UNIVERSITY OF TARTU

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Institute of Computer Science

Information Technology

Kadri-Liis Kusmin

Application of Scrum methodology in long-term student projects on the example of ESTCube-1 Mission Control System

Bachelor Thesis (6 EAP)

Supervisors: Silver Lätt, Meelis Kull

Author: ……..… "….." May 2012

Supervisor: ……..…. "….." May 2012

Supervisor: ……..…. "….." May 2012

Approved for defense:

Professor ..... ……. "….." May 2012

TARTU 2012

Table of Contents

1. Introduction

The aim of this thesis is to analyze the use of Scrum-methodology in long-term student projects, on the example of the development of ESTCube-1 Mission Control System (further on MCS). As there is no unified definition for the term “long-term student project”, the concept is defined based on the characteristics of MCS.

Scrum is a software development methodology for managing software projects as well as for product or application development. Currently all ESTCube-1 MCS teams use a customized implementation of Scrum for software development.

As the thesis is written while the development process of MCS is still in progress, the analysis reveals whether Scrum is a reliable development methodology for the projects of given type, or the teams should consider choosing another approach. The reader is also given an overview of modifications that have been made in the implementation of Scrum during the process so far. If the analysis verifies that Scrum is a suitable methodology for this project, additional recommendations for further adjustments are given. However, if it turns out that Scrum counteracts the development rather than promotes it, a detailed recommendation is written that can be used as a reference when choosing a new methodology. To ensure the reliability of the analysis, three other agile methodologies are compared with the Scrum implementation of ESTCube-1 MCS.

The objectives can be summarized with the following:

1. Analyze the use of Scrum in ESTCube-1 MCS and find weaknesses in its application.
2. Ascertain the optimal methodology for MCS and similar projects.
3. Give recommendations.
    a. If the analysis indicates that Scrum is the optimal methodology for MCS, write additional recommendations for further adjustments in Scrum application.
    b. If the analysis shows that Scrum should be replaced, suggest another methodology.

The first chapter of the thesis, “ESTCube-1 – long term student project” gives an overview of the ESTCube-1 project and its sub-project ESTCube-1 Mission Control System. Based on the peculiarities of the project, a central concept of the thesis – “long-term student project” – is defined to explain why such projects need a different approach regarding software development. The chapter also covers basic information about agile methodologies and investigates whether they comply with the nature of long-term student projects.

The second chapter “Scrum” concentrates on Scrum: it explains the roles, artifacts and different meetings characteristic to Scrum.

The third chapter “Scrum-methodology in ESTCube-1 MCS – analysis” analyzes how Scrum practices are employed in the project. It also highlights problems and difficulties as well as special modifications in Scrum implementation.

To provide a comparison, chapter four “Other methodologies” introduces three other software development methodologies and gives a basic overview of their main practices.

Chapter five “Comparison of methodologies in the context of ESTCube-1 MCS” is dedicated to examining whether using Scrum in given situations would be the best choice or the project would benefit more from some other methodology.

The results of the analysis are provided in the sixth chapter “Summary”. If the analysis confirms that Scrum is not the optimal methodology for projects of given type, another methodology is suggested. Otherwise, recommendations are given to make the best out of Scrum and its modifications.

2. ESTCube-1 Mission Control System – long-term student project

2.1. ESTCube-1 Mission Control System

ESTCube-1, also known as Estonian Student Satellite, is a collaboration project of Estonian students and their instructors. Its main objective is to promote studies and knowledge in space technology. The Mission Control System is the sub-project of the given project and concentrates on the development of a universal system for the ground stations, capable of sending commands and queries to satellites.

Project ESTCube-1 is particularly unique because the majority of the developers are students. That means the development process takes place alongside other courses and schoolwork, making it impossible to use the organizational plans of typical commercial software projects. The present thesis is, therefore, beneficial for everyone planning to start or already working in a similar setting and who is deciding which software development methodology to implement.

The development of MCS started in September 2011 and the launch of the first prototype that meets the primary requirements is planned for the end of 2012 or the beginning of 2013. Consequently, the duration of the whole MCS project is approximately four semesters, one of which falls on the summer period.

Given the circumstances, the MCS development team changes with each semester. This means that at least four different teams would work on the project, with some team members contributing for longer than one semester.

The teams are mainly formed in cooperation with the professors of the University of Tartu. Negotiations are held with professors and students of Information Technology, Informatics or other software development related courses. The students are then offered an opportunity to advance their skills in software development for space technology in the framework of MCS.

By the time of completing the thesis, three teams have taken part in MCS development. In autumn 2011 the team consisted of five Information Technology students of 2nd or 3rd

year. In spring 2012, the development was taken over by two teams: six 1st year and five 2nd year Information Technology or Informatics students.

The above said characterizes clearly one of the peculiarities of the ESTCube-1 project – the levels and skill sets of the development teams are rather different and after the previous team has finished, a less experienced team can take over and continue with the project.

The term "long-term student project" (in short "student project") in this context can be defined by the following:

1. The vast majority of workers in the project are students.
2. The project lasts longer than one semester.
3. The project concentrates on software development.
4. The objectives of the project are mainly educational but may also provide a practical output.

2.2. Agile methodologies

Agile methodologies, also called lightweight methodologies, evolved in the mid-1990s, in contrast to heavyweight methods (Larman & Basili, 2003). The difference between the two lies in some key principles: heavyweight methodologies rely on predictable, heavily regulated and regimented software development, whereas lightweight methods promote adaptive planning, rapid and flexible response to change with iterative and incremental development.

Lightweight methodologies include Scrum, Extreme Programming, Lean Software Development, Adaptive Software Development, Feature Driven Development, Crystal Clear and many more.

The term agile was introduced in 2001 when 17 software developers met at Snowbird, Utah resort, to discuss lightweight development methods (Highsmith, 2001). This lead to the publication of a document describing a new approach to software development: the Manifesto for Agile Software Development (Beck, et al., Manifesto for Agile Software Development, 2001), also known as the agile manifesto. The agile manifesto reads:

"We are uncovering better ways of developing

software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on

the right, we value the items on the left more."

The manifesto is based on twelve principles (Beck, et al., Principles behind the Agile Manifesto, 2001) published at the same time with the manifesto. The main ideas of the principles are as follows:

1. Early and continuous delivery of working software that satisfies the customer.
2. Requirements may change during the development process, even late in development.
3. Frequent delivery of working software, with a preference to a shorter timescale.
4. Business people and developers work all together daily, throughout the project.
5. Projects are developed by motivated individuals who need to be encouraged and trusted.
6. Promoting face-to-face conversation.
7. Working software is the primary measure of progress.
8. Maintaining constant pace in development.
9. Continuous attention to technical excellence and good design.
10. Simplicity is essential.
11. Teams are self-organizing.
12. At regular intervals, the team discusses how to become more effective, and makes the necessary changes.

2.3. Agile and long-term student projects

On the example of the above said, it is possible to analyze whether the use of agile methodologies in student projects is reasonable.

To examine which of the previously mentioned agile principles overlap with the needs of long-term student projects, each principle is analyzed to validate its importance in the context of long-term student projects. Based on a discussion carried out among the MCS team members of autumn 2011, the principles are given one of the two evaluations: *important* or *expendable* for the given type of projects. To ease the analysis, the author of the thesis has categorized the principles between the four qualities mentioned in the manifesto.

1. We value individuals and interactions over processes and tools.

    a. Projects are developed by motivated individuals who need to be encouraged and trusted.

        As the students working on this project do not gain any material benefit, it is essential to keep them motivated via other means. This prevents the development process from becoming tedious schoolwork - working on the project becomes a hobby instead. The latter advances the learning process and ensures the students' continuous contribution to the project.

        This principle is *important*.

    b. Promoting face-to-face conversation.

        The development process might benefit from face-to-face conversation but it is not always possible to employ such communication in student projects. In that case, the meetings can be carried out using on-line communication technologies, *e.g.* Skype. However, face-to-face meetings should still be preferred.

        This principle is *expendable*.

    c. Teams are self-organizing.

        Self-organizing teams are key figures in software development companies that use agile methodologies. However, to assure that the students achieve

their educational goals, there should be at least one supervisor per team that helps to organize and distribute tasks, provides technical assistance and makes sure the development follows the schedule.

This principle is *expendable*.

d. Maintaining constant pace in development.

It is very important to keep constant pace throughout the whole project. Falling behind schedule or working too hard to get ahead of it can both result in failure of the project. Studies have shown that people who consistently work very long work weeks get burned out, making their team less effective too (Robinson, 2012).

This principle is *important*.

e. At regular intervals, the team discusses how to become more effective, and makes the necessary changes.

This principle helps to benefit in both developmental and learning processes. Giving and receiving feedback helps the students objectively evaluate their own and the whole team's progress and teaches how to increase their efficiency.

This principle is *important*.

2. We value working software over comprehensive documentation.

a. Early and continuous delivery of working software that satisfies the customer.

While many software development methodologies do not concentrate on early and continuous working software delivery, it is considered very important in agile development. This is also a good practice in teaching – the students submit their work after each interval and thus get feedback continuously throughout the project. Besides ensuring that the software satisfies the needs of the customer, it also helps the students to split the workload into equal portions, avoiding unnecessary stress at the end of the semester.

This principle is *important*.

b. Frequent delivery of working software, with a preference to a shorter timescale.

   Similarly to the previous principle, frequent delivery of software helps to keep the students on track. When using shorter timescales, the development portions are smaller, making it easier to keep track of functionalities that are already implemented and those that are not.

   This principle is *important*.

c. Working software is the primary measure of progress.

   Although working software delivery is important, in student projects the primary measure of progress is the educational aspect. This can be measured through different approaches, one of which is the students' own evaluation of their learning progress.

   This principle is *expendable*.

d. Continuous attention to technical excellence and good design.

   The aim of student projects is to teach the students to excel in the corresponding field.

   This principle is *important*.

e. Simplicity is essential.

   Simplicity is vital in software development and should be promoted in early stages of learning. "Simplicity is prerequisite for reliability." - Edsger W.Dijkstra, 1972. Turing Award winner.

   This principle is *important*.

3. We value customer collaboration over contract negotiation.

   a. Business people and developers work all together daily, throughout the project.

      Many student projects are organized by a company that is not related to

any university. In other student projects, the university supervisors act as business people involved in the software development. It is essential to demonstrate the importance of understanding the customers' needs correctly and building the software in compliance with the requirements.

This principle is *important*.

4. We value responding to change over following a plan.

    a. Requirements may change during the development process, even late in development.

    Even the strictest projects may run into difficulties and need reconsideration of requirements, so it cannot be ruled out in student projects either.

    This principle is *important*.

The above considerations show that the majority of agile principles are also important for student projects, therefore, the analysis confirms that using agile methodology is justified. Note that all principles classified as expendable might not play a crucial role but should, nevertheless, be considered in student projects.

Scrum

In rugby, a scrum refers to the manner of restarting the game after a minor infraction. In rugby, plays often become chaotic and frenzied. Predefined steps in the play will be abandoned, because they do not work in the random nature of the sport. Instead, the rugby coach must teach his players how to be spontaneous, self-organizing, and able to act on the fly. The Scrum approach to IT project management works in a similar manner to the way a rugby coach organizes his plays (Grimme, 2010). Scrum methodology works more as a framework than a real methodology: rather than providing detailed descriptions of how everything should be done, most is up to the team itself to decide (Cohn, 2012). The reason behind it is that in Scrum, the software teams are self-organizing.

2.4. Scrum roles

There are three core roles in Scrum teams: Scrum Master, Product Owner and Development Team.

Scrum Master

The Scrum Master (sometimes written as "ScrumMaster") is the person responsible for the optimal use of Scrum. Other tasks include enforcing rules and removing obstacles and distractions, so the development team could focus only on important tasks. In other words, Scrum Master ensures that the team is fully functional, motivated and productive.

Product Owner

The Product Owner is the representative of the customer, and is thus responsible for the project's success. The Product Owner's tasks include outlining the objectives in the Product Backlog (a prioritized features list, containing short descriptions of all functionalities desired in the product) and prioritizing them based on their business value. As it is forbidden to give the development team new tasks during Sprint (a Scrum time unit), it is of utmost importance that the Product Owner correctly determines the most important features of the product, the deadlines when they have to be functional, and other relevant information.

Development Team

The Scrum Development Team usually consists of seven members, plus or minus two, and is cross-functional. This means that the team does not include traditional software

engineering roles such as architect, programmer, tester, etc. Instead, they are all equally responsible for designing the software, writing the code, testing it and also producing the required documentation.

2.5. Sprint

As mentioned earlier, Sprint is the basic time unit of software development in Scrum. A Sprint lasts from one week to one month and can be viewed as a separate development process – each Sprint has its own deliverables, deadline, and sometimes even budget. Every Sprint starts with a planning meeting, where items for the new Sprint Backlog are picked by Product Owner. During each Sprint, a part of the product is completed. After a Sprint is completed, the team presents the new software features.

2.6. Scrum artifacts

In Scrum, there are three important artifacts, around which the whole process is concentrated.

Product Backlog

The Product Backlog is the prioritized list of desired project outcomes and features. Although it can be opened and edited by anyone involved in the project, Product Owner is ultimately responsible for keeping the Product Backlog relevant.

Sprint Backlog

The Sprint Backlog is a list of tasks, selected from the Product Backlog, which the team agrees to complete in a Sprint. The list is derived by selecting stories or features from the top of the Product Backlog until the Development Team feels it has enough work to fill the Sprint. The Sprint Backlog is the property of each Development Team.

Burndown

The Sprint Burndown chart is a diagram which shows the remaining work in the Sprint Backlog. It is displayed publicly to provide visualization of the tasks that have been completed or still need to be worked on. There can also be other types of Burndown charts, i.e. for the project in general.

2.7. Scrum meetings

There are four fundamental meetings in Scrum, each with its own objective.

Sprint Planning meeting

The Sprint Planning meeting is held at the beginning of each Sprint. The purpose of this meeting is to select new items from the Product Backlog and add them to Sprint Backlog – these are the tasks that need to be completed by the end of a given Sprint.

Daily Scrum

The Daily Scrum is basically a project status meeting that takes place every day. This is the moment when the team members can discuss obstacles in their work as well as progress. The meeting lasts for 15 minutes and each team member answers three questions:

1. What did you do since the last meeting?
2. Do you have any impediments or obstacles?
3. What will you be doing until the next meeting?

Although only members of the Development Team are allowed to speak, organizing the Daily Scrum is the responsibility of the Scrum Master, who is also responsible for finding ways to overcome obstacles, should there be any.

Sprint Review meeting

During the Sprint Review meeting the team demonstrates to the Product Owner the work it has completed during the Sprint.

Sprint Retrospective meeting

The Sprint Retrospective meeting is an opportunity for the team to discuss different ways to improve the product and the process. All team members can reflect on the completed Sprint and two questions are answered:

- What went well during this Sprint?
- What can be improved in the next Sprint?

3. Scrum-methodology in ESTCube-1 MCS – analysis

The following chapter describes the use and customization of Scrum-methodology in ESTCube-1 MCS and is based on the earlier analysis of the compatibility of agile methodologies with student projects. It describes how the use of Scrum in MCS complies with the corresponding agile principles and uncovers practices that could be improved. In view of simplicity, the analysis concentrates only on the principles earlier marked as important. To complete the overview, an additional section analyzes common agile support tools currently used in MCS development.

Projects are developed by motivated individuals who need to be encouraged and trusted

In ESTCube-1 MCS the person responsible for team motivation is the Scrum Master. Both supervisors and students can fill the role of a Scrum Master. Everyone involved in the project has equal rights but at the same time equal responsibility for the success of the project.

The motivation level of MCS teams depends on many factors and varies from group to group. Based on the groups that have so far participated in the development of MCS, the most important factors are (see charts and table below):

- Scrum Master's occupation (supervisor/student)
- Scrum Master's experience
- The group's average university education in years
- The group's earlier knowledge of Scrum or agile
- The group's earlier experience in software development
- External factors (other courses, personal life)

As the MCS teams consist of volunteers interested in space- or information technology, the team members are usually highly motivated. However, there have been cases where a team loses its motivation and the quality and/or quantity of work deteriorates. An example can be given from the second semester of development. The following "Created vs Resolved" charts describe the work of two MCS teams that joined the project in Spring 2012. The x-axis describes the dates and the y-axis describes the number of issues: red –

unresolved, green – resolved issues. The charts are exported from ESTCube-1 MCS issue tracking environment JIRA, provided by Atlassian, which will be introduced later on in this chapter.

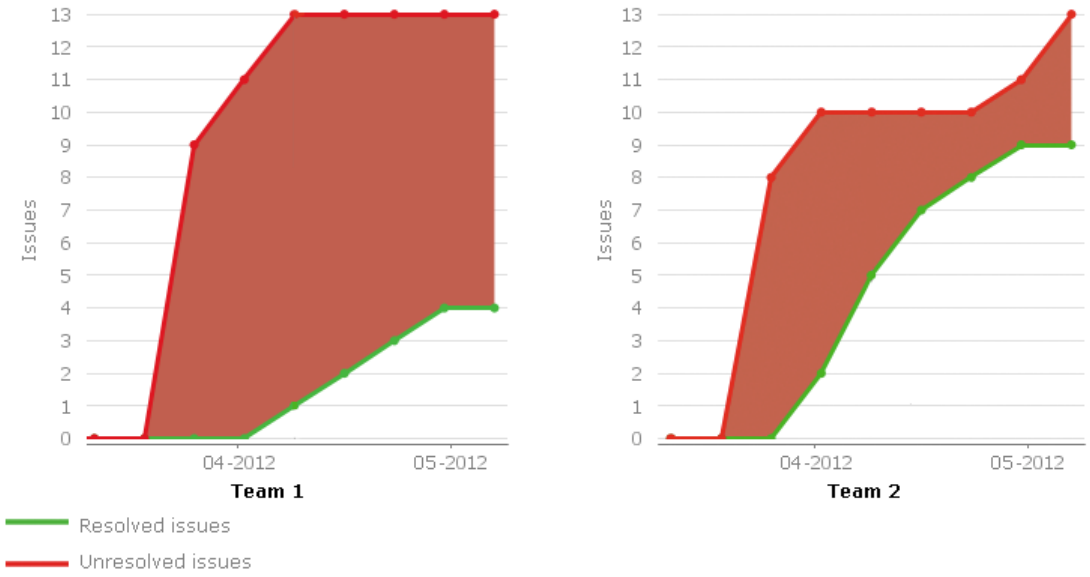

Figure 1. Motivation levels of two MCS teams, based on created and resolved issues ratio. The red line symbolizes the number of unresolved issues while the green one symbolizes the number of issues already resolved.

Although the teams worked on different objectives, the work loads of the groups were equivalent and the tasks rather similar. Both teams' tasks during the period shown in the graphs can be divided into three stages:

1. Familiarization with the project.

   This stage covered learning the basic information about the scope, structure and goals of the project. The teams also had to get acquainted with the development tools used in the process and get an overview of the previous team's work.

2. Interviewing the client

   The second stage started with establishing the goals, after which both teams carried out interviews with the clients – the supervisors – to create requirements documents.

3. Development stage

   Although the name suggests otherwise, this phase also includes periodically reviewing requirements and – if needed – replanning work. The teams' tasks in this phase were different in nature but of equal complexity.

It can be seen from figures 1 and 2 that although the teams started approximately at the same time, Team 1 fell behind with work after two weeks and could not establish persistent work schedule after that. For better understanding it must be said that the teams started at the beginning of the semester when other courses are less intense. This means that the problem of Team 1 lies in team motivation rather than lack of time. To find the reason behind this, both teams' motivation factors were analyzed based on the interviews with the team members of Team 1 and the Scrum Master of Team 2.

Table 1. Comparison of motivation factors of Teams 1 and 2.

Factor	Team 1	Team 2
Scrum Master's occupation	Student	Student
Scrum Master's experience	First time being Scrum Master, one previous Scrum experience	First time being Scrum Master, several previous Scrum experiences
Team's average year in university	1.	2.
Team's earlier knowledge of Scrum or agile (none, medium, high)	None	Medium
Team's earlier experience in software development (low, medium, high)	Low	Medium

Some external factors such as other courses and personal life can also be important in the context of team motivation, but they are hard to measure and are thus left out of the table.

It appears that in Team 1 both the Scrum Master as well as other members are less experienced than those in Team 2. The problem causing the lack of motivation could be that the Scrum Master has not been successful in establishing Sprints. To solve this

problem, the Scrum Master should enforce very strict Sprints for a while so that the team falls into a routine in its work. If the Scrum Master fails to do so, another problem might lie in the inexperience of the Scrum Master. In this case, the possible solution could be replacing the Scrum Master of Team 1 with someone more experienced, i.e. a supervisor.

Maintaining constant pace in development

The development of MCS is divided into Sprints that usually last for a week. This helps to maintain a constant pace and provides a development rhythm to the team. Sprints are made up of relatively few tasks so it also relieves the team from stress – instead of looking at a high workload that has to be completed in a long period, the team members can set short goals.

As mentioned earlier, it is essential to clarify the importance of Sprints to teams that are less experienced. There has not been a case where an MCS team that followed a Sprint precisely exceeded a time limit, so it can be said that using Sprints is effective and should be continued.

The critical points regarding the pace of MCS development are the transition stages from one team to another. It takes time for the new participants to get acquainted with the project in general as well as the work of the previous team and it may result in chaos. The teams follow agile approaches and keep the documentation and code as simple as possible to make them easier to understand for next teams.

At regular intervals, the team discusses how to become more effective, and makes the necessary changes

After each Sprint, the MCS teams discuss the completed Sprint and determine what could be changed in the work process that might make the next Sprint more enjoyable and productive. Besides making the work of individuals more effective, it also helps the team members to gain trust by giving and receiving respectful but objective feedback to or from each other.

Early and continuous delivery of working software that satisfies the customer

The requirements of the project are in frequent change, so the planning stages of MCS are rather short and the teams start working on technical tasks a lot faster than they would in other conditions.

However, recent feedback from teams has shown that the students would prefer the initial planning stage to be longer and more thorough. Currently, many team members find it hard to understand the scope and purpose of some of their tasks and this can eventually result in the decrease of their efficiency.

Frequent delivery of working software, with a preference to a shorter timescale

At least once a week the MCS teams attend a technical meeting where a supervisor analyzes work done, gives advice and helps to solve problems that might later on become obstacles. These meetings are not inherent to any agile methodology, but are necessary as the team members are not yet experienced enough to supervise each other's work by themselves.

Continuous attention to technical excellence and good design

The two most important practices regarding good technical quality in software development are code reviews, testing and continuous integration. The MCS teams use Atlassian Bamboo continuous integration environment to automate building, testing deploying and releasing software.

There have been meetings with the sole purpose of reviewing code, but as the teams' work is analyzed every week anyway, there is no direct need to carry it out separately.

Simplicity is essential

Simplicity is the main theme in agile methodologies, but unfortunately there is no direct practice for it in Scrum. However, the MCS architecture is developed keeping in mind that the code and documentation produced have to be easily understandable to the next team that takes over the project after the current team has finished.

Business people and developers work all together daily, throughout the project

ESTCube-1 is a non-profit project initially started by university professors interested in space technology. This means that the supervisors themselves have also taken the role of the customer. Consequently, it can be said that the supervisors are all Product Managers, because they attend all meetings and ensure that the software meets the requirements.

Requirements may change during the development process, even late in development

While it is forbidden in Scrum to change a Sprint's objectives after its launch, the MCS teams adapt to changes as soon as possible, even if it means making changes to the ongoing Sprint itself. Since the MCS teams have always taken into account the possibility of reconsiderations in requirements, all significant changes so far have gone smoothly.

Agile support tools

To optimize their work, MCS teams use a variety of agile support tools provided by Atlassian (http://www.atlassian.com).

1. Atlassian JIRA – project and issue tracking environment
2. Atlassian Confluence – enterprise wiki system. Although agile approaches see less value in documentation than other methods, MCS teams find that all relevant documents should be stored and kept available. Confluence is used to store team members' contact information, Product Backlog, User Manual draft, as well as different demos from one team to another etc.
3. Atlassian Bamboo – continuous integration and release management tool. Bamboo provides automated building, testing, deploying and releasing of software. The Bamboo Clover plug-in helps to keep track of test automation.
4. Atlassian FishEye – unified view of source code repository.

4. Other methodologies

In order to analyze whether Scrum is the optimal methodology for ESTCube-1 MCS, its possibilities should also be compared with the ones provided by other methodologies. It is already established that agile approaches comply with the nature of the project, so the methodologies to be compared are selected amongst them. The selection is based on the spread of use and topicality:

- Extreme Programming (XP)
- Crystal Clear (CC)
- Lean Software Development (LSD)

All of them follow agile fundamentals, but the main differences lie in the ways they are put into practice.

4.1. Extreme Programming

The main practices of Extreme Programming, according to "The Rules of Extreme Programming" (Wells, 2009):

1. Pair programming – All code is produced by two people programming simultaneously using one computer.
2. Planning game – The main planning process within XP, a meeting that occurs once per iteration.
3. Test driven development – Unit tests are written before the eventual code is coded.
4. Whole team – Everyone who contributes to an XP project is considered a team member: developers, customer, testers etc.
5. Continuous integration – All parts of the team have the latest possible version of the code.
6. Design improvement – The code should be refactored frequently.
7. Short releases – Software is delivered in small but working portions.
8. Coding standards – The team agrees to follow one coding standard throughout the project.
9. Collective code ownership – Everyone is responsible for the code.
10. Simple design – All functionalities should be implemented as simply as possible.

11. System metaphor – As the team includes people who are not familiar with programing languages, the system is explained in metaphors so everyone can understand how it works.
12. Sustainable pace – Developers should not work over 40 hours per week.

4.2. Crystal Clear

The main practices of Crystal Clear, according to "Crystal Clear: A Human-Powered Methodology for Small Teams" (Cockburn, 2004):

1. Frequent delivery of usable code to users
2. Reflective improvement
3. Osmotic communication preferably being co-located
4. Personal safety – All team members are able to speak when something is bothering them, without fear of reprisal.
5. Focus – It is important for each member to fully understand what and why they are working on.
6. Easy access to expert users
7. Automated tests, configuration management, and frequent integration

4.3. Lean Software Development

The main practices of Lean Software Development, according to "Lean Software Development: An Agile Toolkit" (Poppendieck & Poppendieck, 2003):

1. Eliminate waste – Everything that is not a direct necessity, is considere waste.
2. Amplify learning – Software development should be a continuous learning process.
3. Decide as late as possible – An options-based approach where descisions are delayed as much as possible until they can be made based on facts and not on uncertain predictions or assumptions.
4. Deliver as fast as possible
5. Empower the team – The development team has the same rights as the managers.
6. Build integrity in – The system being build has to be integral.
7. See the whole – All members of the team have to understand the so-called "big picture" of the project and Lean itself.

5. Comparison of methodologies in context of ESTCube-1 MCS

The objective of comparing Scrum and other agile methodologies is to find out whether using Scrum in given circumstances is the most optimal approach or the project would benefit significantly were it to switch to some other methodology.

The comparison is based on the method proposed by Alan Koch (Koch, 2005). The practices of chosen methodologies are compared empirically in the context of the given enterprise, having categorized them on the basis of agile principles. Each comparison includes points given to each methodology: 1 – the practices are relevant for long-term student projects, 0 – the practices are irrelevant for long-term student projects. The scoring is also explained.

Projects are developed by motivated individuals who need to be encouraged and trusted

It can be clearly seen from the example given in the analysis that the greatest threat for the project lies in team motivation (see figures 1 and 2). However, as Scrums practices have been reliable in team motivation of other MCS teams, the issue might be solved with the options provided earlier in the analysis: enforcement of strict Sprints of replacement of the Scrum Master.

- Scrum: ”Scrum Master” - 1

The corresponding practices in other methodologies:

- XP: “Whole team” and “collective code ownership” - 0
- LSD: ”Empower the team” - 0
- CC: ”Personal safety” - 0

Although the given practices are somewhat related to the problem in hand, it cannot be said that any of them could be seen as a direct solution or an improvement for the project in general.

Maintaining constant pace in development

The development of MCS has not had any problems with this principle. It can be said that Scrum Sprints are a feasible practice for managing steady pace:

- Scrum: ”Sprint” - 1

It appears that without knowing, the MCS teams have implemented the corresponding practice of XP: none of the teams work more than 40 hours a week on the project. However, it cannot be said that the practice is implemented integrally – this estimate does not take into account the tasks related to other courses and/or work:

- XP: ”Sustainable pace” – 0

At regular intervals, the team discusses how to become more effective, and makes the necessary changes

The practices are very similar in all methodologies that follow this kind of approach:

- Scrum: ”Sprint Retrospective” - 1
- CC: ”Reflective improvement” - 1

It can be said that Sprint Retrospective fulfills the requirements of this principle.

Early and continuous delivery of working software that satisfies the customer

All of the four methodologies aim for this principle:

- Scrum: ”Sprint” - 1
- XP: ”Small releases” - 1
- CC: ”Frequent delivery” - 1
- LSD: ”Deliver as fast as possible” - 1

There have been no issues with using Scrum Sprint, therefore there is no need to reconsider the practice for this principle.

Frequent delivery of working software, with a preference to a shorter timescale

This principle overlaps with the previous one in terms of practices, so the same solution can be applied to both of them.

Continuous attention to technical excellence and good design

In MCS development, the Scrum Master is responsible for the team’s well-being so the latter could focus entirely on achieving technical excellence. The MCS teams use Atlassian Bamboo continuous integration environment to automate building, testing, deploying and releasing software. To make the team’s work even more effective as well

as beneficial in educational purposes, the teams should consider implementing another one of XP practices: pair programming. This helps to level the team members' competence and also produce reliable results in shorter time. LSD version of this practice is relevant, but coincides with the corresponding aim of agile in general.

- Scrum: ”Scrum Master” - 1
- XP: ”Pair programming”, “Test driven development”, “Coding standards”, “Continuous integration” - 1
- LSD: ”Build integrity in” - 1

Simplicity is essential

Although simplicity is one of the centralized ideas of agile, only two of the chosen methodologies have corresponding practices:

- XP: ”Simple design” -1
- LSD: ”Eliminate waste” - 0

The MCS teams keep the design at its simplest to make it easier for next teams to familiarize with it, therefore it can be said that the XP “Simple design” practice is already implemented. Using corresponding LSD practice is, however, disadvantageous as it is necessary to preserve all documentation for the transition period between teams.

Business people and developers work all together daily, throughout the project

All given methodologies support the idea of continuous cooperation between all partners:

- Scrum: ”Product Manager”, ”Product Backlog” -1
- XP: ”Whole team” -1
- CC: ”Easy access to expert users” -1
- LSD: ”See the whole” -1

With some differences, all of those practices see the customer as a part of the team. In MCS the supervisors also fill the functions of Product Managers so it can be said that the client is integrated in the team.

Requirements may change during the development process, even late in development

The MCS teams adapt to changes as soon as possible with some violation of Scrum rules: although changing the ongoing Sprint is forbidden, it is still practiced in critical situations.

Only one other methodology has a direct practice for fulfilling this principle which matches the current MCS implementation of this principle:

- LSD: ”Decide as late as possible” – 1

The implementation if this practice might help to avoid violating Scrum’s strict rules related to Sprints.

Summary of comparison

The results can be compiled in a table:

Table 2. Scores of given methodologies based on the relevance of their practices in the context of MCS.

Principle (corresponding chapter)	Scrum	XP	CC	LSD
Team motivation (6.1)	1	0	0	0
Constant pace (6.2)	1	0	0	0
Increase of efficiency (6.3)	1	0	1	0
Early and frequent delivery (6.4, 6.5)	1	1	1	1
Technical excellence (6.6)	1	1	0	1
Simplicity (6.7)	0	1	0	0
Cooperation of all participants (6.8)	1	1	1	1
Late changes in requirements (6.9)	0	0	0	1
Sum	6	4	3	4

When summarizing the scores in Table 2, it can be seen that the optimal methodology for ESTCube-1 MCS and analogous student projects is Scrum, followed by LSD and XP with only a two-point difference. CC fulfills some of the principles but all in all is not reliable in the project’s context.

However, as found in the analysis, the most beneficial strategy would be to supplement Scrum with some practices from other methodologies:

- “Pair programming” (XP)
- “Decide as late as possible” (LSD)

6. Summary

The purpose of this thesis was to analyze the use of Scrum-methodology in long-term student projects following the case of ESTCube-1 Mission Control System, and to examine whether it is a reliable methodology in the projects of given type. The analysis revealed that the implementation of Scrum in ESTCube-1 is a suitable methodology for long-term student projects. The only weakness in the current methodology implementation is vulnerability to possible lack of team motivation. However, the potential problem can be avoided with establishing a work routine that is strict enough for less experienced teams or choosing another, more experienced, Scrum Master.

Another objective of the thesis was to give additional recommendations for further adjustments for Scrum if the reliability of Scrum was verified. To increase work efficiency using best practices from other methodologies alongside with Scrum should be considered.

To make the teams' work more effective as well as beneficial in educational purposes, the teams should consider implementing "Pair programming" from Extreme Programming. As this technique requires that all code is produced by two people programming simultaneously on one computer, this practice would help to raise the competence of less experienced team members and produce reliable results in shorter time.

To reduce the number of cases where ongoing Sprints have to be changed, the teams should implement the "Decide as late as possible" practice from Lean Software Development. While the Sprint Backlog is currently composed of the most important items already specified in the Product Backlog, the implementation of the practice in question would reduce the time spent on tasks that later would have to be reconsidered.

Scrum-metoodika kasutamine pikaajalistes tudengiprojektides ESTCube-1 Missioonijuhtimissüsteemi näitel

Bakalaureusetöö eesmärk oli analüüsida Scrum-metoodika kasutamist pikaajalistes tudengiprojektides, kasutades näitena ESTCube-1 Missioonijuhtimissüsteemi (edaspidi MJS). Analüüs tõi välja senise Scrum-implementatsiooni nõrgemad küljed ja nende võimalikud lahendused. Et töö kirjutati ajal, mil MJS arendusprotsess oli alles pooleli, analüüsiti ka, kas antud metoodikaga tasub jätkata või tuleb valida mõni teine. Analüüsi usaldusväärsuse tõstmiseks on sisse toodud kolm teist arendusmetoodikat ja neid MJS Scrum-implementatsiooniga võrreldud.

Scrum on agiilne tarkvara arendamise metoodika, mis põhineb iteratiivsetel ja inkrementaalsetel praktikatel. Iteratiivne lähenemine näeb ette, et esmalt ehitatakse valmis väike osa tootest, mida edaspidi inkrementaalselt ehk järk-järgult järgmistes etappides laiendatakse. Scrum'i üheks olulisemaks tunnuseks ongi etappidepõhine tsükliline arendamisprotsess. Samuti iseloomustavad Scrum'i iseorganiseeruvad meeskonnad ja tugev rõhk tehnilisel täiuslikkusel. Bakalaureusetöö kirjutamise hetkeks on Scrum arendusmetoodikat kasutanud kolm MJS-meeskonda.

Analüüsist selgus, et hetkel MJS arenduses kasutusel olev Scrum-implementatsioon vastab hästi antud projekti kontekstile. Ainsa nõrkusena võib välja tuua haavatavuse juhul, kui meeskond motivatsiooni kaotab. Võimalikku probleemi saab aga vältida rangete arendustsüklite kehtestamisega või kogenematu Scrum Master'i (Scrum'i meeskonnajuhi) väljavahetamisega.

Eelnev oli ainus Scrum'i kasutamisega seotud potsentsiaalne oht, seega võib öelda, et hetkel kasutusel olev Scrum-implementatsioon tuleb projektile kindlasti kasuks. Sellegipoolest võiks töö optimeerimise eesmärkidel kaaluda ka kasulikemate praktikate laenamist teistest metoodikatest.

MJS meeskondade liikmete tase on tihti väga erinev ja seega tasuks meeskonnasiseseks ühtlustamiseks rakendada ekstreemprogrammeerimise (*Extreme Programming*) paarisprogrammeerimist. Paarisprogrammeerimine on kahe inimese üheaegne programmeerimine ühe arvuti taga. See aitaks vähesema eelneva kogemusega meeskonnaliikmetel teistele järele jõuda ja samal ajal ka projekti arendamisesse panustada.

Et vähendada juhtusid, mil käimasolevaid Sprinte (Scrum'i arendustsükkel) on vaja muuta, võiksid meeskonnad implementeerida *Lean Software Development* metoodika praktikat "Otsusta nii hilja kui võimalik". Hetkel alustatakse tööd kõrgema prioriteediga ülesannetest, olenemata sellest, kas need on lõplikult määratletud või mitte. Antud praktika aitaks vähendada olukordi, kus varasemalt määratletud eesmärki muudetakse ja sellega seotud töö on vaja kas osaliselt või täielikult uuesti teha.

References

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Manifesto for Agile Software Development*. Retrieved 05 2012, from Manifesto for Agile Software Development: http://agilemanifesto.org

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Principles behind the Agile Manifesto*. Retrieved 05 2012, from Manifesto for Agile Software Development: http://agilemanifesto.org/principles.html

Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams.*

Cohn, M. (2012). *Introduction to Scrum*. Retrieved 2012, from Mountain Goat Software: http://www.mountaingoatsoftware.com/topics/scrum

Grimme, A. (2010, 02). *Rugby vs. Development: What Is SCRUM?* Retrieved from Global Knowledge: http://www.globalknowledge.com/training/generic.asp?pageid=2167&country=United+States

Highsmith, J. (2001). *History: The Agile Manifesto*. Retrieved 05 2012, from Manifesto for Agile Software Development: http://agilemanifesto.org/history.html

Koch, A. (2005). *Agile Software Development: Evaluating The Methods For Your Organization.*

Larman, C., & Basili, V. R. (2003). *Iterative and Incremental Development: A Brief History.*

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit.*

Robinson, S. (2012, 03). Why We Have to Go Back to a 40-Hour Work Week to Keep Our Sanity. *AlterNet* .

Wells, D. (2009, 09). *The Rules of Extreme Programming*. Retrieved 05 2012, from Extreme Programming: A gentle introduction: http://www.extremeprogramming.org/rules.html