UCN dmai0916

3rd semester project

Group 1

SYSTEM DEVELOPMENT REPORT

ConQuestion Game

18th December 2017

Mirjana Erceg
Tamas Kalapacs
Zahro-Madalina Khaji
Sangey Lama
Andreas Richardsen

University College of Northern Denmark

Technology and Business

Computer Science AP Degree Programme

System development report

dmai0916
ConQuestion Game

Project participants:

Mirjana Erceg
Tamas Kalapacs
Zahro-Madalina Khaji
Sangey Lama
Andreas Richardsen

Supervisor:

Dimitrios Kondylis

Submission Date:

18th December 2017

Table of Contents

Introduction 4
Problem statement 4
  Idea generation 4
  Vision statement 5
  Personas and Scenarios 5
      Examples 5
Theory 7
  Plan driven versus agile development 7
  Choice of method 9
  Risk analysis 10
  Main principles in planning and quality management 11
    Planning (Sprint 0) 11
    Quality management 13
      Quality assurance and quality control 13
      Quality criteria and architecture 15
Reflections on methods and their use in practice 16
  Sprint 1 16
  Sprint 2 18
  Sprint 3 20
  Sprint 4 22
Conclusion 24
References 25

Introduction

In this report we will present the differences between plan-driven and agile development. Introduce some of the Agile methodologies, mainly Scrum and eXtreme Programmming. Then we shall describe our experience in using Scrum through planning and execution of our project in which we made a system ConQuestion Game.

Problem statement

There are not enough quiz-like games that you can use with friends for learning, and that are customizable.

By using what we have learned from our course and from research we will make a conquest - quiz game which focuses on battling for game resources which are decided in a quiz based on a chosen subject e.g. computer science. The goal of the game is to have the most resources at the end of the game when time/rounds are up or when one player conquers all the resources. At the start of the game everyone has a base and if it gets conquered then the player's resources are transferred. Our purpose is to create a tool for learning together in an enjoyable way for everyone who needs it.

Some of the challenges we expect to face during this project include: Cheating & security – preventing players from modifying their clients in any way to gain an unfair advantage. Transactions and race conditions – Managing scenarios where players will race to answer correctly first. Handling performance, a big issue with any online game is performance if players experience lengthy delays it can ruin game experiences. Other issues that may need consideration include, players disconnecting during an active game, issues with security, implementing agile principles etc.

Idea generation

Before we started thinking about our projects ideas, a few agreements were made. We wanted this project to be challenging for us and we wanted everyone to enjoy working on it.

To start our brainstorming session, we had everyone come up with ideas separately. After each member had at least three thoughts written down, we discussed every idea together. The result of this session was a local trading application.

As soon as we started working on that, we started feeling like it was too similar to our previous projects, so it would not be as challenging as we wanted it to be.

In this session we took a different approach. Instead of focusing on our requirements, we focused on the agreements we made before. We came up with the idea of making a game. Here, another problem appeared, we did not know what kind of a game should we create.

We had two different thoughts, a quiz game and a two-dimensional survival game. The group felt like one was too simple and the other one was too difficult in consideration of the time we had available. We tried making the survival game simpler, but not long after that, we realised that making the quiz game more complicated was the way to go.

Vision statement

Our vision is to help students and anyone that needs to practice what they learned. If you are old, young or just need to strengthen your knowledge, we want to help by making it fun, educational and that it can be shared with friends and colleges. So, you will not have to do it alone. We are introducing the possibility of playing with people from all over the world and sharing your questions with them.

Personas and Scenarios

The customer is responsible for creating personas and scenarios, unfortunately we do not have a real customer, so we will create them as a group.

Personas are created as imaginary persons which in this case, present a user who would benefit from our system. They help us think on a level of which we as developers might not yet understand.

Scenarios are short descriptions about personas (users), it includes their basic information, interests and future goals. A brief description of their everyday life is created.

Creating personas and scenarios makes the writing of User stories easier and we are more likely to create better systems.

Examples

Mark Markus, 19

Background: Student

Description:

Mark is a second year Game development student. He is single, unemployed and sharing the apartment with his roomate in the centre of Copenhagen. In his free time he enjoys playing games, reading and playing some light sports. At the University he is not an overly active student and doesn't pay much attention in classes, the teachers are boring from his perspective.

Goals:

- His goal is to learn as much as he can in a fun way
- Ideally, he would like a game that would challenge him to study and learn more
- He would like to make more friends

Scenario:

Mark comes to University earlier than usual. He hears his classmates talking about a new quiz game so he decides to download it. The big exam is coming up soon and Mark needs to start studying and what better way to start then with a quiz game. He starts playing open online games with random people, and finally with his classmates so it becomes kind of race for knowledge. Mark feels like he is opening up to new frendships and is more confident that he can become a good Game developer. Mark is learning a lot from the questions in the game and started paying attention in class and studying at home.

Tina Johansen, 27

Background: Marketing manager

Description:

Tina is currently employed as a marketing manager with 6 years of experience and an Economic University degree. She works for a big marketing company in Aarhus. The company's number of IT business partners is increasing and it has become obvious that knowledge about new technologies is a must. She is a bookworm, has excellent social skills but doesn't know much about technology.

Goals:

- She wants to take some IT courses
- Learning at a fast pace because she is usually very busy
- Wants to be promoted
- Fit into the IT world, so she can have stronger connections with her business partners

Scenario:

Tina comes home from work and she is very tired. Her boss yelled at her again for not knowing anything about IT during a meeting they had today. She opens a bottle of red wine, pours herself a glass and opens Google of course. She is determined to find something that is going to help her get that promotion she wants for years. Google's search engine finds the most popular quiz game for learning, she clicks on it and starts playing. After playing for a while her confidence rises and she decides to enroll a course. The next few weeks she shares the game link with her colleagues and they all play it during the breaks or after work. The marketing company signs the contract with the creators of the game within months and Tina is the Project manager for their future projects and she finally gets the promotion that she deserved.

Theory

Plan driven versus agile development

Plan driven development is a process where all the project activities are planned and determined in advance. We have encountered several Plan driven models before, such as the Waterfall model, Iterative and incremental plan-driven model, V-model and Spiral model.

In the sequential Waterfall model all the requirements and system specifications are determined and documented before the start of the development process. The phases are as followed: making of software and system design, the implementation and unit testing, the integration or verification and system testing, and finally maintenance of the fully functioning system. However, some researches show that this approach has the highest estimate of failing in developing software, therefore the iterative methods proved having higher productivity and progress percentages.

Iterative and incremental plan driven development such as Unified process implements project development from smaller parts, meaning that we break down the system in several iterations, and with each iteration completed system grows (expands). The system's information, features and requirements are determined in the Inception phase, when the interview with the customer is conducted. In the Elaboration phase the architecture and the complete detailed design of the system is drafted. In the Construction phase, integration, implementation, testing and re-designing is performed through several iterations. Transition phase is for deployment of completed software to the customer, but before that extra testing must be performed. If everything is satisfactory documentation is finalised and the software is handed over to the client. All phases could potentially have more than one iteration, especially in the construction phase where most of the implementation is performed.

The V-model is also very dependent on the fundamental documentation created in the starting phase just like the Waterfall model. The uniqueness of this model is that all the testing is carefully predesigned, so it can be performed later in the process. This kind of testing is also called Beta testing, where the system is being launched as a software product to several customers which will then use the software and report any problems to the development team.

The spiral model represents a method of software development where the requirements are developed in distinct levels of detail. The starting phase of the process is mostly business oriented, user and non-functional requirements are determined. The following phase includes prototyping, in this phase we can use agile development instead, meaning we can develop requirements and system implementation at the same time. At the time of the requirements elicitation process, developers work with the stakeholders, trying to determine the demands of the system. New requirements may be discovered and previous can be changed, and they are specified, modelled and reviewed, therefore the original problem statement must be changed.

Some of the biggest problems of plan driven development are as following: creating extensive documentation which is considerably time-consuming (painfully detailed process), the whole process cannot be accurately estimated, unable to project the final version of the product, the main features of the program are misunderstood and/or are not implemented and the changes are much harder to execute which leads to more expenses, the level of stress is higher because the development teams tend to be often distracted by the complexity of occurred issues and change of requirements etc.

On the other hand, agile development planning and process is executed in small increments throughout the entire process, therefore it is easier to perform the necessary changes and overall risk is greatly reduced, product owners have some vision of the end product.

Frequent communication with the customer is a very important segment of the process because receiving constant feedback reduces the risk of unexpected refactoring at the end of the development process. Agile software development implements testing and programming right from the start of the process, leading to a working prototype of software at the end of each cycle followed by minimal documentation and system specification.

When we are talking about creating larger and more complex or critical systems (e.g. medical instruments), the best method to use is the plan-driven process simply because the problems with iterative development of large systems can become immense. Large systems require stable design and architecture which must be clearly defined in advance alongside extensive documentation. Agile practices are the most fitting for new software development, and not so much for maintaining existing systems.

Agile software development includes several different methods like eXtreme programming (XP), Scrum and Kanban.

Extreme programming implements pair programming and test first development in many short development cycles. The requirements are defined as User stories (functionalities of the system) and divided in several tasks. Programmers write the unit tests for each task, after all tests have passed the code is integrated into the system. Communication within the development team is crucial, customer is responsible for writing user stories, determining their priority and writing acceptance tests. The design is fairly simple, and changes are easy to implement, meaning maintainability is improved by continuously refactoring the existing code. The core values are courage, simplicity, feedback and communication (and respect added recently).

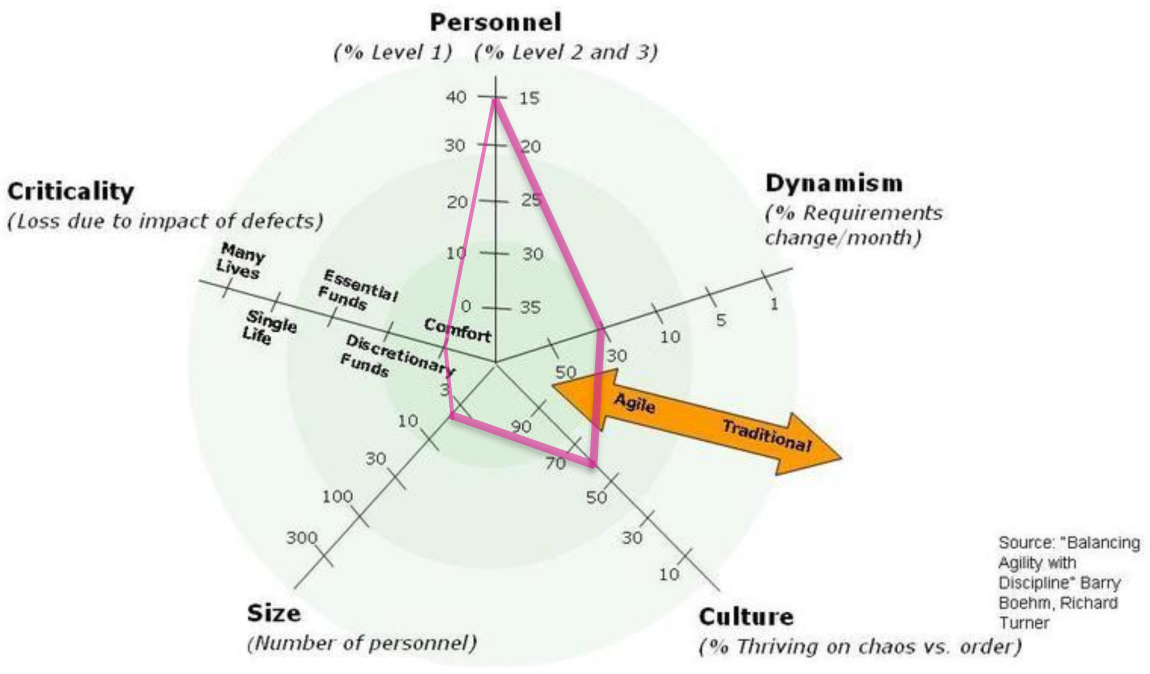
Scrum is a most widely used agile method consisting of several development iterations called Sprints that can be anywhere from two to four weeks long. All the team members are assigned certain roles, but only three are defined, Product owner, Development team and Scrum master. Usage of practices is not specifically defined; thus, it is more adaptable to already existing practices in the company.

Kanban is a lean method that has more freedom than XP and Scrum, but it requires more experienced developers and iterations are not timeboxed. Like in previously explained agile methods, in Kanban the work is broken down into tasks. Explicit time limitations are set for each task, and leading time is measured (average time of completing a task for a better optimisation of the process).

Although both plan-driven, and agile methodologies have their own advantages and disadvantages, every company can choose what suits best to their work environment. Furthermore, there is a possibility to combine some of the plan driven practices with and vice-versa.

Choice of method

Our choice of a preferred software development methodology was determined by a tool developed by Boehm and Turner. It is a Dimensions affecting method selection diagram which helped us to decide on the methods we are going to work with, plan-driven or agile.



This diagram shows the Five critical factors: personnel, dynamism, culture, size and criticality. The double-headed arrow presents agility or more traditional/plan-driven software development approach, depending on the selected numbers in the diagram.

Personnel axis shows the skills and experience of the people who are working on the project, agile methods tend to have a higher level of more experienced people, which in this case as a group we do not. Dynamism determines how dynamic is the environment, how simple/complex is the design, if the project has a high or a low rate changes per month, more changes – more agile. The culture axis shows which practices will be proved to be a better choice in a chaos thriving culture (agile) or in an order thriving culture (plan-driven). Size axis determines the number of personnel that are going to work on the project. Lastly, the criticality axis displays the complexity of the project/system development, again, less critical means more agile.

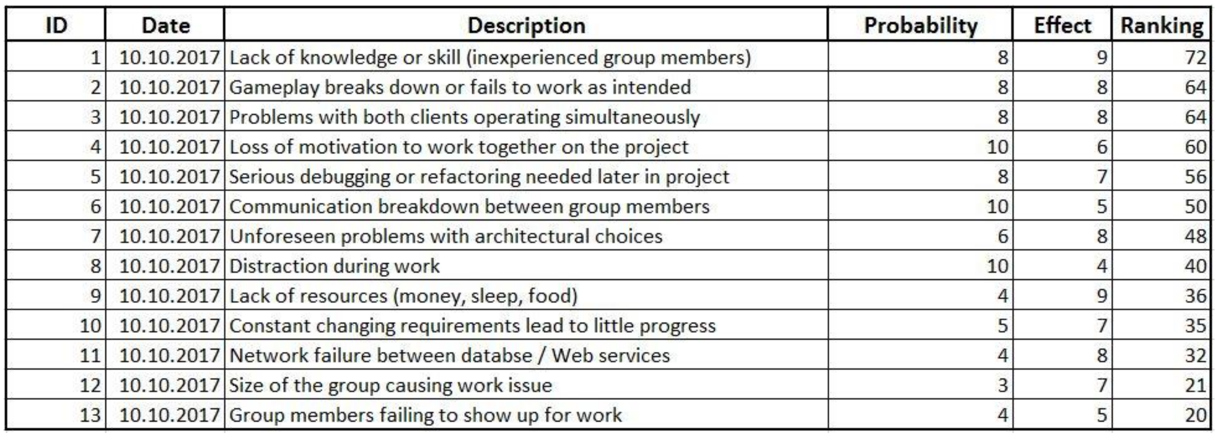
The reason for choosing Scrum and XP principles is mainly because we were introduced with this new and interesting software development concept in this semester. This approach seemed suitable for the project idea we have had in mind, simply because we were not creating a serious banking application, but a game. That being said, our game did not require plan-driven development where all the design and architecture are pre-determined, it is a simple quiz game and we strongly agreed to generate minimum documentation.

Scrum principles we used are: iterative development, self-organisation, process control, collaboration, value based prioritisation and time-boxing (which includes Sprints, Daily Stand-up Meetings, Sprint Planning Meetings, and Sprint Review Meetings).

The principles we used from XP include, but are not limited to pair-programming, test-driven development, collective ownership, coding standards, continuous integration, and values like communication, courage and simplicity.

Risk analysis

We have conducted a risk analysis to cover and possibly reduce the risks of all the internal and external factors that might affect the project performance and/or could result in some data loss.



ID	Date	Description	Probability	Effect	Ranking
1	10.10.2017	Lack of knowledge or skill (inexperienced group members)	8	9	72
2	10.10.2017	Gameplay breaks down or fails to work as intended	8	8	64
3	10.10.2017	Problems with both clients operating simultaneously	8	8	64
4	10.10.2017	Loss of motivation to work together on the project	10	6	60
5	10.10.2017	Serious debugging or refactoring needed later in project	8	7	56
6	10.10.2017	Communication breakdown between group members	10	5	50
7	10.10.2017	Unforeseen problems with architectural choices	6	8	48
8	10.10.2017	Distraction during work	10	4	40
9	10.10.2017	Lack of resources (money, sleep, food)	4	9	36
10	10.10.2017	Constant changing requirements lead to little progress	5	7	35
11	10.10.2017	Network failure between databse / Web services	4	8	32
12	10.10.2017	Size of the group causing work issue	3	7	21
13	10.10.2017	Group members failing to show up for work	4	5	20

Probability means how likely will the risk come true. Effect shows the outcome size of a risk.

The lack of knowledge and skills was ranked highest and because of that, we have decided to do pair-programming so our teammates can also learn new things. For our second risk we have used the value of courge and we have simplified the game because the more complicated it got, the bugs were getting harder and harder to fix. As far as loss of motivation goes, suprisingly, we have not really experienced it like on previous projects, since we have enforced collective code ownership and improved our communication.

All other risks did not really affect the project as much as we have anticipated in the beginning when performing the risk analysis.

Main principles in planning and quality management

Planning (Sprint 0)

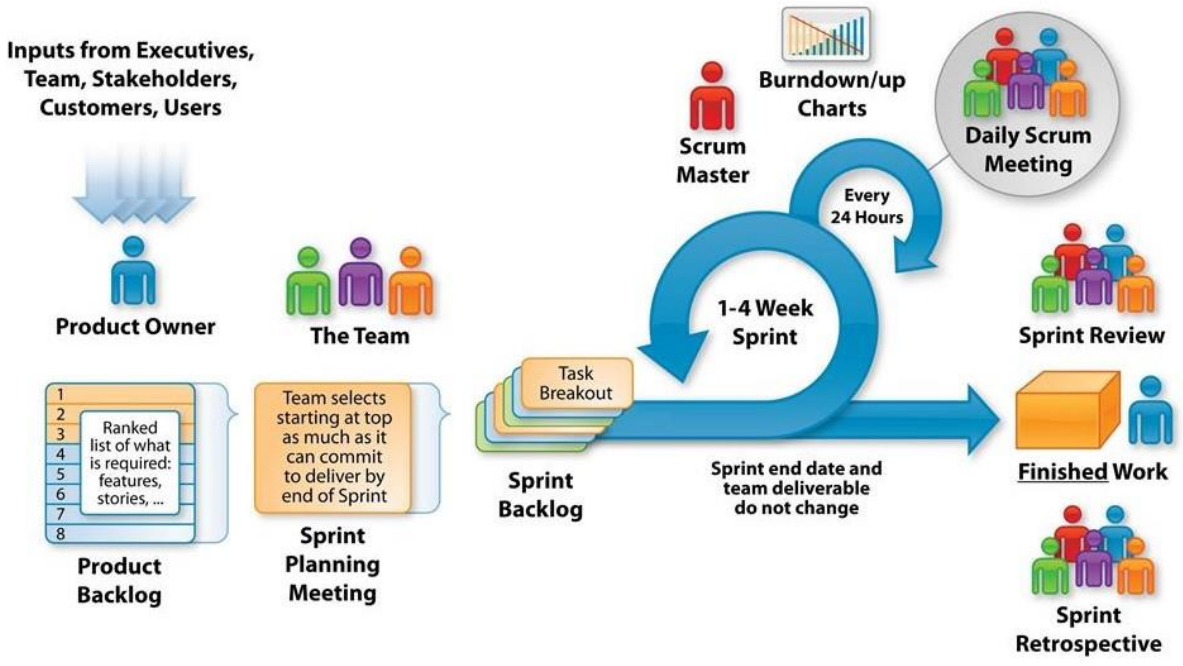
In our planning process we have used both principles from eXtreme programming and Scrum, and also the domain model which is considered being a part of plan-driven principles. We decided to include a Domain model given our lack of experienced developers and to help guide us in developing our system.

We considered identifying Spikes as a necessary part of our project planning. Their purpose is to perform research/analysis/exploration so that we could solve design/technical problems smoothly. Functional spikes helped us in planning and organising the project work, assigning Scrum roles, dividing the work, risk analysis.

Technical spikes were used for researching new technologies, finding different implementation solutions etc. We strongly agreee this was an extremely helpful decision, because we are not very experienced programmers.

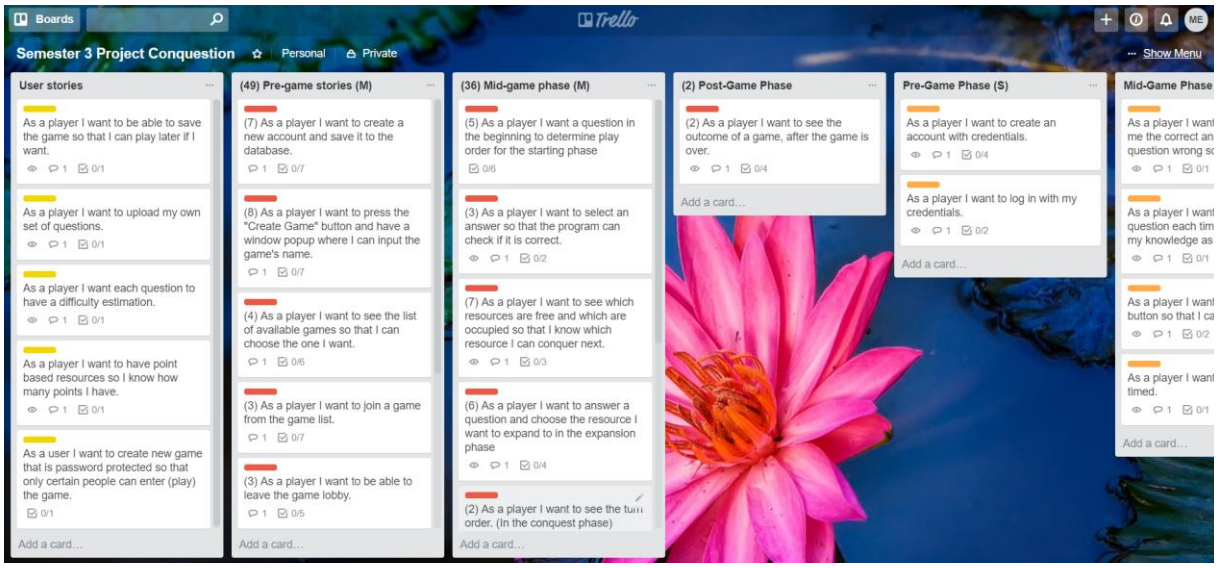
After deciding on an idea, we created personas and scenarios which would then help us in writing the User stories. The plan driven Domain model was created as an orientation, we ended up changing it as we simplified our project.

We have assigned roles of Product owner, Scrum Master and the Development team. For the purpose of this project we changed the roles after each sprint.


Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint end date and team deliverable do not change
Sprint Review
Finished Work
Sprint Retrospective

User stories were noted down on a card and each was estimated with a number of Story points that show how much time would it take to program it (our Story point has a value of 45 minutes); on the back of the cards we wrote the Acceptance criteria. Usually the Customer is creating Personas, Scenarios and User stories with Acceptance criteria, in our project the Product owner (who can also be a customer representative) was in charge for writing it. The Product owners assignment is to

write a Product backlog with all the prioritised features of the program. He/She decides which and how many User stories are going to be performed in each iteration/sprint.
Considering the project time limitation we have utilised the Moscow prioritarisation, and divided our User stories to Must, Should, Could or Won't have. Must have are the requirements with the highest importance (cannot deliver the main idea without it), Should have with medium importance meaning the features are still very important but the product can also do without them. Could and Won't have have the lowest priority, Could have are the desirable features that could be added if there is, for instance, more time.



Semester 3 Project Conquestion — Personal — Private

User stories
- As a player I want to be able to save the game so that I can play later if I want. 1 0/1
- As a player I want to upload my own set of questions. 1 0/1
- As a player I want each question to have a difficulty estimation. 1 0/1
- As a player I want to have point based resources so I know how many points I have. 1 0/1
- As a user I want to create new game that is password protected so that only certain people can enter (play) the game. 0/1

(49) Pre-game stories (M)
- (7) As a player I want to create a new account and save it to the database. 1 0/7
- (8) As a player I want to press the "Create Game" button and have a window popup where I can input the game's name. 1 0/7
- (4) As a player I want to see the list of available games so that I can choose the one I want. 1 0/6
- (3) As a player I want to join a game from the game list. 1 0/7
- (3) As a player I want to be able to leave the game lobby. 1 0/5

(36) Mid-game phase (M)
- (5) As a player I want a question in the beginning to determine play order for the starting phase 0/6
- (3) As a player I want to select an answer so that the program can check if it is correct. 1 0/2
- (7) As a player I want to see which resources are free and which are occupied so that I know which resource I can conquer next. 1 0/3
- (6) As a player I want to answer a question and choose the resource I want to expand to in the expansion phase 1 0/4
- (2) As a player I want to see the turn order. (In the conquest phase)

(2) Post-Game Phase
- (2) As a player I want to see the outcome of a game, after the game is over. 1 0/4

Pre-Game Phase (S)
- As a player I want to create an account with credentials. 1 0/4
- As a player I want to log in with my credentials. 1 0/2

Mid-Game Phase
- As a player I want me the correct an question wrong sc 1 0/1
- As a player I want question each tim my knowledge as 1 0/1
- As a player I want button so that I ca 1 0/2
- As a player I want timed. 1 0/1

As shown in a picture above, we digitalised our Product backlog with the help of Trello, for easier tracking of things that needed to be done. By using the Moscow prioritarisation we coloured the Must have User stories in red, Should have's in orange and Could have's in yellow; the Won't have's were not included. In addition to the prioritarisation, keeping in which order should the stories be executed, made our job significantly easier and more organised. We divided the User stories in three phases; Pre-game phase, Mid-game phase and Post-game phase. We chose the Pre-game phase to start with in Sprint 1.

Subsequently, the whole team meets up and plan how many User stories can be delivered in every sprint. The requirements are broken down into tasks, also called the Sprint backlog, main sprint goals are set and acceptance tests are created. Here comes in, the role of the Scrum master who is a project manager of sorts, leading and coaching the team, and planning the events of the project. He/She is making sure that everything goes as it was initially planned, and is trying for everyone to understand the principles of Scrum. The Scrum master is filtering the extern information for the development team, ensuring they are not distracted. Scrum master is also in charge of creating and updating the burndown chart for each iteration that shows the work that needed to be done and actual performance.

In the Scrum process, daily meetings lasting usually about 15 minutes, are held to review what has been done the previous day, what is planned for today and what problems could appear during work. After the daily meeting we have divided the programming pairs and started working. Our objective was to have working weeks not longer than 37 hours. We feel like daily meetings helped us a lot regarding our communication, even if we had small problems we have managed to solve them easily.

The length of the sprint is usually defined in advance. In theory, sprints duration can be anywhere between two to four weeks long. For our project, the sprints length was defined by the insitution and each sprint was around a week long.
There were some slight problems at the start, because we have underestimated our User stories, we did not consider possible problems that can occur and we did not consider testing so we had to change our existing User stories.

Following the XP principles we started our project with Test Driven Development and Unit tests.

We researched and set up our accounts with GitHub and GitBash for easier code transportation and remote work; testing it out with our model classes.

At the end of each Sprint we had a sprint review with the Product owner where we presented a basic working system, discussed which features have been implemented and created plans for next sprint. Sprint retrospectives should be done after each sprint. It is a chance for all the group members to share what they think it went well, what did not go so well and what could be done better in the next sprint.

Product owner was in charge for planning what User stories are going to be implemented in the next sprint (creating a Product backlog).

Simultaneously we wrote short notes for each sprint, explaining the progress of the whole development process. The planning Sprint 0 lasted for 11 days.

Quality management

Quality assurance and quality control

Software quality management is defined as a group of techniques, processes and standards that ensures the highest possible product quality, delivers the customers idea and meets the needs of end users. Quality assurance is a software management technique that provides a reliable system within the predetermined budget and time frame. In quality control we apply these techniques, and by comparing the final product with the quality attributes and standards, we verify its quality. The main difference between quality assurance and quality control is that QA is more oriented on the development process and preventing program defects while QC is more oriented on the product. Sometimes, the project managers are willing to risk the quality of the product just so they could deliver the product on time and not exceed the budget.

The functional requirements in software quality are used as an implementation guidance, because they are the core components of the program and they deliver the main idea. In other words, the goal of quality assurance is to check if the specified functional requirements fit with aimed quality attributes.

The software quality is not just dependent on functional requirements, it is also heavily reliant on its non-functional attributes, which are:

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

However, we are not able to implement all of these attributes, for instance, if we implement a high level of security, we risk the system becoming slower, and the systems performance is an important factor in software development; nobody wants to use a slow system. Failing to implement an important non-functional requirement can lead to an ineffective/useless system.

In plan-driven development, the documentation for quality assurance is quite extensive and the process of testing and validation is explained in high-level of detail. In agile development the quality assurance management is performed during the whole development process. Considerable time is spent on testing and prototyping. Code development is the central focus together with the team member's informal communication. The team is responsible for delivering a quality system, so all the possible problems, changes and/or quality matters are discussed, and the group as a collective makes final decisions.

In our project we ensure the quality assurance by enforcing the following:

- Defining User stories
  Writing down the core funcionalities of the system in the form of User stories, based of that the Acceptance criteria is written which then leads to creation of tests

- Collective ownership
  Each person in the group is responsible for delivering quality code, any team member can change any part of the code any time in the project if he/she encounters errors

- Pair-programming
  Two programmers on one computer; errors are reduced and higher quality code is produced

- Continous integration
  Daily integration of different developer's work, performing automated building tests and fixing bugs

- Test driven development
  Automated unit tests are formed before the code, when creating tests we are ensuring that our code won't be mistakenly damaged, the bugs are easier to find and fixed; all tests should pass on a certain functionality, verifying that it works before we release it

- Communication
  Everyday Scrum meetings, discussing problems and plans

- Code refactoring
  Making existing code more readable, removing duplicate code, changing the original design so the system meets the desired requirements

- Changes
  By choosing quality over quantity, we dropped some features that would take longer time to implement and by doing so we gained more time to focus on the core functionalities of the game and the security issues

- Pull requests before merging
  After completing a piece of code, the code is uploaded on GitHub and reviewed and merged by a different team member

Quality criteria and architecture

FURPS is an acronym for Functionality (capability, reusability and security), Usability (human factors, documentation and responsiveness), Reliability (failure frequency, recoverability, predictability and accuracy), Performance (speed, efficiency, resource consumption and throughput) and Supportability (testability, flexibility and install-ability), which is used for classifying software quality attributes.

We decided on the functionality in our Product backlog, but we also considered the non-functional requirements for our system (URPS).

Usability: factors important to the usability of our system included ease of use, understanding of various functions without explanation, and interaction with the system is pleasant to users.

Reliability: we decided that reliability was not a high priority for our system, while it is annoying to have a game session drop mid-play it is not life threatening nor part of vital systems day to day meaning a lower risk. However, we would still like the system to be reliable the majority of the time as a broken system will discourage use of our system.

Performance: this is somewhat important as the speed the system processes messages is important in a time sensitive game scenario. Lengthy delays retrieving information or receiving feedback can be very dissatisfying to user experience.

Supportability: supportability is not vital to our system as again this is not a critical system for daily use. However, if we would like to implement new features to our game and be able to fix the system if there are any issues we should design the system to allow for easy supportability.

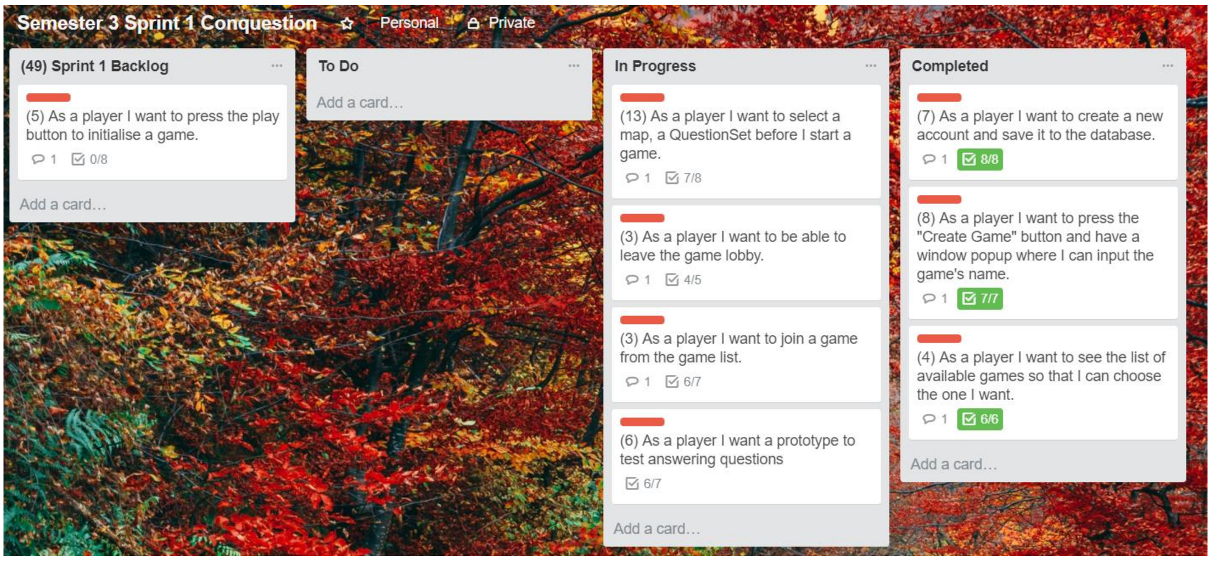
From our analysis we have identified some key non-functional requirements for our quiz game. Our system should have good usability in that learning how to use the software is very quick.

Performance and Reliability are also important to gameplay to create a smooth experience that doesn't feel broken or buggy for players. Finally supportability isn't vital however if we would like to add features in the future we should design the system to allow for scalability. We will have to consider this non-functional requirements in the design and deliberations for our system architecture.

Reflections on methods and their use in practice

Sprint 1

Our plans for the first sprint included implementation of User stories from the Pre-game phase. The goal was to create the beginning of the game like login screen, start screen and lobby screen. The role of the Scrum Master was carried out by Tamas and we chose Andreas to be our Product owner.



Semester 3 Sprint 1 Conquestion · Personal · Private

(49) Sprint 1 Backlog
- (5) As a player I want to press the play button to initialise a game. 1 · 0/8
- Add a card…

To Do
- Add a card…

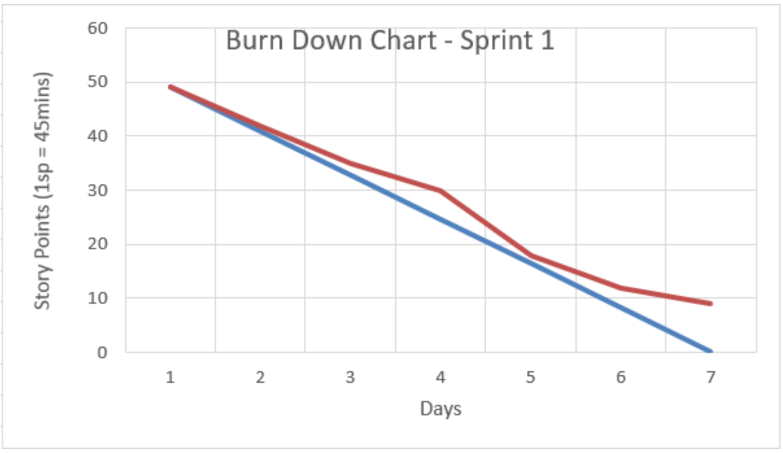
In Progress
- (13) As a player I want to select a map, a QuestionSet before I start a game. 1 · 7/8
- (3) As a player I want to be able to leave the game lobby. 1 · 4/5
- (3) As a player I want to join a game from the game list. 1 · 6/7
- (6) As a player I want a prototype to test answering questions 6/7
- Add a card…

Completed
- (7) As a player I want to create a new account and save it to the database. 1 · 8/8
- (8) As a player I want to press the "Create Game" button and have a window popup where I can input the game's name. 1 · 7/7
- (4) As a player I want to see the list of available games so that I can choose the one I want. 1 · 6/6
- Add a card…

The picture is showing our Sprint Backlog at the end of Sprint 1. In the beginning we have estimated a total of 49 story points. In planning phase we agreed that our working weeks will not be longer than 37 hours, and we had 7 days to complete Sprint 1.

Every day of the sprint we started off by having a short meeting, which did not last longer than 10 minutes. On the meeting we communicated plans for the current day, decided which programmers will work together in pairs, and split up the User stories for each pair. Our team consists of 5 members so we could have only 2 programming pairs with the fifth person helping both pairs.

The burnchart was updated daily by the Scrum master, at the end of the Sprint 1 it looked like this:



The blue line represents estimated velocity, in this case 49 story points and the red line represents our actual work. Having a pretty good start, some things did not go as planned. On the last day we had approximately 9 points left, which needed to be transfered to the next sprint.

At the end of the sprint we sat down to discuss the plans for the next sprint, we created a new Product backlog and broke down our User stories into smaller tasks. Even though it was Product owner's job to prioritise requirements, because we did not have a real customer representative we strongly felt it should be done as a group. We have estimated each task separately and delegated story points. Our goals for the next sprint were to have core game mechanics finished, to change some game logic and to implement some features at a later phase.

We organised the Sprint retrospective, and here are some things that we discussed:

What went well

- we regularly showed up on time each day
- held daily Stand-up meetings that lasted for about 5 mins in which we said all our expectations for the upcoming day and retrospective of a previous day, what problems did we run into (if any)
- we did pair-programming
- communicated well and did not have any major group conflicts

What did not go well

- we are not sure if we did the testing properly
- we were struggling with task solving from our Sprint backlog
- struggled with GitHub merge conflicts
- underestimated some User stories and their Story points
- we accidentally lost some data
- we put too much emphasis on the UI instead of on the back-end development

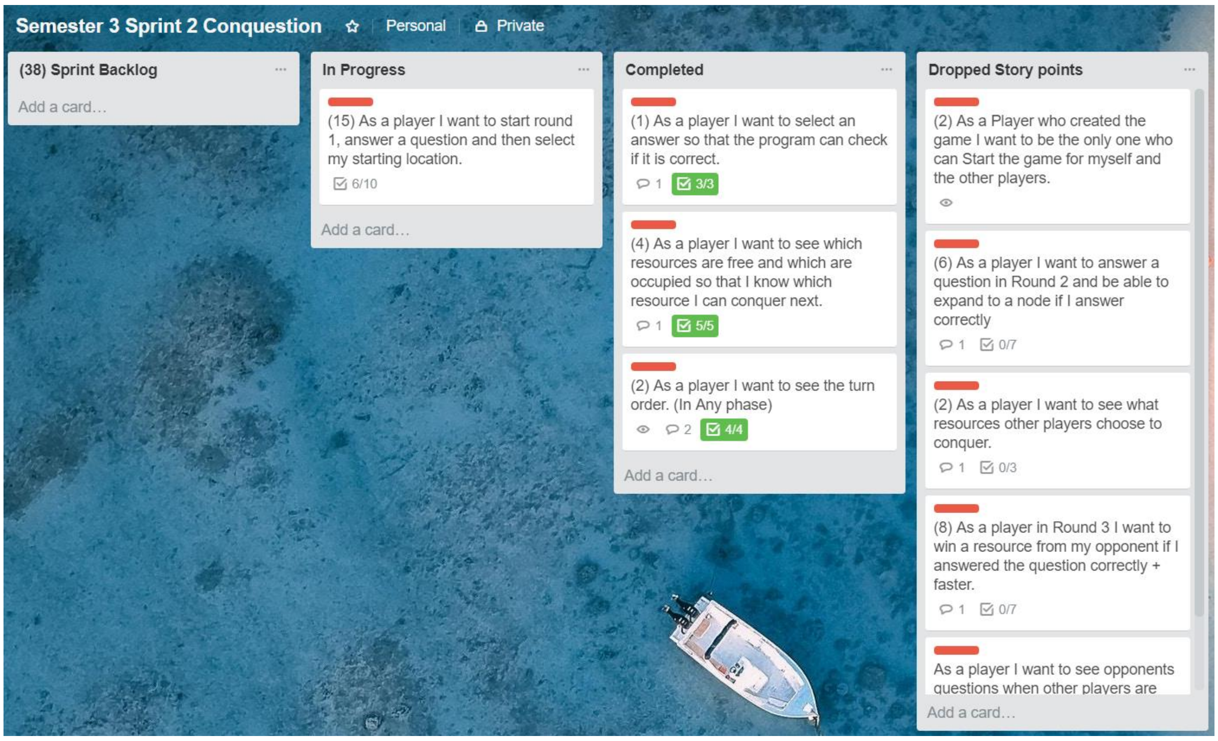
What can be improved

- do not put so much emphasis on Test driven development
- focus more on System tests
- try to explain more in pair programming
- let weaker programmers code with the supervision of better programmers
- enforcing working on the weekends if we are missing sprint days

All in all, we managed to finish 40 story points in Sprint 1 and managed to push out a basic working system. We have encountered several problems trying to carry out the XP principles. Test driven development did not go so well for us, because we were not sure what is the proper way of doing Unit tests and it took a lot of our time trying to create proper useful tests, therefore we have agreed to do system tests instead. Estimation of User stories and tasks was wrong, we have not included several things, like testing and research.

We felt that pair-programming went mostly well, although some minor adjustments could be done. The 37-hour week has been slightly reduced to approximately a 30-hour week.

Sprint 2


Semester 3 Sprint 2 Conquestion ☆ Personal Private

(38) Sprint Backlog

Add a card…

In Progress

- (15) As a player I want to start round 1, answer a question and then select my starting location. ✓ 6/10

Add a card…

Completed

- (1) As a player I want to select an answer so that the program can check if it is correct. 1 ✓ 3/3
- (4) As a player I want to see which resources are free and which are occupied so that I know which resource I can conquer next. 1 ✓ 5/5
- (2) As a player I want to see the turn order. (In Any phase) 2 ✓ 4/4

Add a card…

Dropped Story points

- (2) As a Player who created the game I want to be the only one who can Start the game for myself and the other players.
- (6) As a player I want to answer a question in Round 2 and be able to expand to a node if I answer correctly 1 ✓ 0/7
- (2) As a player I want to see what resources other players choose to conquer. 1 ✓ 0/3
- (8) As a player in Round 3 I want to win a resource from my opponent if I answered the question correctly + faster. 1 ✓ 0/7
- As a player I want to see opponents questions when other players are

Add a card…

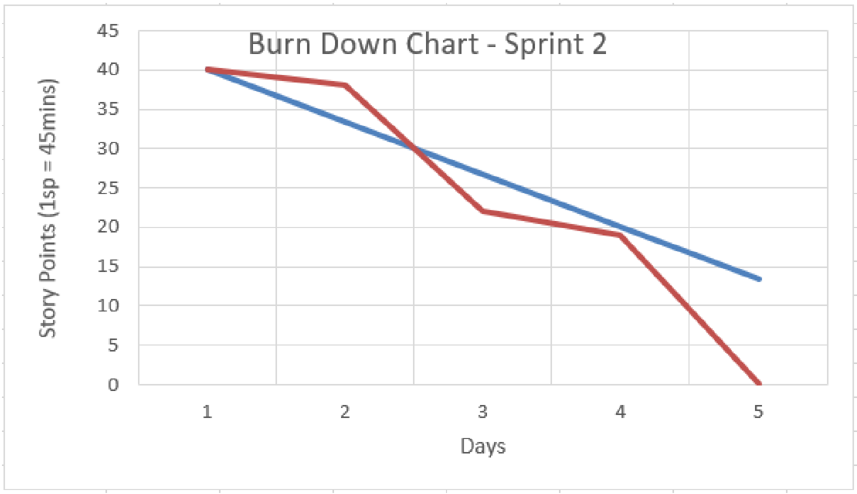
This is the Sprint backlog for our second sprint. The Scrum master was Madalina and Sangey was the Product owner. The duration of Sprint 2 was 5 days, originally 4 days but we were willing to work an extra day on the weekend. Team's goal was to finish 38 story points.

We had spend some time refactoring some of the core functionalities like the Start game button, that only the creator of the game can start the game. It was a functionality we had not think of before and spent two extra story points on it.

The countdown timer and function which determines order of players has been implemented and system tests were performed for each new functionality. Player order and Map selection functionality has been fully implemented, unfortunately it took way more time than we have anticipated. The original logic idea got really complicated and debugging got problematic.

As a group we have decided that the first version of the game might be too complicated to implement after all, so we agreed to make a simpler version while continued working on the more complicated version at the same time. Our domain model has been simplified, we are keeping only the Starting round, without the other two and we are dropping the map, refactored the necessary code. The next few days, we have continued to work on the simpler version.

The Scrum master tried to do the best possible job by motivating us and trying to put us on the right track when we have wandered off. Programming pairs stayed the same. The hardest parts of trouble-shooting was done by three people, but eventually we have realised it is not working out.



We have managed to finish only 21 story point in Sprint 2. The reason for that is the application of the mid-game phase functionalities, which has become extremely difficult. The only way to get back on track was to drop the rest of the story points.

The last sprint day was reserved for planning of the next sprint, breaking down the requirements and creating the Product backlog, creating the Sprint backlog and reviewing all the completed things in the Sprint retrospective.

Team's goals for next sprint will be revolving around changes of the User stories, meaning we have decided to create some new and update some old ones. Other goals are: implementing the Log in system together with security, finishing up the game logic and creating end game conditions, refactor our client using an authentication service and properly handling concurrency and transactions.

We gave our opinions in the Sprint retrospective as follows:

What went well

- we regularly showed up on time each day
- daily Stand-up meetings
- we did pair-programming
- did not have any major group conflicts
- other team members tried to help with bug fixes

What did not go well

- we were struggling with implementing some functionalities
- the pair-programming became one man's show
- bug solving (major errors)
- wrong estimation of requirements
- dropping User stories

What can be improved

- simpler design release
- better communication amongst programmers

- let weaker programmers code more often so they could learn more
- complete all the other User stories accordingly

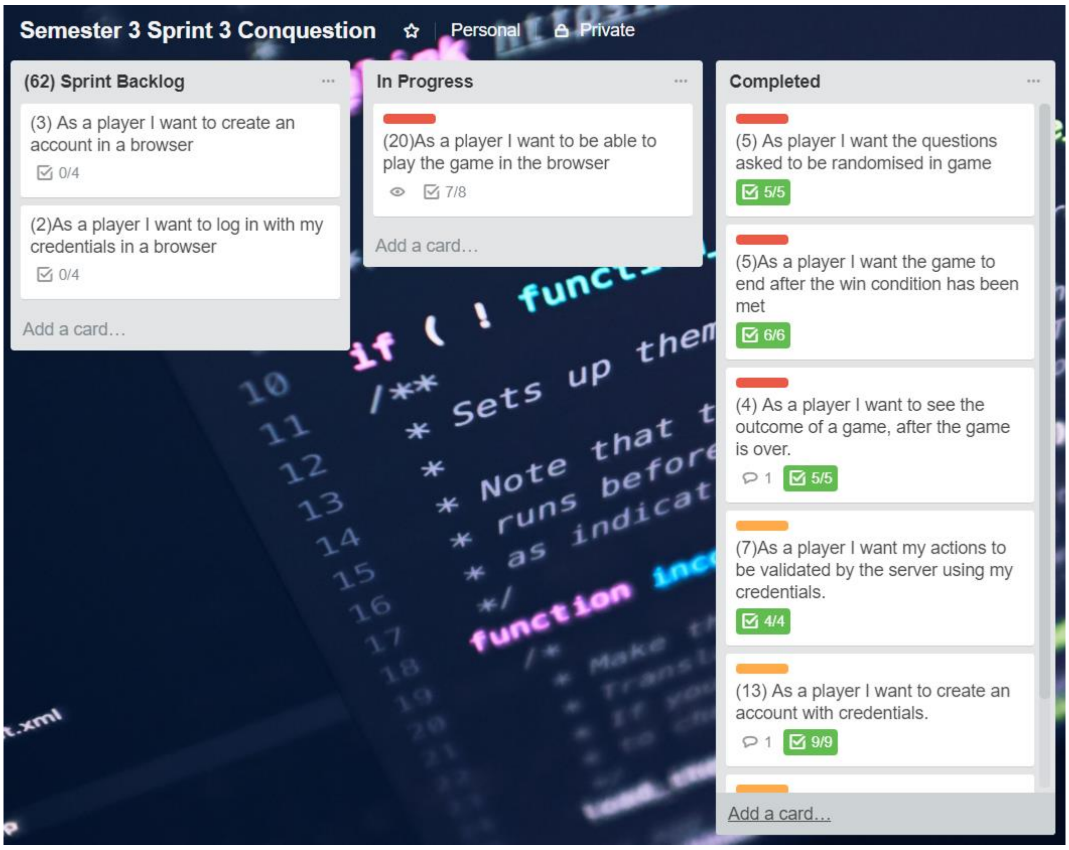
The biggest obstacle we came across is programming inexperience and/or lack of knowledge. The pair-programming turned into waiting for help from more experienced programmers who also struggled and trying to figure out their own bugs. We have managed to release a piece of the simpler version, even though it was not close to being completed. By simplifying our design we had to refactor a large part of the code, which took some time and it was done by more experienced programmers. The values we respected most are communication, simplicity and courage, because it was a hard decision to simplify and change original project plan for some members of the team.

Sprint 3

This was by far our most productive sprint with 50 completed story points in only 4 days. The Scrum master role was performed by Mirjana and Product owner by Tamas.

As planned in the previous sprint, we had to change some of our existing stories and add some new ones, because we were confident we are able to implement more features (register and log in and security in web and dedicated client), now that the game has been simplified.

In the first few days we tried to implement the Web client for the core game functionalities, game security as an important part of our system and to finish up the main features in our dedicated client (randomised questions, game ending and outcome of a game). The highest number of story points was knocked off in the first day. We changed the programming pairs and they all did very well (with Sangey being a supervisor, since he is the most experienced out of all of us).


Semester 3 Sprint 3 Conquestion ☆ Personal Private

(62) Sprint Backlog
- (3) As a player I want to create an account in a browser — 0/4
- (2)As a player I want to log in with my credentials in a browser — 0/4
- Add a card…

In Progress
- (20)As a player I want to be able to play the game in the browser — 7/8
- Add a card…

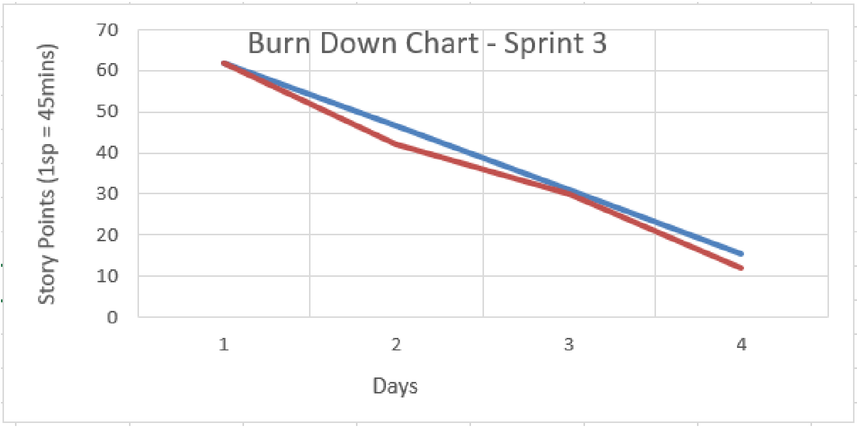
Completed
- (5) As a player I want the questions asked to be randomised in game — 5/5
- (5)As a player I want the game to end after the win condition has been met — 6/6
- (4) As a player I want to see the outcome of a game, after the game is over. — 1 — 5/5
- (7)As a player I want my actions to be validated by the server using my credentials. — 4/4
- (13) As a player I want to create an account with credentials. — 1 — 9/9
- Add a card…

We have implemented the Log in and some other core functionalities for the Web client, Create and Join game have the biggest number of story points, therefore it is considered to be an epic. However, we did not split the epic into smaller User stories, we split it up in tasks instead like all the other User stories when we were creating the Product backlog during Sprint 2.

Trying to implement final features in the Web client and security programming pairs hit a wall, thus some research was necessary to perform during the weekend. Other implementation like End game screen was also harder to implement than expected.

On the third day we tried to do as much refactoring as we can, finish with security implementation and end game in the dedicated client. Merging problems poped up again, but we figured it out quickly.

The last day was reserved for solving existing problems, implementing transactions and concurrency. The biggest issues we had was with Entity framework and refactoring was the most important item on our agenda. A few members that finished their tasks started writing the reports.



At the end of this sprint we did not manage to finish 12 story points and they were transfered to Sprint 4. Overall, we considered doing very well in Sprint 3, especially because we were slightly ahead of schedule.

Again, we sat down and discussed the sprint work flow. Plans for Sprint 4 were noted and a few new User stories were added and broken down into tasks, Sprint and Product backlog were created. Targets for next sprint: creating Unit tests to satisfy the quality assurance principles, report writing, project merging and refactoring.

Sprint retrospective:

What went well

- we regularly showed up on time each day
- daily Stand-up meetings
- the programming pairs worked better than usual
- we had effective communication
- we all worked equally on separate things
- implemented extra features like user registration on both clients, security and concurrency

What did not go well

- underestimated research spikes
- quality suffered
- bug solving and merging pieces of code
- wrong estimation of User stories

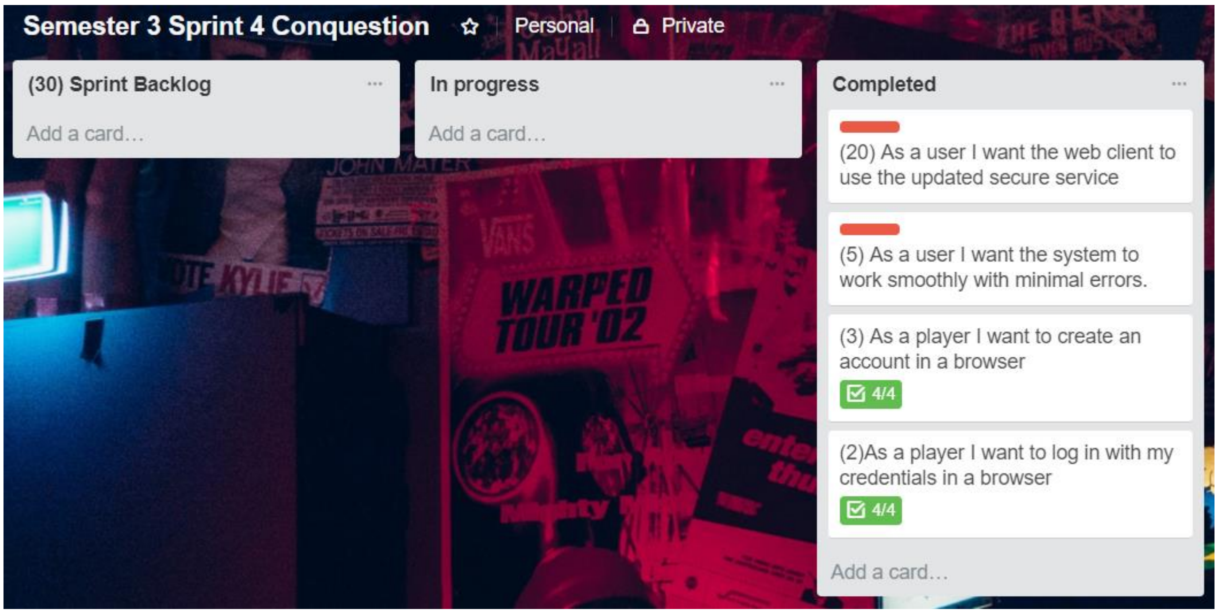
What can be improved

- quality assurance
- make unit tests
- complete all User stories
- better research for the report

Refactoring was a painful process together with security and Web client implementation, but we have managed to do it well within the coding standard guidelines.

Sprint 4

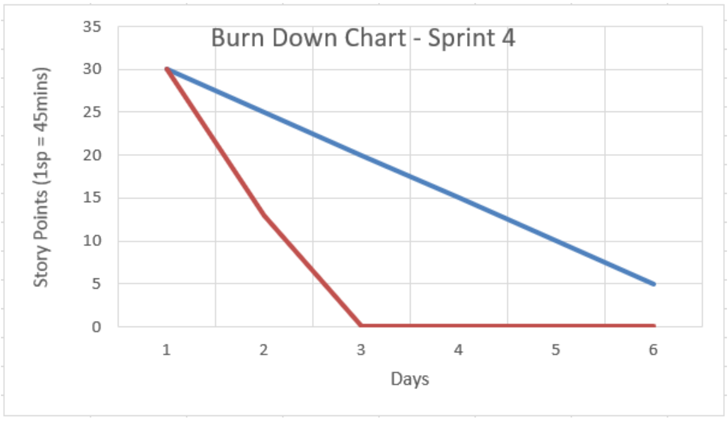
Scrum master of this last sprint was Andreas and Madalina was the Product owner.



(30) Sprint Backlog	In progress	Completed
		(20) As a user I want the web client to use the updated secure service
		(5) As a user I want the system to work smoothly with minimal errors.
		(3) As a player I want to create an account in a browser 4/4
		(2)As a player I want to log in with my credentials in a browser 4/4

Our final sprint had 30 story points, the points that were left from the previous sprint were moved to and epic. We had 6 days to complete our system and make the final changes.

We completed implementation of all the User stories in 3 days and the rest of the sprint will be spent on heavy refactoring and testing.



According to the final burndown chart we were way ahead of schedule, and that is the reason we used the time left to polish up the system.

At this point we were happy how our system looked like and felt we could utilise our time more productively by doing some quality control and code review of other programming pairs.

Sprint 4 went by pretty fast, and we were able to release a fully functioning system. However, some minor refactoring changes needs to be done to make several clients run at the same time.

Conclusion

By being introduced with plan-driven development in our first year we have learned how to do a detailed process when developing a new system, and now, we were introduced with agile methodology.

While there are many differences between plan-driven and agile development, as we have researched and proved through concrete examples of our project and have explained in this report, it is possible to use some concepts together. Some of the plan-driven methodologies like Waterfall model demonstrated to be inefficient and unable to adapt to constant market changes and new technologies. Both methodologies have their pros and cons but each company is using the software development practices that fits them best.

Personally, we find that Agile development is much more interesting and simpler to execute than the plan-driven models. We learned quite a lot through this process and agile principles. Everyday communication was a chance to discuss important subjects from our project or simply, to resolve small issues we have had within the group.

We liked the fact we were able to add and/or remove features from our system, without needing to change the whole project, like we would be forced to do in plan-driven development. There was no need for extensive documentation and drawing diagrams for each step of the process. The atmosphere was pretty relaxed most of the time, work was flexible and we all tried to contribute equally.

Scrum and eXtreme programming practices opened new horizons and we feel more confident with our system development planning and programming skills, and it has also helped us connect more with other team members.

The only possible downside is that we are not yet experienced enough with agile development, therefore we may have made a lot of mistakes in this project. Luckily, we were able to overcome some of them.

Like in everything we do, knowledge comes with experience, and the more we use and work with Agile development the better systems we will be able to produce.

References

1) Ian Sommerville – Software Engineering 10th edition, 2016.
2) Craig Larman – Applying UML and patterns, 2005.
3) Mike Cohn – User stories applied, 2004.
4) Henrik Kniberg – Scrum and XP from the trenches 2nd edition, 2015.
5) Henrik Kniberg & Mattias Skarin - Kanban and Scrum - making the most of both, 2010.
6) https://airbrake.io/blog/sdlc/v-model
7) https://www.scrumalliance.org/community/articles/2014/april/plan-driven-versus-value-driven-planning
8) https://www.scrumstudy.com/whyscrum/scrum-principles
9) http://ptgmedia.pearsoncmg.com/images/9780321186126/samplepages/0321186125.pdf
10) https://www.agilebusiness.org/content/moscow-prioritisation-0
11) https://www.scrumalliance.org/why-scrum/scrum-guide
12) http://edelalon.com/blog/2017/09/scrum/
13) http://www.extremeprogramming.org/rules/pair.html
14) http://www.extremeprogramming.org/rules/unittests.html
15) https://github.com/defuse/password-hashing/tree/a00bdf9b6d7861bff2fb9ea95de4bd209b3252fe
16) https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/
17) http://www.extremeprogramming.org/values.html