University College of Northern Denmark

System development report

Homework platform

Dmai0914 Group1

Cosmin Gabinat

Iliyan Stoev

Janis Indrans

Nikita Cimaskevic

I5

Content

Introduction 2

Problem statement 2

Plan driven versus agile development elaborated through concrete methods 3

Method choice 7

Size of the project 7

Size of the team 7

Customer organizational culture 7

Primary project goal 8

Dynamics in the requirements 8

Quality Assurance 8

Quality criteria and architecture 9

Reflections on methods 12

Sprint 0 12

Sprint 1 13

Sprint 2 14

Sprint 3 16

Conclusion 17

References 19

Introduction

In the 21st century the computer technologies have become available to a higher percentage of the world's population. In more developed countries, especially in western countries, the personal computers and the smartphones are available to mostly everyone, these technologies being part of their daily life. With such a high availability, the computer technologies became an important tool of the community, especially in the storage and the administration of different types of the data. Being widely used, different purposes and requirements appear, creating the need of different computer software to run on these machines and manipulate the data according to the requirements.

Schools are part of the institutions that deals with the large amount of data. Thousands of students and employees create, submit and update the new data to the school every day. This process can be problematic and time consuming. This is why lately, it became more and more common to find the computer technologies implemented in the schools. These technologies help in dealing with sorting, reading and storing the data. However, the use of the computer technologies and the computer software in the schools has a very wide variety of purposes. One specific purpose of the computer technologies in the schools is represented by the use of the online platforms required for submitting the documents. This helps in saving both the time and the physical resources like a paper. These are the reason that this type of the online platform represents a valuable asset for any modern school nowadays.

Problem statement

The problem of delivering the homework in the elementary schools can be sometimes a time consuming activity. The children may forget to bring it to the school, and their parents cannot keep a track of their children's activity.

Considering the above mentioned information, this project is aiming on formulating and answering a problem statement that will improve the issue of computer technologies use in relation to the process of submitting homework in the schools.

The problem statement is formulated as follows: How can the schools avoid the daily time and the resource consuming process of delivering homework, by using the computer technologies? Could this be designed so it can be supported by different platforms and devices?

The computer software is mainly created by the programmers, using one or more programming languages from a wide variety of the languages available on the market. The programing language is rarely a problem when developing a piece of software, even though sometimes this software or a program becomes very large and complex. However, when dealing with the development of a large program, an appropriate development method is required in order to keep a track of the progress, to be structured and flexible. In term of the software development processes or methods, similar to the programming languages, there are also a variety of different options available. According to the requirements of the project, the team needs the skills to select the appropriate development method to fit their needs. Some of the most popular methods of development are UP (unified process), SCRUM, which is part of a larger development method called Agile, and lastly there is Kanban. Additionally there are a lot of other tools to be considered along these development methods.

Each of these methods is different and comes with its own features and the sets of pros and cons. In the next chapters of this report, two different developments methods will be discussed from the point of view of a developer, comparing the working process and the flexibility. In the end, one method will be chosen for the development of a given project. And for this particular case the choice will be reasoned.

Plan driven versus agile development elaborated through concrete methods

Both of these types of development are good for the specific types of the projects. Plan-driven development mostly aims for the perfection, planning everything ahead, lots of documentation, it

is usually split in phases and every phase has to be done and "singed off" before the next one begins. In agile development the processes are planned incrementally starting from the most important ones and the new ones added as the development continues. The priority is to do the current increment and to release the working functionality of the software quickly so it could be used in the practice but still keep working on the rest of the software. Agile manifesto principles clearly describe the differences between the plan-driven and the agile development [agile manifesto]:

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

This means that agile methodologies are focused on keeping customer close to the project by involving him as much as possible instead of making a lot of agreements, planning and contract negotiations, trying to deliver working software quickly instead of making a lots of documentations and welcomes changes in requirements without a lot of modifications in projects overall instead of modifying all the previous work.

The progress of the plan-driven development is measured by the plan which was made from the beginning; however in the agile development progress is not that visible to be measured as the software grows by adding new functionality and the requirements change very often. To address this issue Schwaber, Beedle and Rubin proposes to use the Scrum agile method to provide a framework for organizing agile projects and, to some extent at least, provide external visibility of what is going on [Schwaber and Beedle 2001; Rubin 2013]. The Scrum follows principles from agile manifesto. The other way to address this issue is to use Kanban method [kanban]. Kanban also provides with a framework for organizing projects but works on slightly different principles:

- Start with existing process
- Agree to pursue incremental, evolutionary change
- Respect the current process, roles, responsibilities and titles
- Leadership at all levels

The Kanban method starts with the existing roles and processes and stimulates continuous, incremental and evolutionary changes to the system however Scrum does not have a specific role except scrum master, which may change, and the product owner, which is the customer or other stakeholder representative. In Scrum every member of the developer team is usually involved in everything. All the principles mostly aimed to the roles which are not that important for the Scrum. However Kanban has fewer rules than Scrum, higher degree of freedom, needs more experienced developers [Kanban slides]. Both of these are also referred as agile development methods. They also might be combined with other methods.

The projects are not always perfect and requirements may change all the time as the time goes. When changes have to be made in the plan-driven development it is expensive as the previous phases and documentation has to be modified. In agile development the changes are done all the time as the customer is closely involved in the development by using informal communications rather than formal meetings with written documents.

Waterfall model is typical plan-driven development which is split in five phases [waterfall model]:

- Requirements definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

As mentioned before every next phase can begin after previous is done.

Incremental approach is typical agile development where the processes of the specification, design and implementation are interleaved [Incremental development]. According to agile development techniques Extreme Programming (XP) is one of the most significant methods in agile development [XP]. In contrast to waterfall model XP practices different approaches of development:

- Planning Game
- Small releases

- Metaphor
- Simple design
- Define test first
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 37 hour week
- On-site customer
- Coding standards

It doesn't have any phases and everything works incrementally and done in many iterations. The principles of XP are also based on the agile manifesto.

When it comes to maintenance the plan-driven development outperforms agile development. As a result of detailed documentation of the software the maintenance can be done much easier. That's where agile development is weak as documentation is really small. To address this issue you must keep customer involved in process.

- The plan-driven development is good for:
- Embedded systems where the software has to interface with hardware systems
- Critical systems where there is need for extensive safety and security analysis of the software specification and design
- Large software systems that are part of broader engineering systems developed by several partner companies

Agile development is good for:

- situations where informal team communication is possible
- situations where software requirements change quickly

Method choice

In order to fulfil whole project requirements, make system development efficient within given period and take in to consideration all the risks within new the system, appropriate software development methodology should be chosen. In terms of software development, methodology includes procedures, tools and different techniques which help developers implement the new system. There are two main methodologies to choose between Heavy Methodologies and Light Methodologies, which also called plan driven and agile. There is also possible to combine two methodologies to achieving greater result [Selecting methods].

When selecting the appropriate methodology some factors which have an influence on the project must be taken in to consideration:

- Size of the project
- Size of the team
- Customer organizational culture
- Primary project goal
- Dynamics in the requirements

Size of the project

The project size factor affects choosing methodology drastically. Usually the biggest projects increase number of people working on it, and it should be well process documented in order to have a better communication across large groups.

Size of the team

Depending on the project complexity and size, different amount of developers may work on it. To have higher efficiency in the big groups it is more appropriate to use heavy methodologies. For small groups it is reasonable take light methodology for the purpose of to be more dynamic by interactions and clear responsibilities.

Customer organizational culture

There are different types of contracts between client and development team which determine the relationship between them during development process. If customer is interested in the close

cooperation with the team and has intention to have a contact with it as often as possible and the light methodology should be taken in consideration. Otherwise the plan driven method is going to be more suitable for that purpose.

Primary project goal

This factor relies on the final purposes of the project which determines if the project is safety critical and requires high assurance with predefined requirements, so the plan driven development fits better. If the project is not safety critical it is better to apply agile methodology, in order to assure that there are possibilities to add new features and methods in the system.

Dynamics in the requirements

Do we have stable requirements and they can be determined in advance? If we answer this question yes we may use plan driven method, otherwise agile fits better.

Quality Assurance

Quality assurance (QA) is the definition of processes and standards that should lead to high-quality products and the introduction of quality processes into the manufacturing process [SE9thEdition].

Software quality is a subjective process where the development team has to use their judgment to decide if an acceptable level of quality has been achieved or not, by following already established software standards, testing, use some practices proposed by agile developing methods: for example, pair programming, doing reviews, test first and etc.

When talking about Quality Assurance there are certain questions needed to be asked in order to consider whether or not the software is fit for its intended purpose:

- Have programming and documentation standards been followed in the development process?
- Has the software been properly tested?
- Is the software sufficiently dependable to be put into use?
- Is the performance of the software acceptable for normal use?
- Is the software usable?
- Is the software well-structured and understandable?

In order to the above asked questions to be answered, tests should be carried out. Testing against what?

How can the development team state that the software is ready to be handed in to the Product Owner?

The general assumption in software quality management is that the system should be tested against its requirements, therefore the decision whether or not the desired functionality has been achieved, should be based on these tests. The mentioned tests could be done by the development team, or like in some organizations, an external testing team has been assigned for this job.

Requirements are playing huge role in a software system. Usually being set by the Product Owner, they define if the demanded functionality has been implemented and how the system as a whole is operating [Functional or Non-Functional]. If the Functional Requirements are not exactly met, the users can always find their workaround to do what they are up to, but if Non-Functional Requirements are not met, it can make the system totally unreliable or too slow making it practically impossible to achieve user's goals.

Therefore, it can be said Non-Functional requirements are taking the bigger part of the quality assurance process, because pretty functional interface is nothing when the performance for example it's not in its place.

Quality criteria and architecture

The quality criteria is a criteria that can be used to judge operations of the system instead of specific behaviors. In system development quality is classified as non-functional requirement of service requirements. Non-functional requirements are often called quality attributes. As it is stated on MSDN Microsoft webpage there are 13 common quality attributes which are categorized in 4 specific areas [MSDN Microsoft]:

Design qualities: conceptual integrity, maintainability, reusability.

Run-time qualities: availability, interoperability, manageability, performance, reliability, scalability, security.

System qualities: supportability, testability.

User qualities: usability.

Each of these attributes provides the key issues and the decisions you must make when building your software:

- **Conceptual integrity** is responsible for defining the consistency and coherence of the overall design.
- **Maintainability** is responsible for the system's ability to undergo changes with a degree of ease.
- **Reusability** is responsible for the system's ability to continue operating in the expected way over time.
- **Availability** is responsible for defining the proportion of time that the system is functional and working.
- **Interoperability** is responsible for the system's or different system's ability to operate successfully by communicating and exchanging information with other external systems written and run by external parties.
- **Manageability** is responsible for defining how easy it is for system administrators to manage the application, usually through sufficient and useful instrumentation exposed for use in monitoring systems and for debugging and performance tuning.
- **Performance** is responsible for indication of the responsiveness of a system to execute specific actions in a given time interval.
- **Reliability** is responsible for the system's ability to continue operating in the expected way over time.
- **Scalability** is responsible for the system's ability to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged.
- **Security** is responsible for the system's capability to reduce the chance of malicious or accidental actions outside of the designed usage affecting the system, and prevent disclosure or loss of information.
- **Supportability** is responsible for the system's ability to provide information helpful for identifying and resolving issues when it fails to work correctly.

- **Testability** is responsible for measuring of how well system or components allow you to create test criteria and execute tests to determine if the criteria are met.
- **Usability** is responsible for designing the application interfaces with the user and consumer in mind so that they are intuitive to use, can be localized and globalized, provide access for disabled users, and provide a good overall user experience.

Some of these attributes are measured in metrics but some of them are hard to measure. For example performance may be measured with time and space but usability can't be measured in metrics so developers have to be more creative.

The plan for implementing non-functional requirements is detailed in the system architecture. To design the architecture Ian Sommerville suggests that it is required to make a decisions based on these attributes [Architectural design decisions]. It is important to decide on how the architecture will look, how many platforms should it work on, if it's a web based application, should there be more clients (like web client and desktop client), is the software will be split and placed around the separate hardware and locations, what database should it use and so on. For this reason there are many architecture patterns that allow you to decide how you going to build your software and point out all the advantages and disadvantages of using them, and explains in which situations those may be used. Ian Sommerville described the 5 most common architectures: MVC(Model-View-Controller), Layered Architecture, Repository, Client-server and Pipe and filter architectures [Architectural patterns]. Every each of them is unique and good for specific software and might be similar to each other. System architecture should be well documented and viewed from many angles like - logical view, physical view, process view and development view, but developers still argue about it [Architecture views]. Also users of agile methods claim that detailed design documentation is mostly unused and is a waste of money and time. To ensure quality while planning the architecture it's suggested to use these quality attributes and to analyze the trade-offs between them. The importance and priority of the attributes differs from system to system.

Reflections on methods

Following the process of comparing the available development methods, the Agile framework has been chosen, and more Scrum and XP features. First of all, the duration of the project has been divided into four sprints, and a so called "Sprint 0" that actually represented the planning that took place before starting the first sprint.

Also, in accordance to the Scrum process, the Scrum team consisted of a Product Owner, the Development Team and the Scrum Master. Working in a team of only four persons, the product owner and the scrum master were in the same time part of the development team, and the roles were consistent throughout the duration of the project. However, this setup of the Scrum team covered one of the values of the XP, more exactly the "onsite customer".

Sprint 0

Prior to starting sprint 1 a Scrum team has usually a period of time also known as "sprint 0", meant to prepare the development environment and take care of the other requirements, in order to be able to start working on delivering business value from first day of sprint 1.

Our team spent sprint 0 on populating the Product Backlog with the first user stories, which have been previously prioritized by the Product Owner, according to their business value and return on investment. These user stories have been estimated by playing the planning game. Each member gave their estimation on the complexity of the story, by using points. Since the development team lacked the practical experience in developing in the .NET framework, the planning game revealed some considerable differences in the estimation of the stories. However, these differences have been discussed, and the planning game has successfully helped estimating the user stories in order to know the amount of points the team is able to cover during one sprint.

Next steps consisted of creating the database, setting up the environment to be ready for writing code, setting up the repository, connecting the continuous integration and unit test platforms, preparing the scrum board and burn down chart, and not lastly, planning sprint 1. At the end of sprint 0, the Scrum board has been populated with the user stories that had the highest priority and their respective tasks in the "To-Do" column.

However, even though sprint 0 covered a lot of preparation work, the development team has later realized that during this sprint not enough effort has been put into discussing and agreeing on the overall design and the architecture of the project. This led to more time being spent in the second and third sprint, on making changes to the code developed in the respective previous sprint.

Sprint 1

This period represent the beginning of the actual work that creates business value. The heart of Scrum is a Sprint, a time-box of one month or less during which a "Done", useable, and potentially releasable product Increment is created [Ken Schwaber and Jeff Sutherland]. The first day of this sprint, started with the first Daily Scrum Meeting, which was set to take place every day of a sprint at exactly 08:45 AM. Having already decided on using pair programming throughout the whole duration of the project, in order to ensure the collective ownership, and having already the pairs in place, this first Scrum Meeting outlined each pair's plan for the day, and facilitated the selection of tasks.

Even from the first sprint the pair programming did not work as expected. Not everyone felt comfortable and productive using this approach and two of the members of the development team chose to work separately on different tasks. However, the other pair continued using the pair programming throughout the whole duration of the sprint, with the exception of few minor tasks that were completed individually, as it felt more productive this way.

In what concerns the Test first approach, the development team planned to use it from the beginning of sprint 1. This did not happened however, as due to lack of experience in this area, the development team found this approach more complicated and time consuming. This is why during sprint 1, testing took place after the completion of each individual task.

As mentioned before, due to not enough attention given to preparation during sprint 0, coding standards and other principles as DRY (Do not Repeat Yourself) have not been respected, and this lead to spending a large amount of time from sprint 1 on refactoring.

The burn-down chart has been updated almost every day of the sprint 1, and as seen in the picture below, the tasks have been completed before the deadline. This meant that the

development team has over estimated the value of their user stories, and that they are able to take more points in the next sprint.

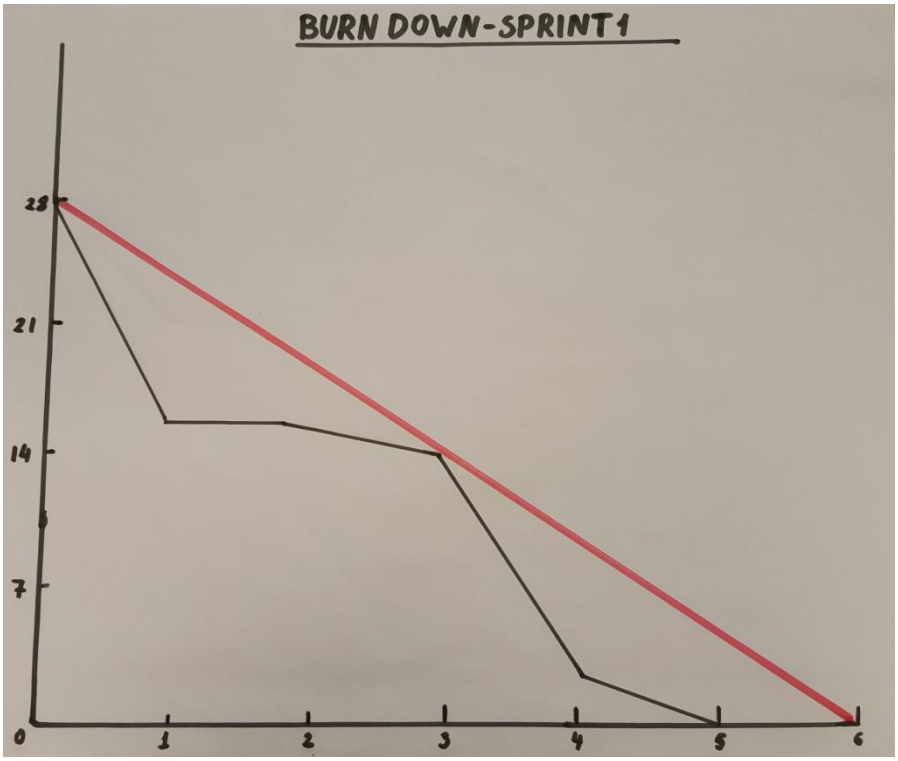

Figure 1 Burn down chart

After the sprint review and retrospective, the development team ended the sprint and started planning the second spring of the project.

Sprint 2

As soon as the first tasks of the sprint were in progress, the development team realized the lack of proper planning of design and architecture for the project. This problem pushed the development team to find a quick fix for the tasks in order to deliver the expected business value, and rely on a later refactoring of the code to implement the coding standards and quality.

During this sprint, the "Test first" concept has been put in practice, and most of the tasks were carried in this manner, designing the tests before satisfying them. This helped developing much simpler code that meets the requirements of the test.

As mentioned earlier, during sprint 1 the development team managed to fulfill the tasks before the sprint ended, meaning that they were able to increase the number of points taken during the

sprints. This is why the burn down chart for sprint 2 shows an increase in the points, from 28 to 30, despite the decrease in the sprint period from 6 to 4 days. As seen in the picture below, the development team kept their progress at a more or less constant pace, unlike sprint 1 when the burn down chart presents a more imbalanced progress. Even so, it can be noticed that similar to sprint 1, the second sprint also started by burning a large amount of points, followed by a somewhat less productive day, with very little points burned. This happened also as a consequence of poor planning. The tasks taken in the first day of the sprint have not been planned thoroughly and so it has been harder to connect them with the following tasks. This is why the development team ended up burning very few points during day two of both first and second sprint, and spending time on redesigning some of the tasks taken during the first day.

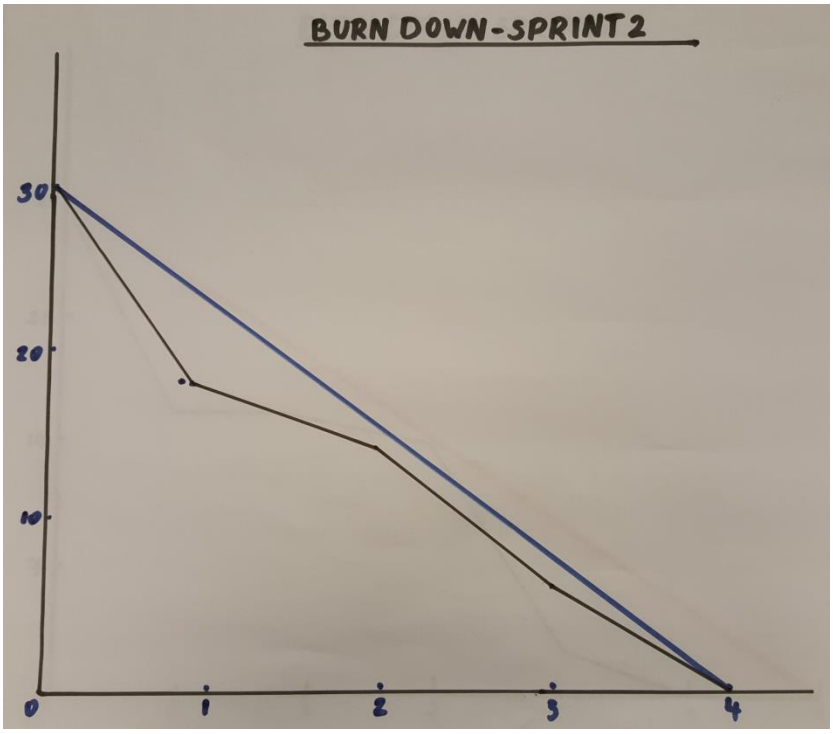

Figure 2 - Burn down chart Sprint 2

Even after increasing the number of points taken during sprint 2, the user stories and their tasks have been completed during day 3 of the sprint. However, as seen in the burn down chart, during the last day an approximate of 5 points have been burned, successfully meeting the deadline of the sprint. This last 5 points represent the redesign of some tasks finished earlier, but which required additional development to connect successfully with the rest of the code. These tasks

have been left to “In-progress” column of the Scrum board until the last day, when they were released with the rest of the code.

Sprint 3

As the sprints were passing, the group became more and more confident in its skills. For Sprint 3 it has been decided to take the most important task for the whole project, plus a much simpler one, which lead the sprint estimated points to be almost double compared to the previous 2 Sprints. With the completion of the chosen tasks the project was about to be pretty much done, which was an enough reason for the team to work hard against fulfilling them. Even though it wasn’t the last Sprint, the group decided to be done with all the planned functionality in order to have more time to focus more on technical documentation in Sprint 4, which is as important as the software developed as well.

The Daily Standup Meeting was held as it was scheduled from the beginning of the project even though the “Standup” part was missing. On some days some group members were late, but this didn’t have any negative effect on the work process.

At this point the group could afford to continue practicing Pair Programming due to the fact the tasks were going well according to the plan for the sprint. It was going surprisingly well for one of the couple programmers, where the lower level one improved his skills a lot since the beginning of the whole project, where the other pair was not using it since it didn’t go well in Sprint 1.

The confidence floating in the group affect on Test First approach also. It became more and more easier and natural to write test code first, compared to Sprint 1 for example. The understanding of what this XP Practice was useful with (helps the developer understands the required behavior of the new code), helped a lot in actually using it.

Difficulties the team faced in the current and previous sprints, separate from the lack of knowledge in certain areas, was connected with the XP Practice – Collective Ownership. Due to the lack of understanding how the version control software used (GitHub) is working, the group

experienced a lot of problems such as: lost code, overwritten code, strange behavior etc. As a result of the mentioned difficulties, a lot of time was lost, but it didn't have negative effect on the overall completion of the tasks in time. As a "victims" of GitHub, the team wanted to suggest to the University, to introduce at least 1 hour of "lecture" how to use the application, as a result it may save a lot of time and concerns to future students.

Conclusion

The outcome of this working process carried over a period of more than five weeks, represents the software intended to answer the main question of the proposed problem statement "How can schools avoid the daily time and resource consuming process of delivering homework, by using computer technologies?" This platform has successfully been created, meeting the deadline, due to the right choice in what concerns the development methods. The constant changes in the requirements would have made it impossible for the development team to meet the deadline, if a wrong development method would have been chosen.

The architecture of the developed software answers the second question of the problem statement: "Could this be designed so it can be supported by different platforms and devices? ". By creating and implementing a WCF service for this online homework platform, the development team created also the possibility for further development of the platform that would allow the implementation of the homework platform on different other platforms or devices, for example on mobile platforms as Windows mobile, Android or IOS.

Additional functionality has been designed and implemented for the users to allow them to book a tutor for their educational needs, answering the last question of the problem statement: "What other functionalities could be added that would benefit the users in the scholar context?" .This functionality implements transactions and concurrency, making it a true distributed system. Beside this functionality, the users can also see a history of their submitted homework and the development team has developed also a windows application that separates the functionalities available for children and the ones available for teachers.

In conclusion, by choosing the Agile development method with different features of both Scrum and Extreme programming the development team made the right choice and managed to meet each of their deadlines and constantly deliver the business value demanded by the product owner, in the same time successfully answering all the questions of the formulated problem statement.

References

[agile manifesto] Software Engineering, 10th edition, Ian Sommerville, Page 76., http://agilemanifesto.org/

[Schwaber and Beedle 2001; Rubin 2013] Software Engineering, 10th edition, Ian Sommerville, Page 85.

[waterfall model] Software Engineering, 10th edition, Ian Sommerville, Page 47.

[Incremental development] Software Engineering, 10th edition, Ian Sommerville, Page 49.

[XP] Software Engineering, 10th edition, Ian Sommerville, Page 77; System development slides, eCampus, session 3

[Kanban slides] system development slides, eCampus, session 4

[MSDN Microsoft] https://msdn.microsoft.com/en-us/library/ee658094.aspx, 01.12.2015

[Architectural design decisions] Software Engineering, 10th edition, Ian Sommerville, Page 171.

[Architectural patterns] Software Engineering, 10th edition, Ian Sommerville, Page 175.

[Architecture views] Software Engineering, 10th edition, Ian Sommerville, Page 173.

[Selecting methods] system development slides, eCampus, session 4

[SE9thEdition]

[Ken Schwaber and Jeff Sutherland] The Definitive Guide to Scrum: The Rules of the Game, 7