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 (14:00 UTC+1)

Submission date: 2017-December-17

Birta Zanigis Heiskonen Nenov

Contents

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.

a. 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 best fits both the team's composition and the project's needs. The following few frameworks are some of the development methods we have taken into consideration.

1.1.1. Plan Driven

a. Waterfall

Waterfall method follows very simple pattern. At first all the requirements are gathered. Then the 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.

b. 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, system sequence diagrams, operation contracts, interaction diagrams and design class diagram. Work is done in iterations, that have 4 phases- Inception, elaboration, construction and transition. Tests are usually done at the end of each iteration.

c. 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, because most of the decisions are taken by the designers.
- Easy to manage and control

d. 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

a. 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 the 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.

However, pair programming and TDD might be a waste of time if used on simple parts of the application.

b. *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 meetings 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 meetings are 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. Because scrum is based on incremental development, divided into iterations, called Sprints, the product's evolution can be easily seen by both the development team and the customer, also 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 path of preventing errors than fixing them in a sloppy way, at the last moment.

Quality assurance should be implemented, keeping in mind previously declared project’s quality attributes. FURPS+ is a model for classifying functional and non-functional requirements. F is for concerns regarding functionality (only functional requirement in FURPS), U is usability, R is reliability, P is performance, S is supportability and + is for others, like implementation and design requirements. Another good idea is to follow chosen development method’s instructions. By following chosen method’s instructions, it is easier to keep the project on track. Having quality control is also important when ensuring high quality in product. Quality control deals with testing already existing product. People in quality control make sure that the end product has come out exactly as planned and the product fulfils all the requirements.

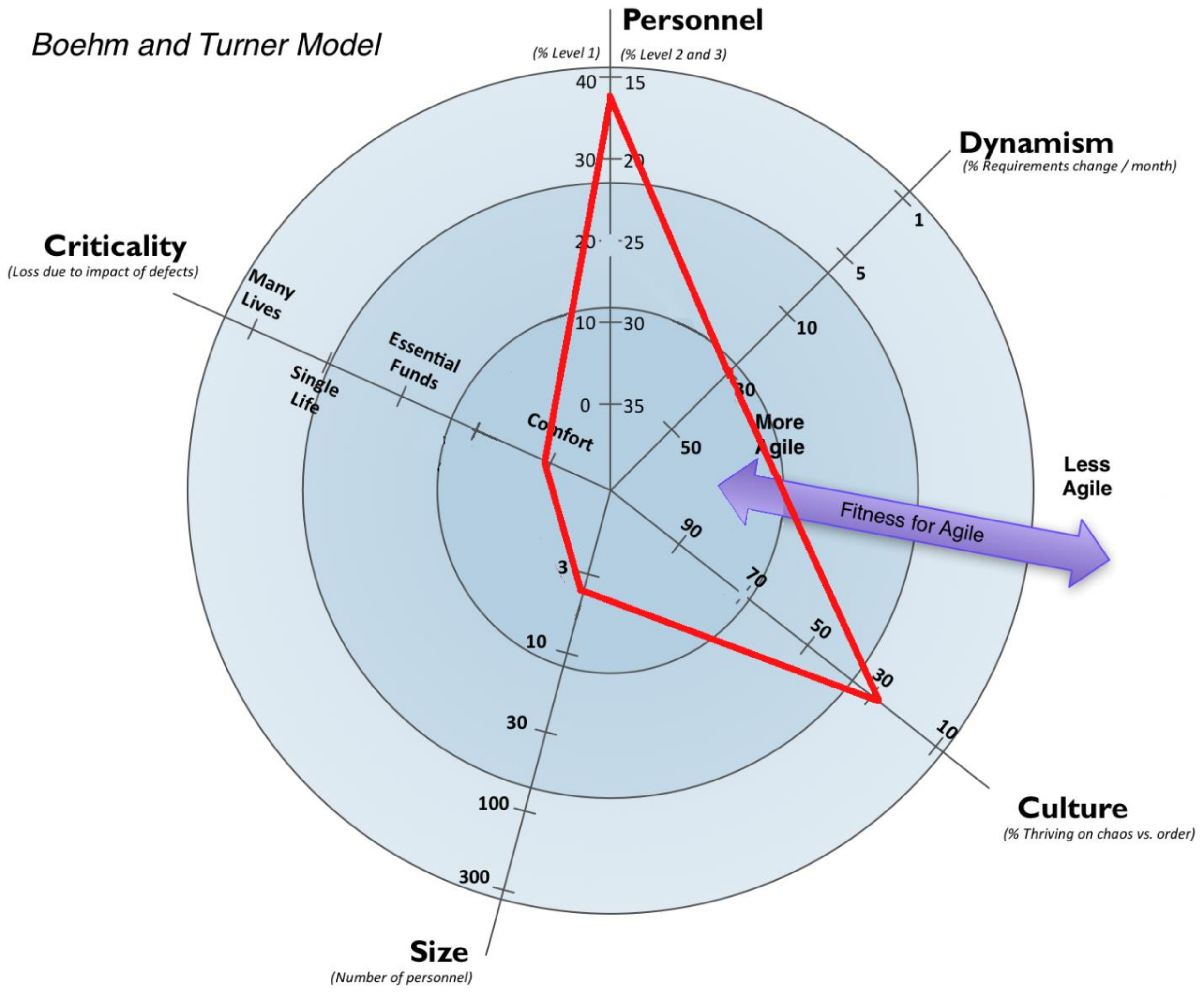
2. Development Process

The following portion represent the practical part of the report and is meant to show how we applied 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 development method, due to the high amount of expected changes, small team size and critical level of the product, but 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 SCRUM and XP methods of development, taking daily meetings and sprint structure of the project work, from SCRUM, while ensuring the sturdiness 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

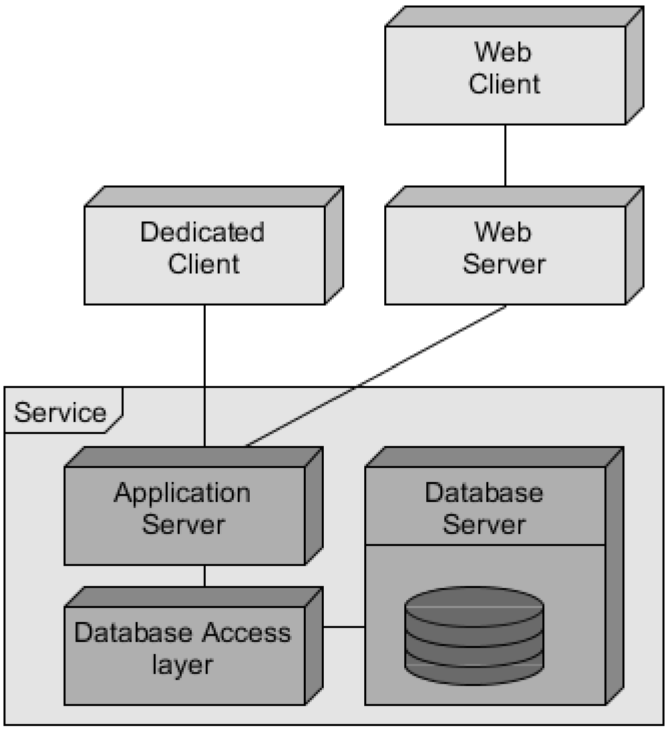
- The daily meetings allow to, very well, understand the stage at which the project is, as well as identify any possibly issues and setbacks as soon as possible, before they become a major problem.
- Splitting the work into sprints provides the opportunity to evaluate how well-planned was each iteration and make changes if necessary (learn from our own 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 accordingly.
- Quickly identify and solve any problems that were lurking and appeared not to be there.
- Pair programming can make 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 rather 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, because two 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:

Products owner's project idea, was that it should allow users to communicate with others and make communities depending on their interests and wishes. The service should be reusable, so that anyone who would like to consume it, is welcome to create their own client and connect to the service. For security, product owner clearly stated that he would rather have better performance, than improved security. That's why this solution implements only most basic security features, such as protection against SQL injections, man-in-middle attack and password hashing.

2. Usability:

As for the user interface, owner's choice was to not focus on it or aesthetics, but rather on responsiveness and consistency. Besides already mentioned requests, product owner wanted well written documentation, which comes with the service, so it would reduce the amount of questions, potential future developers or costumers that would like to consume the service, would have.

3. Reliability:

The availability of program was also one of the focus of this service, because the costumer's preference was for the service to have high uptime, and little to no downtime, thus making it more appealing for users to consume the system. Also, the failure extent should be limited to user, so if problem occurs it doesn't ruin the experience of other users and fixing of these issues should be simple, preferably just user restarting the application.

4. Performance:

Performance for project's owner was of highest importance, that's why many decisions were taken, sometimes even reducing scale of other projects requirements, to improve and support it. Some of the main things that we had to do, to achieve our goals in this part, was making the program more efficient, reducing unnecessary resource consumption and making it scalable, so there is no need to change all the project to support more users.

5. Supportability:

As other parts of our project, it was important, and when possible, we decided to try making it as serviceable, maintainable, sustainable and repairable as possible. The product owner decided to make our product extendable, so at later stage in development, or when needed, new types of clients could be added to it. Testability and flexibility was also required by product owner, to produce high quality software, because the end result was not supposed to be a short-term solution, but rather a long-term investment.

6. Plus:

Some of other requests from product owner were to follow implementation requirements, like using TDD (test driven development), using and implementing YouTube API. The service has to be able to run on relatively weak server with little to no problems.

c. 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 and checking the user inserted data, in order to make sure it is consistent to our standards.
- 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 classroom at 10:00 unless majority of group agreed for different time and place, according to the group contract. In meetings, the group members, with help from 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 meetings, we started working 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

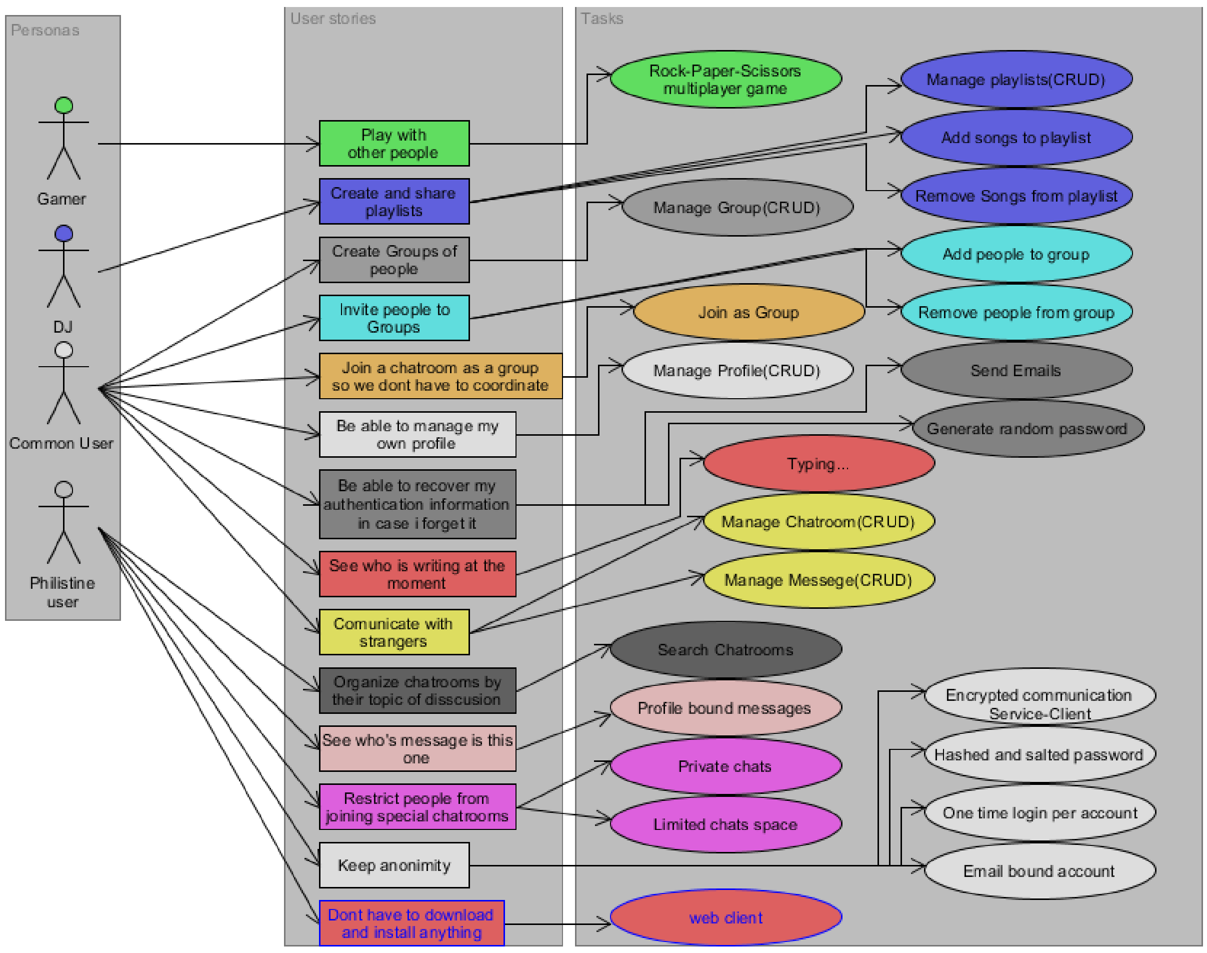
After creating the main idea of our project, it was the product owner's time to shine. We all got to be the product owner, at some point, so the user stories kept on coming, all following the same principles.

Since our target audience is people all around the world, who 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: Philistine user, Common user, 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
Common User	be able to make groups of people that I enjoy spending my time with	I can easily see when they are online
Common User	be able to communicate with strangers	to make new friends
Common User	be able to invite strangers into my group	we can communicate easily
Common User	be able to manage my own profile	I will, more easily remember my authentication details
Common User	be able to recover authentication information, in case I forget it	I can keep using the same account
Common User	be able to join a chatroom with a group of friends	we don't have to coordinate which chatroom to join
Common User	see if someone is writing	we won't talk over each other, making a mess in the chatroom

As a/an	I want to…	so that…
Philistine user	be able to keep my anonymity	I can share theories with other people
Philistine user	be able to restrict who can come into the discussion	I will fill safe talking about taboo topics
Philistine user	not be obliged to download software	it will be less likely to get hacked
Philistine user	be able to organize chatrooms by their topic of discussion	it will be more easy to find what I'm looking for
Philistine 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 us visualize how they have been converted into tasks, and help ease the transition from user stories to product backlog tasks, we created the following diagram, which we like to call: "User-Story-to-Task-Conversion-Diagram (USTCD)".


Personas
User stories
Tasks
Gamer
DJ
Common User
Philistine user
Play with other people
Create and share playlists
Create Groups of people
Invite people to Groups
Join a chatroom as a group so we dont have to coordinate
Be able to manage my own profile
Be able to recover my authentication information in case i forget it
See who is writing at the moment
Comunicate with strangers
Organize chatrooms by their topic of disscusion
See who's message is this one
Restrict people from joining special chatrooms
Keep anonimity
Dont have to download and install anything
Rock-Paper-Scissors multiplayer game
Manage playlists(CRUD)
Add songs to playlist
Remove Songs from playlist
Manage Group(CRUD)
Add people to group
Join as Group
Remove people from group
Manage Profile(CRUD)
Send Emails
Generate random password
Typing...
Manage Chatroom(CRUD)
Manage Messege(CRUD)
Search Chatrooms
Profile bound messages
Private chats
Limited chats space
Encrypted communication Service-Client
Hashed and salted password
One time login per account
Email bound account
web client

2.2.1. Product Backlog

As one 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 allow us to know how much we can realistically achieve in given timeframe. Tasks were then taken from the product backlog and put into sprint backlog by the product owner, thus setting up goals for team to achieve. The following table shows how our product backlog looks like. (see appendices, fig.1)

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

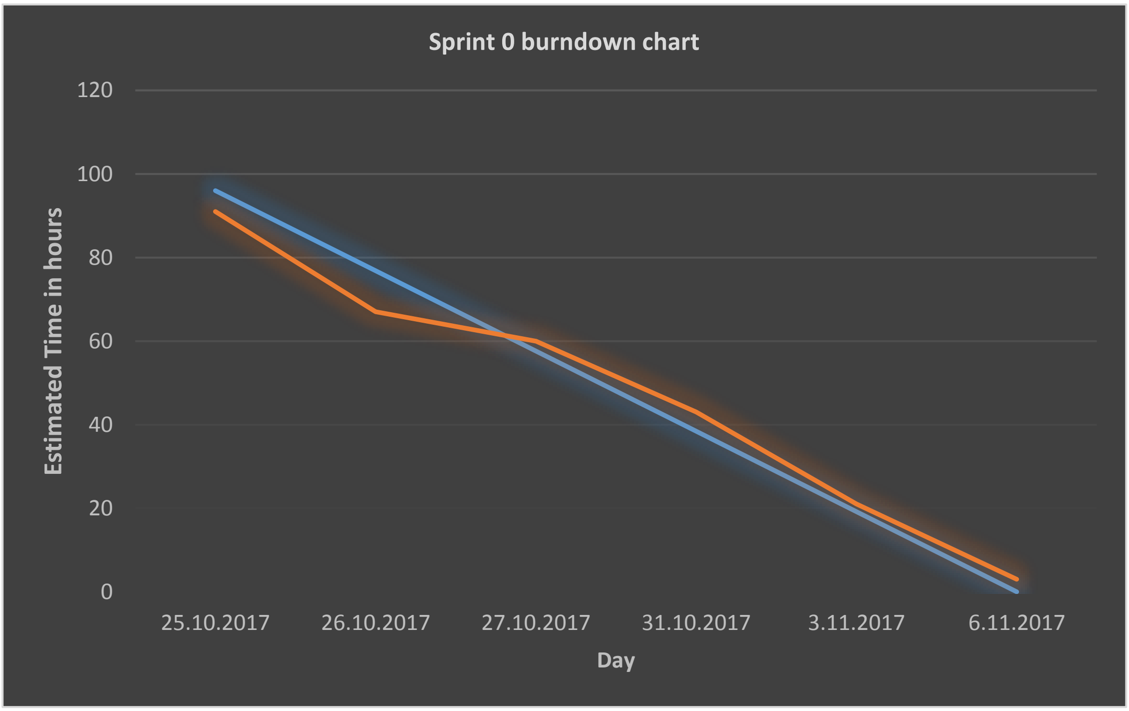
One of the first things we needed to establish were the user stories, and once they were established we needed a way to organize them depending on their importance. Besides documentation this sprint was also used for many spikes, which include technologies like- binding, WCF and YouTube API. And the 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	
	User can't log in if he is online		
Manage messages			
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

b. Sprint Backlog and burndown chart

Priority	Work	time-est.	time-act.	Person
2	Create Mock up	4	5	Ralfs
2	MoSCoW	3	4	Ralfs
3	Read about chats (how to make)	3	2	Ralfs
1	Spike on binding	6	7	Ralfs
1	Spike on WCF	2	3	Ralfs
1	Group Contract	9	8	Andrei
3	Database table creation scripts	2	3	Andrei
3	Drop database script	1	2	Andrei
3	Database trigger tests	2	2	Andrei
3	Database insert row scripts scripts	3	2	Andrei
1	User stories	10	12	Hannes
1	Update problem statement	3	4	Hannes
3	Create report structure	3	2	Hannes
3	Domain model explanation	2	1	Stoycho
1	Spike on youtube	6	8	Stoycho
1	Create project becklog	7	6	All of us
1	Create baclog for sprint 1	4	5	All of us
2	Database model (chat, message, profile)	4	5	All of us
1	Generate product idea	10	9	All of us
2	Domain model (chat, message, profile)	5	4	All of us
1	Decide on development method	2	3	All of us
2	Establish coding standards	3	2	All of us
2	Agree on working conditions	2	3	All of us



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. This sprint was quite useful as it helped us learn a bunch of the things we just had a vague idea about. In this sprint we decided to focus more on chatting option rather than music, simply because we were lacking in concurrency issues. One ludicrous idea, which we considered, for a brief moment, was creating a poker game, which got dismissed quickly, because of unknown amount of work. Although poker gave us a good idea regards joining tables. We took the idea of joining a chat as a group from it.

2.2.3. Sprint 1

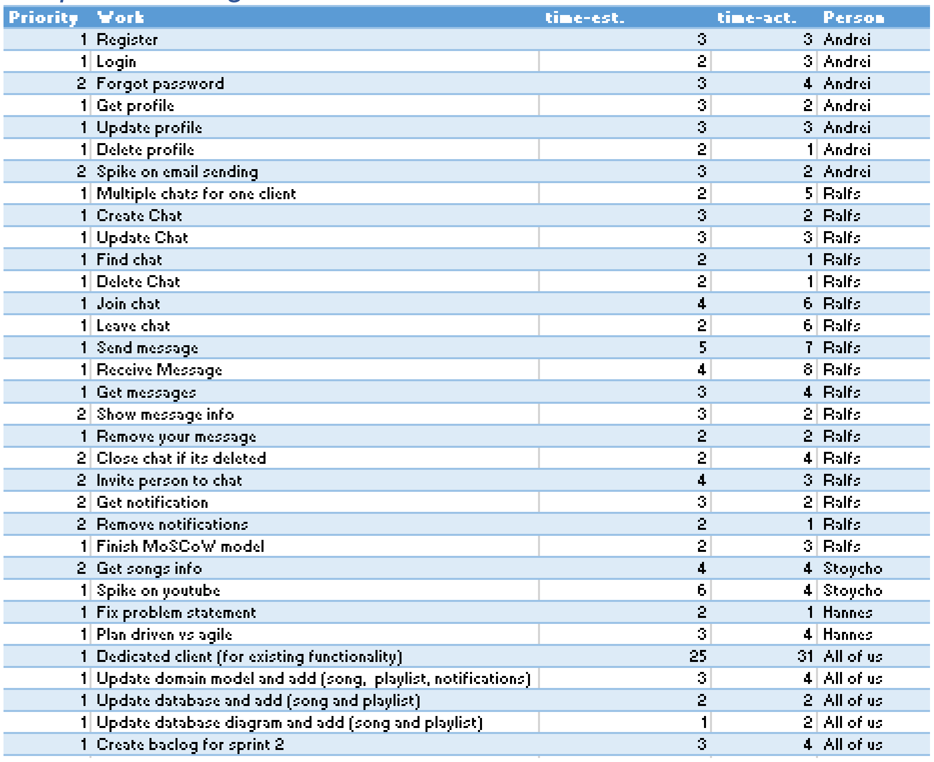
a. Sprint summary

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

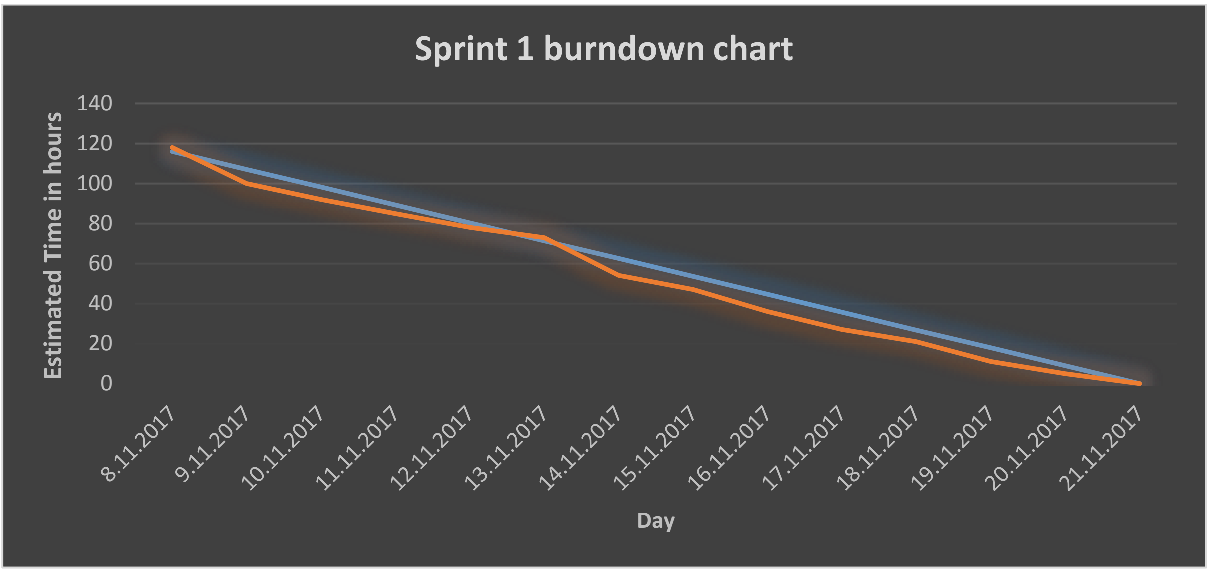
- Product owner- Ralf Zangis
- Scrum master- Stoycho Anastasov Nenov

Sprint 1 was possibly the most important regarding the program. 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, because we had conflicting arguments regarding the future of the 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



Priority	Work	time-est.	time-act.	Person
1	Register	3	3	Andrei
1	Login	2	3	Andrei
2	Forgot password	3	4	Andrei
1	Get profile	3	2	Andrei
1	Update profile	3	3	Andrei
1	Delete profile	2	1	Andrei
2	Spike on email sending	3	2	Andrei
1	Multiple chats for one client	2	5	Ralfs
1	Create Chat	3	2	Ralfs
1	Update Chat	3	3	Ralfs
1	Find chat	2	1	Ralfs
1	Delete Chat	2	1	Ralfs
1	Join chat	4	6	Ralfs
1	Leave chat	2	6	Ralfs
1	Send message	5	7	Ralfs
1	Receive Message	4	8	Ralfs
1	Get messages	3	4	Ralfs
2	Show message info	3	2	Ralfs
1	Remove your message	2	2	Ralfs
2	Close chat if its deleted	2	4	Ralfs
2	Invite person to chat	4	3	Ralfs
2	Get notification	3	2	Ralfs
2	Remove notifications	2	1	Ralfs
1	Finish MoSCoW model	2	3	Ralfs
2	Get songs info	4	4	Stoycho
1	Spike on youtube	6	4	Stoycho
1	Fix problem statement	2	1	Hannes
1	Plan driven vs agile	3	4	Hannes
1	Dedicated client (for existing functionality)	25	31	All of us
1	Update domain model and add (song, playlist, notifications)	3	4	All of us
1	Update database and add (song and playlist)	2	2	All of us
1	Update database diagram and add (song and playlist)	1	2	All of us
1	Create baclog for sprint 2	3	4	All of us



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

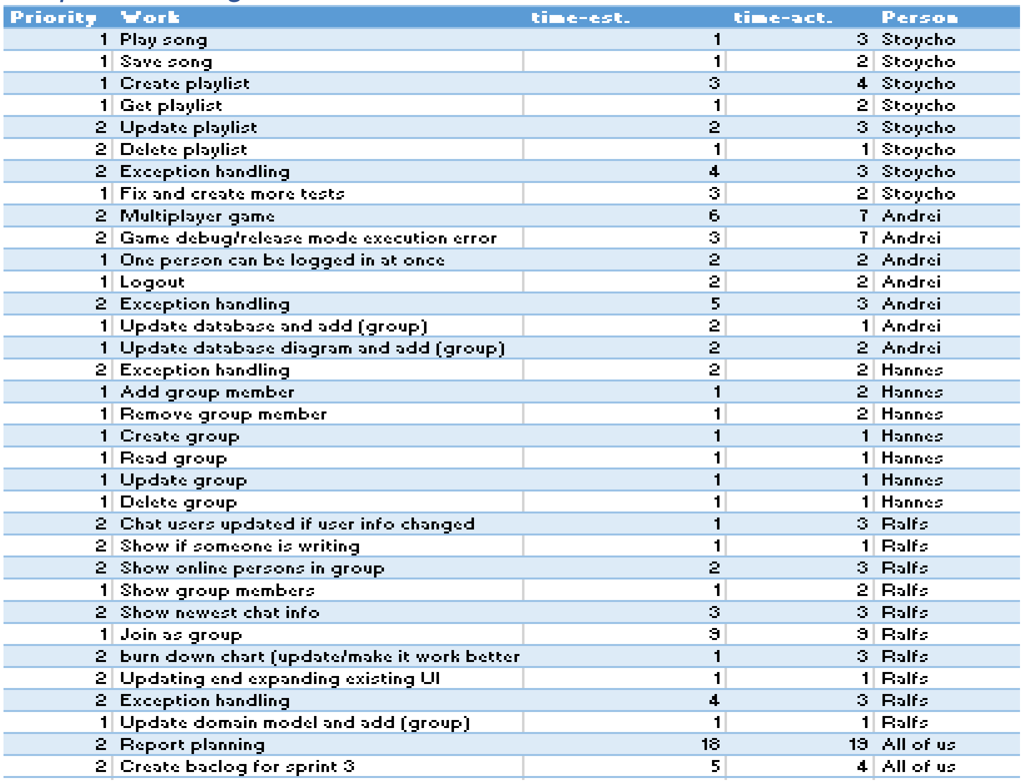
a. Sprint summary

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

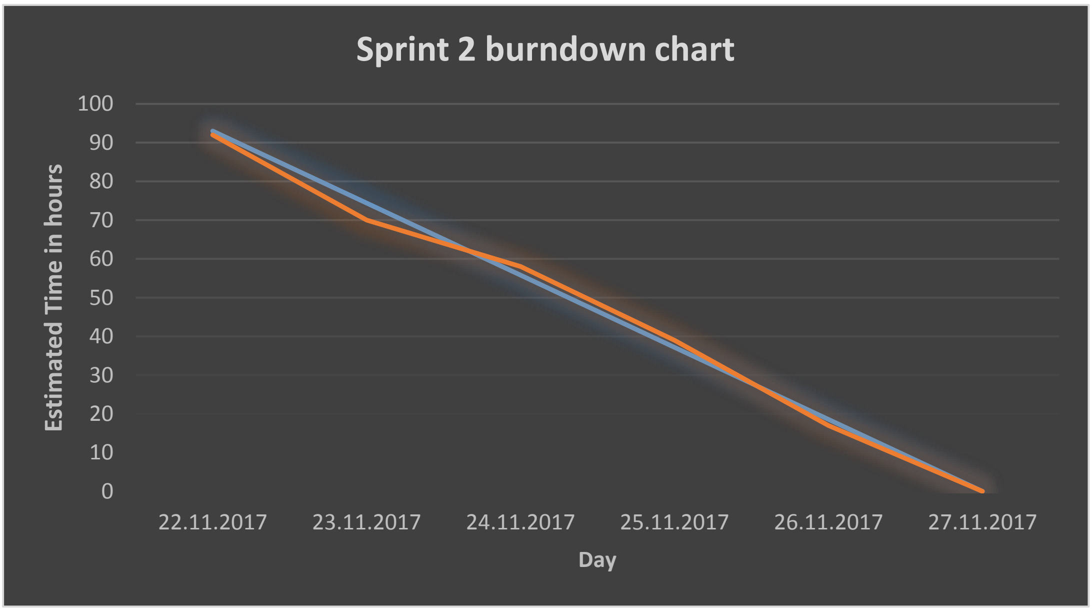
- Product owner- Andrei-Eugen Birta
- Scrum master- Hannes Heiskonen

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. Sprinkles, such as automatic resizing of elements inside a windows form and colored labels. 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



Priority	Work	time-est.	time-act.	Person
1	Play song	1	3	Stoycho
1	Save song	1	2	Stoycho
1	Create playlist	3	4	Stoycho
1	Get playlist	1	2	Stoycho
2	Update playlist	2	3	Stoycho
2	Delete playlist	1	1	Stoycho
2	Exception handling	4	3	Stoycho
1	Fix and create more tests	3	2	Stoycho
2	Multiplayer game	6	7	Andrei
2	Game debug/release mode execution error	3	7	Andrei
1	One person can be logged in at once	2	2	Andrei
1	Logout	2	2	Andrei
2	Exception handling	5	3	Andrei
1	Update database and add (group)	2	1	Andrei
1	Update database diagram and add (group)	2	2	Andrei
2	Exception handling	2	2	Hannes
1	Add group member	1	2	Hannes
1	Remove group member	1	2	Hannes
1	Create group	1	1	Hannes
1	Read group	1	1	Hannes
1	Update group	1	1	Hannes
1	Delete group	1	1	Hannes
2	Chat users updated if user info changed	1	3	Ralfs
2	Show if someone is writing	1	1	Ralfs
2	Show online persons in group	2	3	Ralfs
1	Show group members	1	2	Ralfs
2	Show newest chat info	3	3	Ralfs
1	Join as group	9	9	Ralfs
2	burn down chart (update/make it work better	1	3	Ralfs
2	Updating and expanding existing UI	1	1	Ralfs
2	Exception handling	4	3	Ralfs
1	Update domain model and add (group)	1	1	Ralfs
2	Report planning	18	19	All of us
2	Create baclog for sprint 3	5	4	All of us



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

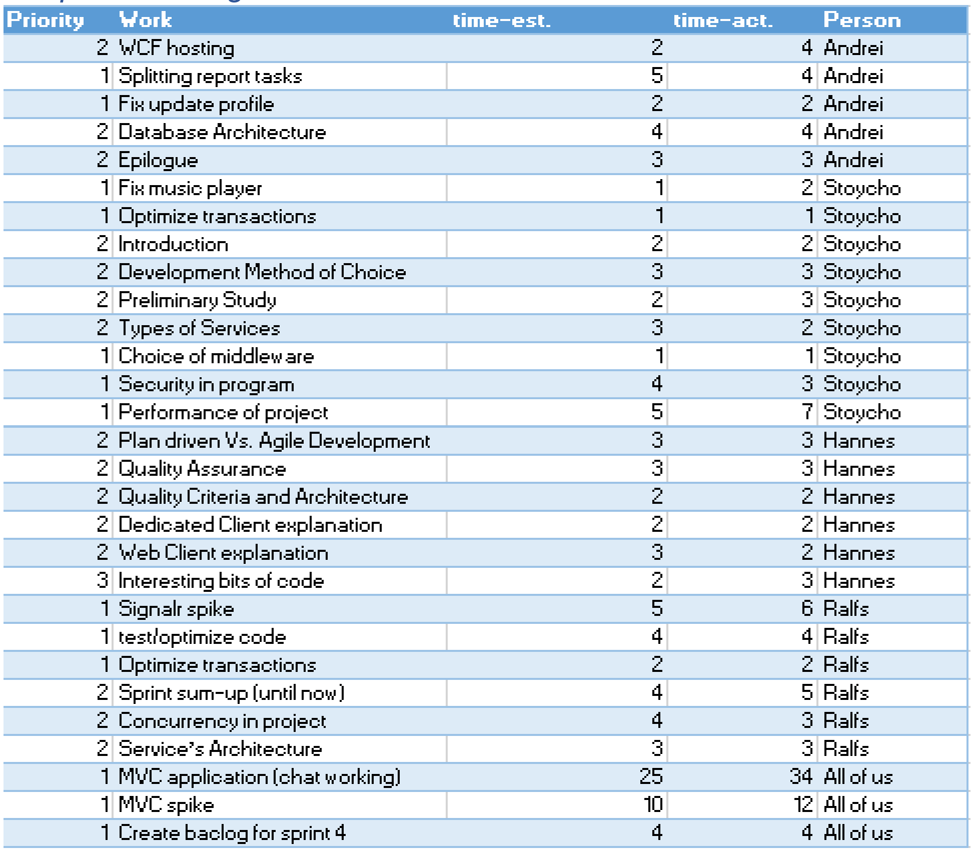
a. Sprint summary

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

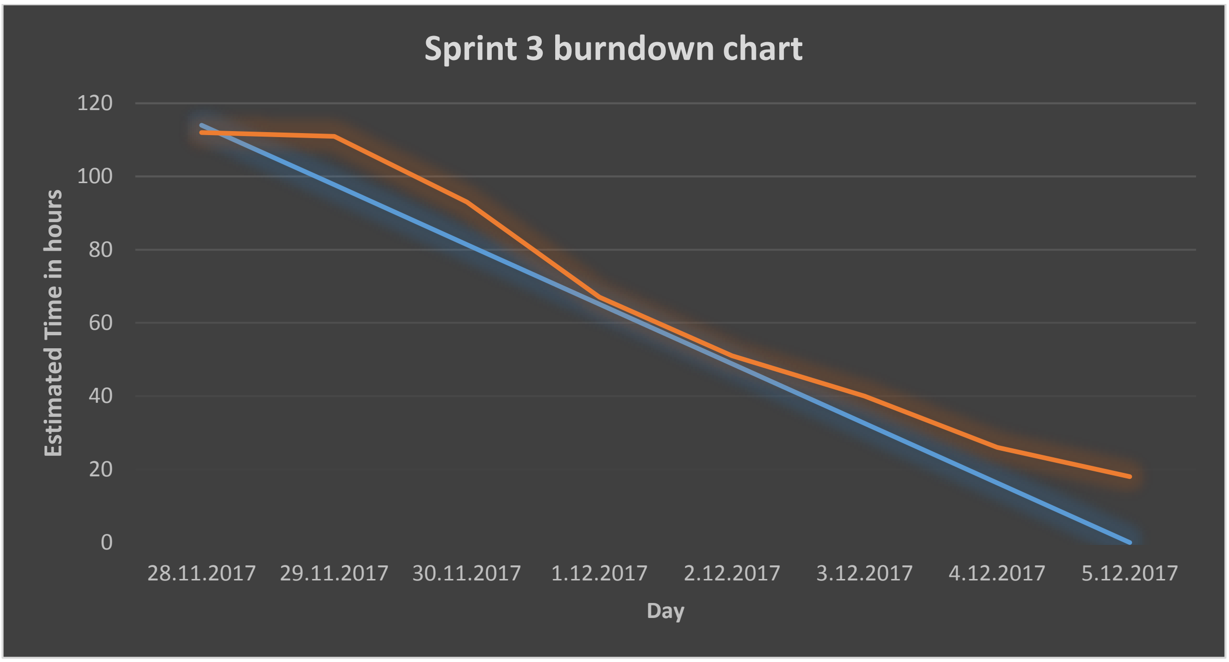
- Product owner- Stoycho Anastasov Nenov
- Scrum master- Andrei-Eugen Birta

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



Priority	Work	time-est.	time-act.	Person
2	WCF hosting	2	4	Andrei
1	Splitting report tasks	5	4	Andrei
1	Fix update profile	2	2	Andrei
2	Database Architecture	4	4	Andrei
2	Epilogue	3	3	Andrei
1	Fix music player	1	2	Stoycho
1	Optimize transactions	1	1	Stoycho
2	Introduction	2	2	Stoycho
2	Development Method of Choice	3	3	Stoycho
2	Preliminary Study	2	3	Stoycho
2	Types of Services	3	2	Stoycho
1	Choice of middleware	1	1	Stoycho
1	Security in program	4	3	Stoycho
1	Performance of project	5	7	Stoycho
2	Plan driven Vs. Agile Development	3	3	Hannes
2	Quality Assurance	3	3	Hannes
2	Quality Criteria and Architecture	2	2	Hannes
2	Dedicated Client explanation	2	2	Hannes
2	Web Client explanation	3	2	Hannes
3	Interesting bits of code	2	3	Hannes
1	Signalr spike	5	6	Ralfs
1	test/optimize code	4	4	Ralfs
1	Optimize transactions	2	2	Ralfs
2	Sprint sum-up (until now)	4	5	Ralfs
2	Concurrency in project	4	3	Ralfs
2	Service’s Architecture	3	3	Ralfs
1	MVC application (chat working)	25	34	All of us
1	MVC spike	10	12	All of us
1	Create baclog for sprint 4	4	4	All of us



c. Sprint retrospective

We underestimated the difficulty level of creating a web client, especially because we have never worked with HTML or java script or SignalR, just like Mr. Brian predicted. 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

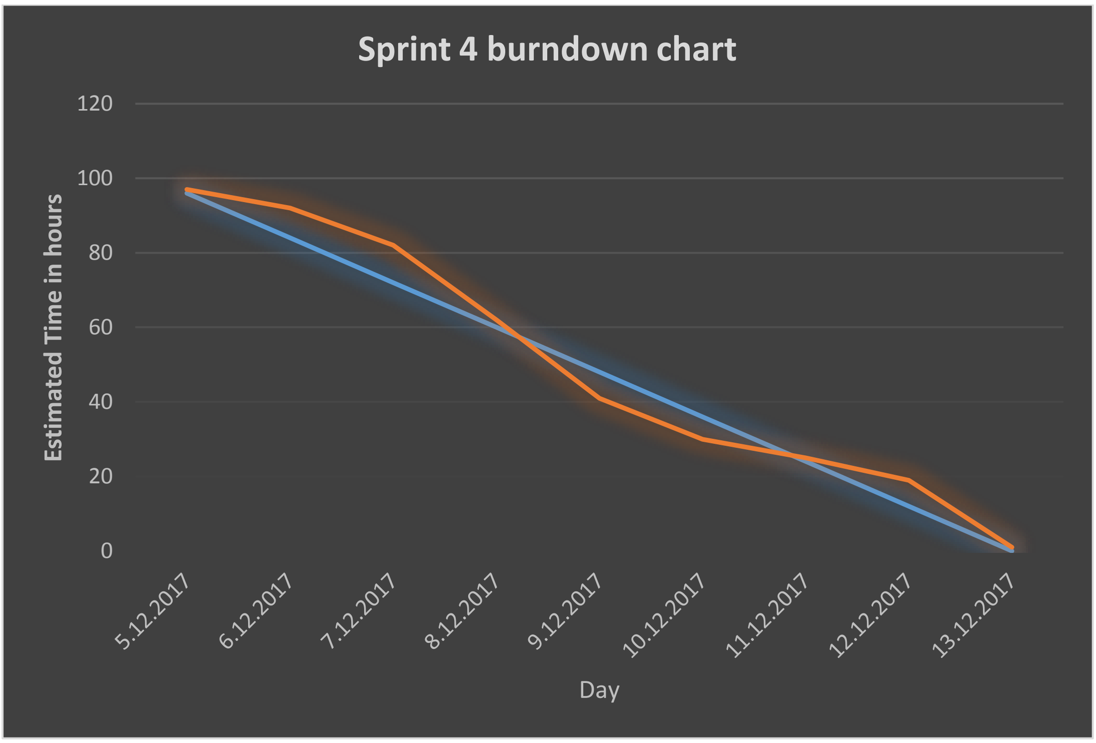
This sprint the assigned roles have been changed to the following:

- 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 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, to increase readability.

b. Sprint Backlog and burndown chart

Priority	Work	time-est.	time-act.	Person
1	Merging Report	6	9	Andrei
2	Giving feedback on report	8	12	Andrei
1	Requesting edits for report	4	3	Andrei
1	Final touches for report	3	5	Andrei
1	Database isolation levels	4	2	Andrei
1	Remove image from playlist in database	1	1	Andrei
1	Risk management	3	3	Hannes
1	Quality assurance updates	4	5	Hannes
1	FURPS+ (using youtube)	5	6	Hannes
2	Update sprint report document	3	5	Ralfs
1	Design class diagram(for report)	10	13	Ralfs
1	On connection lost close connection	2	2	Ralfs
2	Review and if necessary fix backlog	4	5	Ralfs
1	Sprint sum-up (final)	2	3	Ralfs
1	Release product	2	1	Stoycho
2	Fix naming of classes and methods	4	5	Stoycho
1	Fix all exceptions	3	2	Stoycho
1	Fix UI	1	1	Stoycho
1	Fix test	3	3	Stoycho
1	Finalize MVC (chat and login)	20	19	All of us
2	Report review	4	5	All of us



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 is better to find them now, rather than finding them during the exam and suffer the embarrassment. Some teams, including us, suffered at the final presentation we had, in the auditorium.

3. Conclusion

Although we encountered a lot of issues such as, making 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.

Even though, as mentioned before, there are a lot of things that could have been changed in the ways we worked, we are glad, we encountered those issues. Those issues taught us lessons, like how to conduct structured verbal arguments and how to better predict how much time a task takes, that will be useful in our future careers.

To see how we worked and what files we created, one has to follow the link, which will take you to our GitHub repository: https://github.com/bubriks/Turakas

3.1. Denouement

In conclusion, during this semester we managed to not only gain knowledge about different frameworks for 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, also the guiding teachers, who helped and guided us throughout the entire process. All 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.

3.2. References

*Inspired FURPS+: https://www.ibm.com/developerworks/rational/library/3975.html

**Inspired burndown charts: https://www.extendoffice.com/documents/excel/2446-excel-burndown-chart-or-burn-up-chart.html

3.3. Appendices

Fig. 1 – Product Backlog

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

Priority	Name	Time est. (hours)	Done
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

Priority	Name	Time est. (hours)	Done
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