

University College of Northern Denmark

Technology and Business

Computer Science Academy Profession (AP) Degree

Class: DMAJ0916

Title: System Development – 3rd Semester 2017

Project participants (Group 2):

-Ralfs Zangis

-Andrei-Eugen Birta

-Hannes Heiskonen

-Stoycho Nenov Anastasov

Supervisors:

-Brian Hvarregaard

-Per Trosborg

Due date: 2017-December-18 (13:45 UTC+1)

Submission date: 2017-December-17

Birta Zanigis Heiskonen Nenov

Contents

[1. Preliminary Study 3](#_Toc501210729)

[1.1. Introduction 3](#_Toc501210730)

[a. Problem Statement 3](#_Toc501210731)

[1.2. Plan driven Vs. Agile Development (elaborated through methods) 3](#_Toc501210732)

[1.1.1. Plan Driven 4](#_Toc501210733)

[a. Waterfall 4](#_Toc501210734)

[b. Unified Process 4](#_Toc501210735)

[c. Pros of Plan Driven Development Methods 4](#_Toc501210736)

[d. Cons of Plan Driven Development Methods 4](#_Toc501210737)

[1.1.2. Agile 4](#_Toc501210738)

[a. Extreme Programming 4](#_Toc501210739)

[b. SCRUM 5](#_Toc501210740)

[c. Kanban 5](#_Toc501210741)

[d. Pros of Agile Development Methods 6](#_Toc501210742)

[e. Cons of Plan Driven Development Methods 6](#_Toc501210743)

[1.3. Quality Assurance 6](#_Toc501210744)

[2. Development Process 6](#_Toc501210745)

[2.1. Development Method of Choice 6](#_Toc501210746)

[a. Final Choice 7](#_Toc501210747)

[b. Pros 8](#_Toc501210748)

[c. Cons 8](#_Toc501210749)

[2.2. System Architecture and Quality Criteria 8](#_Toc501210750)

[a. System Architecture 8](#_Toc501210751)

[b. Quality Criteria 9](#_Toc501210752)

[2.3. Sprints Summary 10](#_Toc501210753)

[2.2.0. User stories 10](#_Toc501210754)

[2.2.1. Product Backlog 12](#_Toc501210755)

[2.2.2. Sprint 0 14](#_Toc501210756)

[a. Sprint summary and MoSCoW model 14](#_Toc501210757)

[b. Sprint Backlog and burndown chart 15](#_Toc501210758)

[c. Sprint retrospective 16](#_Toc501210759)

[2.2.3. Sprint 1 16](#_Toc501210760)

[a. Sprint summary 16](#_Toc501210761)

[b. Sprint Backlog and burndown chart 17](#_Toc501210762)

[c. Sprint retrospective 17](#_Toc501210763)

[2.2.4. Sprint 2 17](#_Toc501210764)

[a. Sprint summary 17](#_Toc501210765)

[b. Sprint Backlog and burndown chart 18](#_Toc501210766)

[c. Sprint retrospective 19](#_Toc501210767)

[2.2.5. Sprint 3 19](#_Toc501210768)

[a. Sprint summary 19](#_Toc501210769)

[b. Sprint Backlog and burndown chart 19](#_Toc501210770)

[c. Sprint retrospective 20](#_Toc501210771)

[2.2.6. Sprint 4 20](#_Toc501210772)

[a. Sprint summary 20](#_Toc501210773)

[b. Sprint Backlog and burndown chart 21](#_Toc501210774)

[c. Sprint retrospective 21](#_Toc501210775)

[3. Conclusion 22](#_Toc501210776)

[3.1. Denouement 22](#_Toc501210777)

# 1. Preliminary Study

## 1.1. Introduction

This document summarizes the collaboration of Group 2 for the system development exam of the 3rd Semester. The group consists of 4 members of 4 different nationalities. Despite the major differences in our opinions, we agreed on certain rules and guidelines to follow, thoroughly elaborated in the accompanying document called group contract; we managed to harness the benefits of diverse ideas and identify multiple possible approaches to certain problems.

### Problem Statement

The idea of this project is to create a web service that communicates with different types of clients, and allows users all over the world to gather round in topic-specialized chatrooms and facilitate passionate, real time discussions among small groups of people. On top of that, the service will provide several means of entertainment, in order to maintain a healthy community, such as: playing YouTube videos, creating playlists or playing a game of Rock-Paper-Scissors.

After formulating the problem statement and having it approved by the supervisors we were assigned the task of solving the problem by selecting the most suitable agile system development method based on the situation, through well-planned and well-synchronized teamwork.

## 1.2. Plan driven Vs. Agile Development (elaborated through methods)

When it comes to software development, the best way of working as a team, is to select one of the many frameworks that exist, that best fit both the team’s composition and the project’s needs. The following few frameworks are some of development methods we have taken into consideration.

### Plan Driven

#### Waterfall

Waterfall method follows very simple pattern. At first all the requirements are gathered. Then final product is designed, and the product development process may begin. After product is finished, it is tested. Finally, product is released and only jobs left to do are maintenance jobs.

#### Unified Process

Unified Process is an iterative and incremental, use-case driven and architecture centric software development method. At first, all the requirements are gathered. After gathering requirements, use-cases are created along with other diagrams like domain model, sequence diagrams, communication diagrams and design class diagram. Work is done in iterations, iterations have 4 phases- Inception, elaboration, construction and transition. Tests are usually done at the end of each iteration.

#### Pros of Plan Driven Development Methods

* Clear overview of project
* Documentation is easy to write
* Perfect for small or critical projects, or when requirements change little if at all
* Good for beginner programmers since most of the decisions are taken by the designers.
* Easy to manage and control

#### Cons of Plan Driven Development Methods

* Changes are hard to implement and can be extremely expensive
* Little communication between customer and development team
* Little time for fixing errors and can lead to delays

### 1.1.2. Agile

#### Extreme Programming

Extreme Programming(XP) is an agile planning and development method. It is based on 4 values (communication, simplicity, feedback and courage) and 12 principles (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).

XP methodology starts with planning game. User stories are created to describe functions in the program. Then they must go through acceptance test. Next up is estimations for user stories. Team estimates time it takes to finish a certain user story, block of user stories or even the whole project. XP implements many coding practices, a couple of which are pair programming and test-driven development (TDD).

Pair programming is a clever way to solve problems about complex parts of project. Not only because it quickens the idea generation for solutions in different problems, but also because it reduces the chance of imperfections appearing during the coding.

TDD is smart way to prevent code from stopping to do what it is supposed to do. For example, when implementing modifications to existing code, one or more functions might get lost or simply stop working. TDD helps keep track of all the functions in the project by forcing one to create test before writing actual method.

#### SCRUM

SCRUM is an agile planning and controlling method. It has 3 roles (product owner, scrum master and scrum team), 3 ceremonies (sprint planning, daily meeting and sprint retrospective) and 3 artifacts (product backlog, sprint backlog and burndown chart).

At first the scrum master and product owner set up product backlog. Items are then prioritized (by the product owner), and the entire team adds time estimations. Items are then moved from product backlog into sprint backlog, burndown chart can be created, and sprint can begin. Daily meeting is also part of scrum, they are conducted by Scrum master. Meetings are used to get an overview of the state in which the project currently is, and update burndown chart. People discuss what they achieved since last meeting, what do they plan on doing and if there are any problems. Since scrum is based on incremental development, divided in iterations, called Sprints, the product’s evolution can be easily seen by both the development team and the customer, since at the end of every sprint a working product should always be presented.

If any changes in the requirements occur, then new tasks can be added to product backlog at any time and from there they can be put into sprint backlog for development.

#### c. Kanban

Kanban is another agile development method which uses the “Kanban board” for dividing workflow. Kanban has 3 rules: visualize workflow, put limit to number of items in work in progress area, and estimate time it would take to finish the task.

Since there are fewer rules than the other previously mentioned development methods, it can be seen as a double-edged sword, by the developer, because although the developer has the freedom to take decisions, it also means the developer has the responsibility to ensure the quality of the product.

Kanban uses user stories, they are put up on a board into backlog section. Then items are taken from backlog area and put into selected area, which can be compared to the sprint backlog from scrum. From there the developer takes a task and puts it into the development section, which means that this user story is now in development. After finishing with this user story, it is put into testing area. If all tests give expected results, then the user story is moved into final, live area, meaning that it is finished and ready to be released.

A limit on “work in progress area” (development area) is important, because as soon as development of new part of the project stops or slows down, it is visible on the board. Kanban has the best solution for dealing with changes in program, since new user stories can be put into backlog area on the board at any time, as long as the board isn’t filled.

#### d. Pros of Agile Development Methods

* Changes are easy to implement
* Incremental delivery
* Continuous testing, as opposed to some plan driven methods
* Improved relationship between customer and development team
* More freedom for the developer
* Increased customer satisfaction

#### e. Cons of Plan Driven Development Methods

* Big fixed price projects are difficult to manage
* Prioritizing user stories can be a challenge with multiple customers
* Refactoring is expensive
* Can be overwhelming for inexperienced developers

## 1.3. Quality Assurance

Quality assurance is part of project which is making sure that in the end, a high quality product, that satisfies the customer’s requirements, will emerge. Just as the old English proverb says: “better safe than sorry”, assuring the quality of a product, before and during the development process, is usually a cheaper and faster process of preventing errors than fixing them in a sloppy way, at the last moment.

Some external quality attributes include maintainability, reusability, reliability and usability. External quality can be ensured with pair programming, test driven development, continuous delivery and unit testing.

There are several ways to ensure the quality of a product. Following chosen development method’s instructions is first step towards developing high quality product. Final step is to have quality control. Quality control deals with testing already existing product. People in quality control make sure that the end product came out exactly as planned and the product fulfils all the requirements.

# 2. Development Process

The following portion represent the practical part of the report and is meant to show how exactly did we apply the theory, in our work.

## 2.1. Development Method of Choice

Although we have learned about the existence of many development methods, each with its own unique characteristics, we decided that the best way of choosing a development method is by evaluating the team and creating a Boehm and Turner Model.

The following image shows the model we have ended up with and according to which, we have chosen a development method.



### a. Final Choice

From the diagram above, resulted that we needed some kind of agile method, due to the high amount of expected changes and Critical level of the product, that is structured enough in order to accommodate for the high number of low level developers and Culture. We have decided to use a combination of the SCRUM and the XP methods of development taking daily meetings (usually at 10 a.m. at school) and sprint structure of the project work from SCRUM while ensuring the robustness of our program through XP and/or pair programming if necessary.

In the end we have observed that the selected criteria, came with the following perks:

### b. Pros

* The daily meetings allow to very well understand the stage at which the project is as well as identify any possibly issues and setbacks as soon as possible before they become a major problem.
* Splitting the work into sprints provides the opportunity to evaluate how well-planned was each iteration and make changes if necessary (learn from our mistakes). For example, if we assign too many tasks and fail to accomplish them by the end of the sprint, for the next sprint we will be able to adjust and plan correctly.
* Quickly identify and solve any problems that were lurking and appeared not to be there.
* Pair programming can make different developers more familiar with the code overall rather than just their own part.
* Pair programming also ensures code quality as errors are much easily noticed by two people than one.
* Optimizing and testing the code all the time greatly reduces the chance of having major bugs or missing functionality.

### c. Cons

* Planning meetings and travelling daily takes some time, which could’ve been used on work instead.
* Pair programming can sometimes take more time than wanted since 2 people are focusing on the same task.

## 2.2. System Architecture and Quality Criteria

### a. System Architecture

Our service’s architecture is a multi-tier architecture (as can be seen in following image), which provides several benefits, such as easy expandability (for example adding a mobile client) and low-cost maintainability. Not only that, but also, this architecture helps us achieve the goal we have set for ourselves, for this project, that being to pursue “high cohesion and low coupling”. Other possible architectures which we could’ve decided to choose were the classic web architecture and client/server architecture, both being dismissed because of their lower level of flexibility.

### b. Quality Criteria

When we talk about quality criteria we talk about managing risks, how software meets functional and non-functional requirements. We decided to structure all these, using the FURPS+ model.

1. Functionality:

* Our systems capability is mostly limited to a decently broad set of landscapes, providing a way of communication for anyone that would like to use it.
* In terms of reusability, the service can be consumed by anyone willing, and does not limit the access to the officially created clients (web client and dedicated client). In other words, anyone that would like to consume the service, is welcomed to create their own client and connect to the service.
* As for security, all of the implemented security features, such as SQL injections, man-in-middle and password hashing, ensure the safety of our clients.

1. Usability:

* Since we are mere backend programmers, no fancy clients come with our service, however we cannot say they look and act anything less than decent.
* The documentation that comes with the service, should at least reduce the amount of questions, potential developers or costumers that would like to consume the service, would have.

1. Reliability:

* The reliability of our service can be described as “more than satisfactory”. From our personal observations, the chance of a failure occurring is relatively low and even in the rare cases when it happens, it will affect only one client, allowing the others to go on with their business as nothing had happened. As for the one client that got any type of problem, a simple restart on their side, would solve the issue.

1. Performance:

* This subject is described in depth in the technology and programming report, under the “Performance” paragraph. At the end of which we reached the conclusion that, hosted on a low performance machine (6 years old laptop), the service can handle 25 concurrent requests, with an approximate delay of 550 milliseconds.

1. Supportability:

* Since, as previously described, our service was designed under a multi-tier architecture, and a perfect analogy of its creation would be: “like assembling a Lego house”, the service’s maintainability is extremely easy for any skilled programmers, regardless of the fact that they participated in the creation process or not. (for more details check technology and programming report, under “services architecture”)

1. + (Risks): Just as with any other project there were several risks which we encountered and handled as they appeared. Some of which were:

* Not being able to connect several users and allow them to communicate, in chatrooms. We handled this by doing a lot of research on the subject, and we knew it is possible, because of the existence of other services like Facebook, which provide a similar feature.
* Not being able to create web client. We handled this, again, by doing a lot of research on MVC and SignalR.
* Encountering exceptions that can break not only the connection to service but the service itself. Handled by catching and dealing with them everywhere we can.
* Customer changing his mind midway through development process. Handled by following an agile method of development.
* Customer wanting something that we cannot provide (unachievable goals). Handled by prioritizing user stories and researching them, then negotiating with the client.

## 2.3. Sprints Summary

Sprint work was mainly organized by scrum master and decided by product owner. Each working day started with a short meeting, in the class room or other location previously decided by group, but it usually happened at 10:00 unless majority of group agreed for different time, according to the group contract. Meeting in which the group members, with help of the scrum master, discussed the following things- which goals have been achieved since last meeting, what goals are set by the next meeting and if any problems were encountered so far.

After the meeting, work was started on the project, if there were any problems, the scrum master or any other person with experience would help in finding a solution to it. On the last sprint day there was sprint review where group members discussed the good and bad about this sprint and how can we improve in these fields, followed by goal setting for next sprint and work distribution. This process resulted in our estimations getting progressively better, thus allowing us to set more realistic goals.

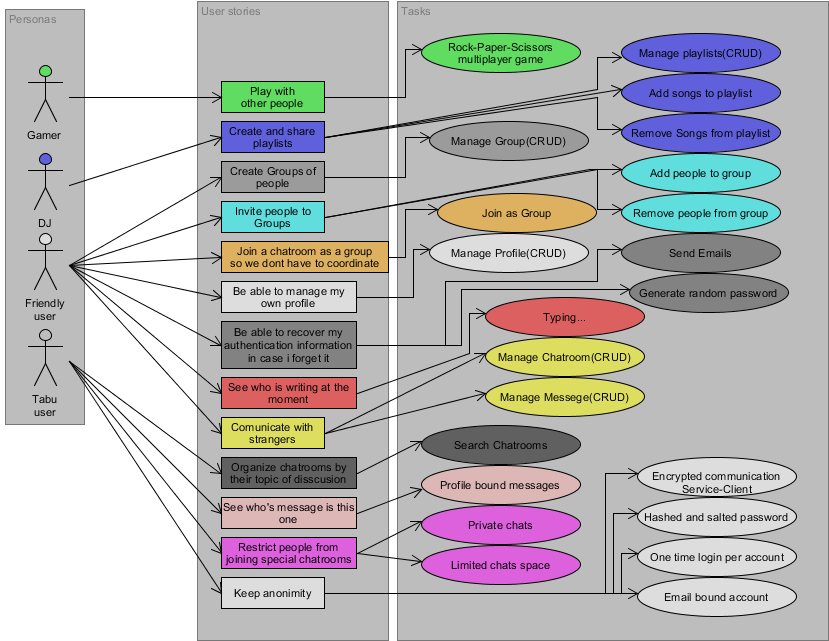
### 2.2.0. User stories

After creating the main idea of project, it was the product owner’s time to shine. And since we all got to be the product owner, at some point, the user stories kept on coming, all following the same principles.

Since our target audience is people all around the world that like to spend their time listening to music, sharing ideas and developing a network, in whatever their interests are, specifically 15-35 year old. We have identified several personas that might be encountered in that age limit and might be interested in consuming our service. Revolutionist, Friendly, DJ, Gamer. The following table represents the final version of our user stories.

|  |  |  |
| --- | --- | --- |
| **As a/an** | **I want to…** | **so that…** |
| DJ | be able to make my own playlists | I can share them with my friends and easily access them |
| Friendly User | be able to make groups of people that I enjoy spending my time with | I can easily see when they are online |
| Friendly User | be able to communicate with strangers | to make new friends |
| Friendly User | be able to invite strangers into my group | we can communicate easily |
| Friendly User | be able to manage my own profile | I will, more easily remember my authentication details |
| Friendly User | be able to recover authentication information, in case I forget it | I can keep using the same account |
| Friendly User | be able to join a chatroom with a group of friends | we don’t have to coordinate which chatroom to join |
| Friendly User | see if someone is writing | we won’t talk over each other, making a mess in the chatroom |
| Taboo  user | be able to keep my anonymity | I can share theories with other people |
| Taboo  user | be able to restrict who can come into the discussion | I will fill safe talking about taboo topics |
| Taboo  user | not be obliged to download software | it will be less likely to get hacked |
| Taboo  user | be able to organize chatrooms by their topic of discussion | it will be more easy to find what I’m looking for |
| Taboo  user | see who wrote what, in a chatroom | I know who I am talking with |
| Gamer | be able to play a game or two | I can assert my dominance over less skilled users. |

Some user stories have proven to be more complex than others, and in order to help visualize how they have been converted into tasks, we have created the following diagram, which we like to call: “User-Story-to-Task-Conversion-Diagram (USTCD)”.



### 2.2.1. Product Backlog

As you might expect, after converting user stories into tasks, we added them to product backlog, which was later used to create and facilitate sprint backlogs. These tasks, after being added to sprint backlog, were examined by scrum team and assigned time estimates to each item, to allow us to know how much we can realistically achieve. And then tasks from this backlog were put into sprint backlog by product owner, thus setting up goals for team to achieve. The following table shows how our product backlog looks like.

| **Priority** | **Name** | **Time est. (hours)** | **Done** |
| --- | --- | --- | --- |
| 1 | Mock up | 4 | YES |
| 1 | MoSCoW | 3 | YES |
| 1 | Use cases | 2 | YES |
| 1 | Group Contract | 9 | YES |
| 1 | Database | 8 | YES |
| 1 | User stories | 10 | YES |
| 1 | Report | 80 | YES |
| 1 | Domain model | 7 | YES |
| 1 | Backlog | 7 | YES |
| 1 | Database model | 12 | YES |
| 1 | Register | 3 | YES |
| 1 | Login | 2 | YES |
| 1 | Get profile | 3 | YES |
| 1 | Update profile | 3 | YES |
| 1 | Delete profile | 2 | YES |
| 1 | Multiple chats for one client | 2 | YES |
| 1 | Create Chat | 3 | YES |
| 1 | Update Chat | 3 | YES |
| 1 | Find chat | 2 | YES |
| 1 | Delete Chat | 2 | YES |
| 1 | Join chat | 4 | YES |
| 1 | Leave chat | 2 | YES |
| 1 | Send message | 5 | YES |
| 1 | Receive Message | 4 | YES |
| 1 | Get messages | 3 | YES |
| 1 | Remove your message | 2 | YES |
| 1 | Fix problem statement | 2 | YES |
| 1 | Plan driven vs agile | 3 | YES |
| 1 | Dedicated client | 25 | YES |
| 1 | Logout | 2 | YES |
| 2 | Forgot password | 3 | YES |
| 2 | Show message info | 3 | YES |
| 2 | Get songs info | 4 | YES |
| 2 | Choice of method | 4 | YES |
| 2 | Play song | 1 | YES |
| 2 | Save song | 1 | YES |
| 2 | Create playlist | 3 | YES |
| 2 | Get playlist | 1 | YES |
| 2 | Update playlist | 2 | YES |
| 2 | Delete playlist | 1 | YES |
| 2 | Join as group | 9 | YES |
| 2 | Join group | 1 | YES |
| 2 | Leave group | 1 | YES |
| 2 | Create group | 1 | YES |
| 2 | Read group | 1 | YES |
| 2 | Update group | 1 | YES |
| 2 | Delete group | 1 | YES |
| 2 | Chat users updated if info changed | 1 | YES |
| 2 | Show newest chat info | 3 | YES |
| 2 | Join as group | 9 | YES |
| 2 | Show if someone is writing | 1 | YES |
| 3 | Close chat if its deleted | 2 | YES |
| 3 | Invite person to chat | 4 | YES |
| 3 | Get notification | 3 | YES |
| 3 | Remove notifications | 2 | YES |
| 3 | One person can be logged in at once | 2 | YES |
| 3 | Multiplayer game | 6 | YES |
| 3 | Show if someone is writing | 1 | YES |
| 3 | Show online persons | 2 | YES |
| 3 | Web client | 30 | YES |

In the following portion we are going to give you a detailed diary of what we have done each sprint.

### 2.2.2. Sprint 0

#### a. Sprint summary and MoSCoW model

This sprint’s roles were distributed as follows:

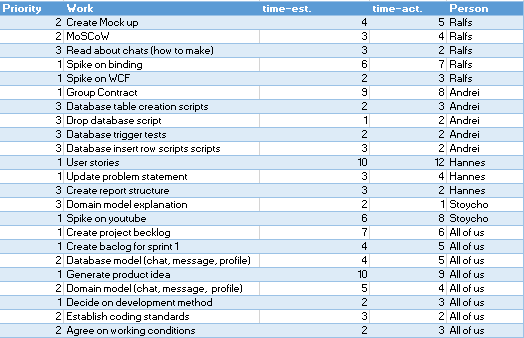
* Product owner- Hannes Heiskonen
* Scrum master- Ralf Zangis

One of the first things we needed to establish were the user stories, and once they were established we needed a way to organize them depending on their importance. Besides documentation this sprint was also used for many spikes, which include technology’s like- binding, WCF and YouTube API. And the best way to do so, is with the help of a MoSCoW Model, so the following, colorful, table, shows our final version of it.

|  |  |  |  |
| --- | --- | --- | --- |
| Must | Should | Could | Wont |
| ------------- | ------------- | ------------- | ------------- |
| Register | Forgot password | Show if someone is writing | Follow/subscribe |
| Login/Logout | Multiple chats for user | Multiple chats for person | Friends |
| Manage profile | Invite to chat | Manage YouTube playlist | News area |
| Manage public chat | Show online users | Multiplayer game |  |
| Manage private chat | YouTube video player | Join with group |  |
| Manage messages | User can’t log in if he is online |  |  |
| Dedicated client | Manage group |  |  |
|  | Web client |  |  |
|  |  |  |  |

The first version of the model looked quite differently, having a lot of text in the first two columns and little to nothing in the last two columns. But as time passed and after we pivoted from the original idea, a few times, the final version of the model slowly started to be appear, resulting in its concluding form sometime in the middle of Sprint 1.

#### b. Sprint Backlog and burndown chart



#### c. Sprint retrospective

Although the sprint was 14 days, we have considered only the 6 days that, represented by the bottom set of numbers in the chart, in which we have spent the most time working on the project.

### 2.2.3. Sprint 1

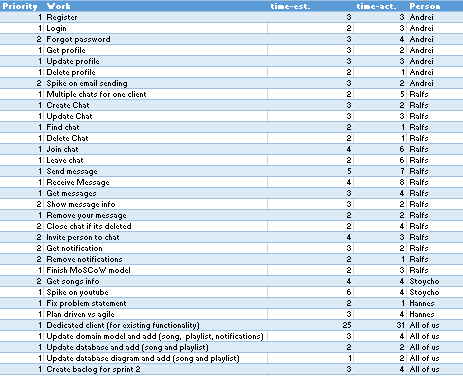
#### a. Sprint summary

This sprint the assigned roles have been changed to the following:

* Product owner- Ralf Zangis
* Scrum master- Stoycho Anastasov Nenov

Sprint 1 was possibly the most important regarding the program, as in this sprint we made spike on email sending and it also resulted in the main features of the project being finished, besides that sprint also tested the team, as there we had conflicting arguments regarding future of project. Conflicts which were solved by the product owner and the scrum master, agreeing on middle ground, resulting in project main idea changing drastically while keeping main user stories.

#### b. Sprint Backlog and burndown chart



#### c. Sprint retrospective

Although the sprint tested our group’s work limits, it was still a great success because we could agree on common project path and it resulted in all intended features being successfully implemented.

### 2.2.4. Sprint 2

#### a. Sprint summary

This sprint the assigned roles have been changed to the following:

* Product owner- Andrei-Eugen Birta
* Scrum master- Hannes Heiskonen

This sprint’s main purpose was finishing touches for the existing features, such as optimizing, refactoring, fixing some of the most obvious “features” and just adding a little sprinkle on top, to make it shine from a crowd of other projects, such as automatic resizing of elements inside a windows form. As a result of our great efforts, during this sprint, we had a completely working dedicated client with all the intended features.

#### b. Sprint Backlog and burndown chart

#### c. Sprint retrospective

This sprint was possibly the most “exciting” of them all, simply because of the problems we have encountered, such as both the Video Player and Rock-Paper-Scissors game, working only in Debug Mode. Problem which we solved and thought us an important lesson: when developing something try it both in release and debug mode and make sure it works as expected.

### 2.2.5. Sprint 3

#### a. Sprint summary

This sprint the assigned roles have been changed to the following:

* Product owner- Stoycho Anastasov Nenov
* Scrum master- Andrei-Eugen Birta

We like to call this Sprint as “The beginning of the end”, simply because this is the sprint in which we intended to wrap up the project, at least start doing so. The main goals for this sprint were finishing the web client, finishing touches for the service itself and started writing the reports. And as usual, all this sprint’s tasks can be seen in the following table, sprint backlog and the accompanying image, burndown chart.

#### b. Sprint Backlog and burndown chart



#### c. Sprint retrospective

But of course, since we have never previously worked with MVC, fell right into Mr. Brian’s “prediction”, that is: underestimating the difficulty level of creating a web client, especially since we have never worked with HTML or java script or SignalR. Just as you can see from the burndown chart, this sprint’s estimations were just as close to reality as we would ever get to winning the lottery. We had to do a spike on MVC, SignalR and several other web client related technologies.

Although at the begging of the sprint we were slightly ahead of schedule, at the end of it, time became an enemy and the race to handing in a “finished” product, began. However this sprint also had a bright side, that being increasing our knowledge regarding web clients and web servers.

### 2.2.6. Sprint 4

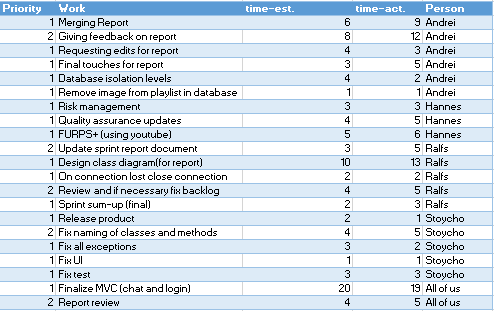
#### a. Sprint summary

This sprint the assigned roles have been changed to the following:

* Product owner- Hannes Heiskonen
* Scrum master- Ralfs Zangis

Just as the previous sprint hinted, all we have done in this last week was assembling the report, reformatting our sprint backlogs, into more easily readable formats, adding some new features to the MVC, like cookies, and making sure we still have a working product as expected and no new “unintentional features” were added, by mistake. Some other worthy mentions for this sprint are: making the web client work with dedicated client, at least regarding messaging in chats (so any both types of clients can see what the other has written), and changing the “architecture” of the technology and programming report, into a formal format, in order to increase readability.

#### b. Sprint Backlog and burndown chart



#### c. Sprint retrospective

This sprint, just as most of the other ones, ended successfully, resulting in finished product. Although last sprint didn’t go as planned, we learned from our mistakes and didn’t set unachievable goals. This sprint also helped us confirm our product’s quality and capability of many different features, implemented by different developers, that worked together as one. We tested the quality by taking the program and trying to deliberately make it break or throw exceptions, we did this because we thought it would be better to find them now, rather than finding them during the exam and suffer the embarrassment some other teams, including us, to some extent, have suffered at the final presentation we had, in the auditorium.

# 3. Conclusion

Although we encountered a lot of issues such as, taking hard decisions, group disagreements, many spikes on technologies new to us, wrong estimates regarding tasks and different ideas of working, we consider this project to be a success and are proud of our final product.

Although as mentioned before there are a lot of things that could have been changed in our ways of working, and numerous restless nights, due to disagreements, we are glad we encountered those issues, as they taught us lessons that will be useful in our future careers.

If you would like to see how we worked, what files we created, the following link will take you to our GitHub repository: https://github.com/bubriks/Turakas

## 3.1. Denouement

In conclusion, during this semester we have managed to achieve not only knowledge about different frameworks of developing software, such as Extreme Programming, SCRUM and Kanban, and seeing how actual developers work, through the company visits; but also, how to communicate and reach out to companies of the profile.

As an ending note, we would like to thank all the readers, who invested their time in reading this paper and also to the guiding teachers that helped and guided us through the entire process. Also, all of the files used in the creation of this report are attached to the hand-in folder, in case you would like to inspect them in great detail.