 2006. Case study methods. Handbook of complementary methods in education research, 3, pp. 111-122.

Zhang, Z. 2007. Effective requirements development-A comparison of requirements elicitation techniques. Software Quality Management XV: Software Quality in the Knowledge Society, E. Berki, J. Nummenmaa, I. Sunley, M. Ross and G. Staples (Ed.) British Computer Society, pp. 225-240.

Zowghi, D., & Coulin, C. 2005. Requirements elicitation: A survey of techniques, approaches, and tools. In Engineering and managing software requirements, Springer, Berlin, Heidelberg, pp. 19-46.

APPENDICES

APPENDIX 1: Company A projects

№	Project	Project duration (lead time)	Requirements	Start date - End date	Delay reason	Responsible person (position)	Testing time	Number of errors
1	The system of replenishment of goods in warehouses - version 1	4 w	1. Data management interface with required selections. 2. The calculation of the minimum and maximum values of the stock of goods in stock warehouses. 3. The estimate of the approximate cost of the stock replenished. 4. Calculation of terms of replenishment: delivery from the supplier, shipping between company warehouses.	29.01.18 - 04.03.18	clarification of customer requirements	COO	1 w	5

№	Project	Project duration (lead time)	Requirements	Start date - End date	Delay reason	Responsible person (position)	Testing time	Number of errors
2	The system of replenishment of goods in warehouses - version 2	4 w	1. Search function for non-optimal routes for shipping and delivering goods from the supplier and between warehouses. 2. The function of blocking changes in data on warehouses by directors of departments.	05.03.18 - 01.04.18		COO	1 w	3
3	Report on how satisfied is the need for warehouses and in what time frame	1 w	1. Calculate the date of delivery to the warehouse - full cycle (the period the need was identified, the time of the order to the supplier, the number of days of delivery, the date of receipt at the central warehouse, the number of days of delivery from the central warehouse to the warehouse of the department).	02.04.18 - 02.05.18	clarification of customer requirements	COO	3 d	4
4	Report of how satisfied is the need for warehouses and in what time frame - version 2	1 w	1. Add the planned quantity that the unit consumes. 2. Calculate the percentage of implementation of the plan for a given formula. 3. Calculation of late deliveries.	03.05.18 - 10.05.18		COO	3 d	1
5	Report of how satisfied is the	1 w	1. Add the calculation of the date of import deliveries.	10.05.18 -		COO	3 d	

№	Project	Project duration (lead time)	Requirements	Start date - End date	Delay reason	Responsible person (position)	Testing time	Number of errors
	need for warehouses and in what time frame - version 3		2. Add the amount of weekly consumption of goods in the warehouses of departments. 3. Add a check for all items in the supply chain to the warehouse: a demand order has been created, a request for a supplier or a transfer order has been created, goods have been received from a supplier or moved from another warehouse, a warehouse receipt has been created.	17.05.18				
6	Report of how satisfied is the need for warehouses and in what time frame - version 4	2 w	1. Calculate guaranteed product delivery dates and compare them with planned dates and actual delivery dates.	17.05.18 - 31.05.18		COO	5 d	1

APPENDIX 2: Company B projects

№	Project	Project duration (lead time)	Requirements	Start date - End date	Delay reason	Responsible person (position)	Testing time	Number of errors
1	Program update (once a month)	1 w	1. Update the program to a new version. 2. Transfer changes of the IT department to the latest version.			CIO	2 d	15
2	The new approach of KPI calculation for the sales department	2 w	1. Create a form for storing employees ratings. 2. Recommended pricing mechanism based on clients type. 3. Store bonus calculation parameters in time. 4. Registration of the amounts of documents verified by the accounting department.	01.06.18 - 24.06.18	constant clarification and change of requirements	CSO, CFO	2 d	10
3	CRM module implementation	4 w	1. Adding CPM functions to the Clients, Contact Persons and Users directories. 2. Adding CPM software modules to the main program. 3. Setting up user rights.	25.06.18 - 22.07.18		CSO	1 w	7
4	CRM telephony module implementation	1 w	1. Adding CPM telephony module to the main program. 2. Setting up calls. 3. Setting up user rights.	23.07.18 - 06.08.18	adaptation of the outdated version of the telephony module implementation instructions, equipment	CSO	5 d	5

№	Project	Project duration (lead time)	Requirements	Start date - End date	Delay reason	Responsible person (position)	Testing time	Number of errors
					settings by system administrators			
5	Automation of sales agents workflow	2 w	1. Creating a form for storing data on agency fees. 2. Creating a bundle with the customer’s order. 3. Controlling the amount of the agency fee when saving the customer’s order. 4. Creating a report to control the execution of agency transactions and rewards for them. 5. Two weeks after the registration of the agency transaction, automatically send to sales managers notice of agent remuneration. 6. Adding agency fee transactions to financial documents.	07.08.18 - 03.09.18	the emergence of new requirements and differences in the implementation process	CFO	1 w	4
6	A form of a stone detailed view	2 w	1. Adding a function for saving photos. 2. Storing stone photos. 3. Adding a feature to assign a separate discount or price for a specific stone. 4. Service for sending pictures to customers. 5. Adding a picture of the stone when taking goods to a warehouse.	04.09.18 - 18.09.18		COO	1 d	2

№	Project	Project duration (lead time)	Requirements	Start date - End date	Delay reason	Responsible person (position)	Testing time	Number of errors
7	The new approach of KPI calculation for the back office	2 w	1. Create a rating system so that an employee may be rated based on specific KPI (range 1-10). 2. Rating store monthly, then calculate the quarterly ratio. 3. The calculated coefficient is taken into account when calculating the premium for the quarter.	04.09.18 - 25.09.18	long waiting for feedback, deviations from the initial task in the form of new requirements	CFO	5 d	5
8	Bonus Calculation System for Branch Directors	4 w	1. Link accounting of goods sold with shipped from stock. 2. From the number of goods sold and shipped for the month deduct returns of goods from the client, the services of sales agents, adjustments to the distribution of sales between branches. 3. Divide the goods sold and shipped into categories depending on their profitability, calculate the coefficient of the premium for each group of goods. 4. Calculate customer debts, calculate the penalty for overdue payments, calculate the coefficient for reducing the premium, and curb the final premium for the quarter by this ratio. 5. Calculate the quarterly bonus director only with a certain percentage of the	26.09.18 - 09.10.18	deviations from the initial task in the form of new requirements	CFO	14 d	3

№	Project	Project duration (lead time)	Requirements	Start date - End date	Delay reason	Responsible person (position)	Testing time	Number of errors
			implementation of the sales plan for the branch for the quarter.					
9	Automation of the system of accounting for product discounts when selling to customers	5 w	1. Automatically calculate the discount to the client when making a sale. 2. Setting a limit on the installation of a discount on a product for a sales manager so that he (or she) does not violate the company's regulations on providing discounts to customers. 3. Provide for the possibility of setting restrictions on discounts for specific items of goods. 4. Provide the ability to set limitations on discounts, depending on the status of the client (only four possible status). 5. Consider the size of the discount to the client when calculating the monthly bonus of the sales manager.	10.10.19 - 12.11.18	delay due to unclear requirements	CFO, CSO	10 d	7