

University College of Northern Denmark

IT-programme

AP Degree in Computer Science

Dmaj0919

SOLVR.ONLINE

SYSTEM DEVELOPMENT

Repository:

Maros Cuninka, Sebastian Holesz, Samuel Horacek, Martin Hotka, Ioan-Sebastian Voinea

Aalborg, 21. 12. 2020



University College of Northern Denmark

IT-programme

AP Degree in Computer Science

**Class:** Dmaj0919

**Title:** SOLVR.ONLINE – SYSTEM DEVELOPMENT

**Abstract:**

**Project participants:**

Maros Cuninka

Sebastian Holesz

Samuel Horacek

Martin Hotka

Ioan-Sebastian Voinea

**Supervisor**:

Nadeem Iftikhar

**Submission date:** 21. 12. 2020

Table of Contents

[**INTRODUCTION** 4](#_Toc59045759)

[**1.** **Product Vision** 4](#_Toc59045760)

[**2.** **Business model canvas** 4](#_Toc59045761)

[**3.** **Prototyping (Personas)** 5](#_Toc59045762)

[**4.** **Mock-ups** 6](#_Toc59045763)

[**5.** **Methodology** 6](#_Toc59045764)

[**5.1.** **Theoretical comparision of plan-driven and agile development** 6](#_Toc59045765)

[**5.2.** **Our approach** 7](#_Toc59045766)

[**5.3.** **How we used XP** 8](#_Toc59045767)

[**5.4.** **How we used SCRUM** 14](#_Toc59045768)

[**5.5.** **Our conclusion** 14](#_Toc59045769)

[**6.** **Planning & Estimation** 14](#_Toc59045770)

[**7.** **Quality assurance & Quality management** 17](#_Toc59045771)

[**7.1.** **Quality criteria and architecture** 17](#_Toc59045772)

[**8.** **Risk analysis** 17](#_Toc59045773)

[**9.** **Requirements definition** 19](#_Toc59045774)

[**9.1.** **Functional** 19](#_Toc59045775)

[**9.2.** **Non-functional** 19](#_Toc59045776)

[**10.** **Configuration management** 19](#_Toc59045777)

[**10.1.** **Version control** 19](#_Toc59045778)

[**CONCLUSION** 19](#_Toc59045779)

[**BIBLIOGRAPHY** 19](#_Toc59045780)

[**APPENDIX** 20](#_Toc59045781)

# **INTRODUCTION**

Text…

# **Product Vision**

When we, as a group, were faced with a decision to come up with a unique idea that could change an industry and could be used for our project, we had a range of ideas. The technical requirements laid out by the curriculum had to be considered, as well as an applicability of the solution to the real word. Thus, we allowed ourselves some time to innovate and we narrowed the ideas just a top 3 from which we voted the winner. Most of the ideas were related to e-shops or customer-to-customer marketplaces with which we decided to go. None of has ever worked on a C2C project ever before, so we took it as a challenge.

Product vision is something that shapes the project itself – it describes the overall mission of the product. For us it means that each member has something to fall back on, something forms the end-goal of the entire process. This is our statement:

“The ultimate marketplace where your homework finds its solver”.

It has to be noted that before coming up with our vision, we focused on the products name. We felt that it needs to be striking and that its domain should be available for us to buy, which we did straight-away. After some changes we ended up with Solvr.online.

# **Business model canvas**

After we came up with an idea what product we want to develop we decided to find out if it is even possible and would make sense for us to even start with it. For this reason and for better understanding of the entire idea we created a business model canvas which helped us to visualize the core business concept.

As you can see in Appendix A – Business Model Canvas, on the right side we have our internal factors we considered as relevant for us – Key Partners, Activities and Resources. In the middle we have Value Proposition, which describes what values we want to deliver and on the right side there are external factors, such as Customers and Market in general.

The main goal of the business model is to plan how a business intends to make money and business model canvas makes it easier for stakeholders to understand it. As our project was not meant to be developed in a such way, we have not considered this to greater extent than what was described before.

# **Prototyping (Personas)**

In our case there are more ways how the customer could interact with our product. User of the product is either a poster who needs an assignment to be solved or a solver who is going through posted assignments and looking for the ones he can solve.

According to this we created personas which are fictional characterizations of users that represent real groups of potential customers. Creating these Personas helped us to better understand how the user would interact with our product from a third view perspective and what group of customers we are targeting. We would describe them by name, photo, age, education, job, interests/hobbies.

Referring suddenly to a Persona with a Name instead of just generic user (poster/solver), gave us a clearer vision of who we are developing the product for. It helped us to keep realistic ideas of users. We would commonly say that Muhammad (reference to persona 1) would use the product this way because etc. Although Jan (reference to persona 2) would use the product another way. We would also think about the motivation behind the personas using the product. Either a busy elementary student who does not have enough time to solve all his homework or a high school teacher who wants to earn money on the side by solving assignments.

Text

Description automatically generated

Figure 1 Persona

# **Mock-ups**

The first rule of UI that we applied when creating the mock-ups is the principle of common region. It defines most of our screens, especially when the user is creating and solving an assignment. The two pages mentioned above contain forms that are separated in regions, each region with its own defined scope. The separation of those regions makes the form clearer to the user and what he/she must fill in.

The second principle that guided our mockup is the principle of similarity and simplicity of target. Our business model is based on commission, when a user posts an assignment, we take a percentage of the value from the credits that are given with that assignment. Therefore, the easiest action on the platform should be posting an assignment. The “Post Assignment” and “Publish” buttons are separated by position, color, and shape from everything else on the page, making them different and easy to find, besides that, “Post Assignment” is placed on the navigation bar, being always available to the user.

On both pages, Create an Assignment and Display an Assignment, the Deadline, Credits, Academic Level, and Subject have similar functionality. To make it easier for the user to understand that we have combined all of those in a separate region called Details.

# **Methodology**

## **Theoretical comparision of plan-driven and agile development**

In order to develop a system or a product, one has to look at the way the entire development process will look like. How will he ensure that the system will not have to be remade in the middle of the development process and ultimately how will he meet customer’s deadlines & expectations.

There are two main approaches: a more traditional, plan-driven development & more dynamic, agile development. The choice often depends on the internal circumstances (such as budget, team size, time to market, security, and reliability of the product, which can be crucial for a government system, etc.).

It has to be noted that both methodologies have one common attribute – they are both iterative – meaning that the development process is split into even parts, each building on the previous with an option to change more dynamically. This concept is called “incremental development”.

Plan-driven development tries to eliminate all the uncertainty and risks as early as possible. Overall, the process is very predictive, as well as thoroughly documented, meaning that it is quite formal and does not leave much space to change requirements. The process is what matters the most. This development method is based around development stages, with each having a certain output at its end. The most prominent plan-driven development framework is Unified process (UP).

On the other hand, agile development is, as the name suggests, much more about agility and dynamicity. It tries not to predict as much, but rather adapt to change, because it assumes, we cannot get things right up front. In contrast to plan-driven, it is all about the value, not so much the process. The formality is also much lower, meaning the number of ceremonies is lowered to a minimum. What matters is teamwork, strong collaboration with customer and flexibility.

Agile has many frameworks which are used all over the world. The ones that we will focus on are Scrum, Kanban & Extreme programming (XP). Scrum puts more focus on its artifacts and ceremonies. Development team uses artifacts such as product backlog, with user stories to be implemented, sprint backlog with user stories for a specific sprint (iteration). Scrum also has three formal roles (scrum master, product owner, development team), whereas Kanban has none. Kanban also omits most of the Scrum ceremonies and focuses purely on visualizing work on its Kanban board.

XP is the only purely software development framework. It is described in its four values – communication, simplicity, courage and feedback and it has twelve principles. In short, it focuses on frequent releases, programming features only needed at each moment, coding in pairs and constant refactoring.

## **Our approach**

During this project, we have tried out several agile methodologies, practices, implemented many values from eXtreme Programming or SCRUM, but the core of our processes was guided by what is called the “Agile manifesto”. We have followed the 4 point of the bespoken manifesto every day, from sprint 0 to the final day of our project. To show you how we implemented them in our project and everyday developer life, we will present them one by one.

**Individuals and interactions over processes and tools**

Our team is all about communication. We stay in touch at all times, we report to each other our steps, progress and decisions. Like this, we are able to keep the team working perfectly on both individual and group level. We have stepped away from long documentation, complex diagrams and redundant protocols.

**Working software over comprehensive documentation**

We realize that each team has a time budget, which is all they get to develop a piece of software. In our case we had 5 weeks, which we decided to spend on developing working and comprehensible code, rather than spending many hours on writing documentation and thus losing much of the limited time budget

**Customer collaboration over contract negotiation**

This point may have not directly affected our project, or at least it may seem like that, but if we take a closer look, we can see that throughout the whole process we prioritized communication and discussion about features and functionality rather than needless bureaucracy

**Responding to change over following a plan**

Change. Change was the subtitle of this project. To count all the changes would be a laborious task and to say the least, we changed the whole architecture of our project 2 times, which meant redoing and refactoring hundreds of lines of codes and tens of extra working hours spent on it. But in the end, the courage to embrace change helped us to develop a product we were all satisfied with.

## **How we used XP**

During sprint 1, we took advantage of the methodology of Extreme programming (XP), its practices, values and procedures. The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels [1]. That means long sessions of code review, pair programming etc.

Our approach to XP

XP concentrates around 5 values and 12 practices [2]. We have tried all of them, but focused on only those that worked for us, or were applicable to our project. We pushed those that worked for us to the extreme as advised by the creators of the XP methodology. Lets go through them one by one and see how we implemented or embraced them.

**The 5 values of XP**

**Communication**

In our team, which has been together for longer than a year now, we value communication above everything else. We are not only teammates, we are also friends, housemates. We are in touch every single day and we like spending time together. It only feels natural to us to voice our concerns, talk about obstacles and problems and help each other with solving. It is not uncommon that our Teams calls take many hours (the longest being almost 14 hours). Even with the pandemic being hitting hard we have managed to use virtual channels to meet and work together. We always try to discuss our steps, work as a unit and just frankly, speak.

**Simplicity**

This may have been a struggle for us in the beginning. Simple design is an easy thing for advanced programmers and software developers, because it is easy for them to distinguish what is and what is not going to be needed. We have, nevertheless, tried to write our code as simple as possible, do not code “for the future” and only hunt for the functionality needed at the moment. In the end, we have all agreed that we have made an enormous progress in this regard, spending the final weeks on coding only the “right here, right now” functionality.

**Respect**

Respect, as understood by XP, is not only internal (within the team) but also external, as the developers should respect the client’s decisions and opinions, managers respect the developers and that they are the ones developing the system etc. In our case, the respect was only internal, but on a high level to say the least. We know what everybody is capable of. Each of us respects the other team members for what they bring to the table. Those are the fruits of our long-term cooperation.

**Feedback**

In the case of feedback, we received both internal (on developer-to-developer level) and external after every sprint on the sprint review meeting, when we were asked questions not only by our teachers and supervisors, but also other teams. We have always presented the whole picture and as much as possible as soon as possible. We have felt like the earlier we are able to get complex feedback, the more agile we can be.

**Courage**

Courage is an important part of all agile methodologies. It is essential that the teams are not afraid of changing the direction of where the project is going. We think that our actions and decisions during the project demonstrate how courageous we were. It is not an easy decision to change everything in your code, leave no stone unturned. And we did so twice.

**The 12 principles of XP**

The following 12 values are ordered by how much we have embraced them, the ones that we embraced the most being on the top of the list. You can also see a map of how they were connected within our project.

**Refactoring**

Refactoring was a second nature to us. We have refactored from the smallest bits of codes such as local variable names (from id to userId to be more understandable) to complex conditional logic, for example the logic of who can access certain API endpoints, from everybody, through everybody who is logged in, then everybody who is logged in and is a customer all the way to only the person who is logged in, is a customer and posted the specific assignment. The process of refactoring has been embraced at all times and was generally considered as the most important one.

**Pair Programming**

This is a practice we followed even in previous projects. We know the advantages, but also the disadvantages of it. During the first two sprints, we have programmed in 2 groups, one consisting of 2 and one of 3 members. This was very helpful in the early stages of the project when we all needed to set our minds on how we want the code structure to look and how we want the code standards to be implemented. In the later stages, not every coding session was done in pairs, but at least 75% were.

**Collective Ownership**

Collective ownership means that every developer can change whatever they desire to, even though it was not them writing the original code. We have embraced this practice more and more as the project was progressing. At the end, all developers were refactoring and changing code if they spotted a mistake not thinking if that specific line of code was “theirs”.

**37-Hour week (regular working schedule)**

We have tried to keep the working schedule as regular as possible. Every day at 9:30 AM, but sometimes at 9 AM sharp, we met for a daily standup, where we discussed the necessary matters, later proceeding to the coding itself. At the beginning, we have agreed on 30 working hours per week, but the number of hours was raising with every sprint, reaching just a little above 200 for the last one. The burnout did not come as we all saw the end of the project approaching.

**Coding Standards**

We have followed the general C# coding standards [3] when writing our code, if we slipped at any time, we always made sure that we refactor the code later. This way we kept the code clear to everybody.

**Metaphor**

Metaphor is applying words, labels, tags or stories to various elements or chunks within the programming process [4]. We have applied such words to our system to make it easier for us to talk about it. For example, the user, who is a customer and creates an assignment, becomes a Poster, whereas the user who is a customer and solves that assignment, becomes a Solver. This is only applicable with regards to that specific assignment and the roles can be therefore easily switched when talking about a different assignment.

**Simple Design**

As mentioned before, when talking about the value of Simplicity, not coding ahead was a bit of a struggle for us in the beginning, but we have managed to reduce this not-agile practice by the end of the project. We have also decided not to focus on the diagrams too much, and only use them in the inception stage of the project, when we were deciding on how we are going to approach it. We also strived for no code duplication, by putting the repeating code into separate methods.

**Test-Driven Development (TDD)**

During sprint 1, we have tried out TDD as it is both one of the most praised and notorious practices presented by XP. In our case, we used it during the stage, when the foundations of our system were laid, and we could see its advantage of being certain at all time that your code is tested and also simple, as the TDD helps you to write simple code. Nevertheless, the tests we have written became obsolete as soon as we changed the architecture for the first time. We have decided that TDD created too much of overhead and was too hard for us to keep up with it.

**Continuous Integration**

When it came to continuous integration, we have tried to follow it as much as it was possible in our small, artificial environment. We did not have any releases planned except for the hand in in the second half of December, and we also did not have any production or testing servers. We at least did the following things:

- Branched our project in a way that main is only for tested and ready-to-show code, whereas the development (and branches that were branched from development) are meant for the implementation of new features

- We did not have a “clean computer” just for daily builds and integration tests, but we at least implemented gitignore file to avoid the issue of different packages and frameworks installed on different computers, which are usually the cause for the most common explanation of programming issues: “but it works on my computer”

- Since our project was not so big in its size, we were able to clean and rebuild often, always seeing if there are no build errors

- The feedback cycle (Develop test case 🡪 code 🡪 integrate 🡪 test) consisted of only the last 3 steps, because we did not develop a test case for every piece of code, we were implementing

**On-Site Customer**

We did not have an on-site customer. In our case it was all about the team, and we were therefore getting feedback about the implemented functionality only from each other.

**Planning Game**

The main goal of a planning game is to exchange the information between the customer and the development team. Customer has information about the value and developers about the cost. This was different in our case, because we did not have on-site customer as was mentioned before. Thus, the development team provides all the information about the value of project with cost calculation as well. Planning game consists of 3 phases.

In the Exploration phase, we wrote down short user stories (more about them explained later), then we prioritized, and the last step was to estimate. If we could not estimate one user story properly, was it too difficult for us to estimate, we split this epic story into two separate ones. For example, in user story Log in, we had task Log in using third-party APIs. Also because of time constraints and the difficulties to estimate, we decided to create a distinct user story.

In the Commitment phase, customer chooses the content of the first release. We would consider release planning, but we would need more time, somewhere around sprint 6-7. For us, the main focus was to have working code after each sprint with hand-in deadline. This phase was little bit different from what XP dictates in our case.

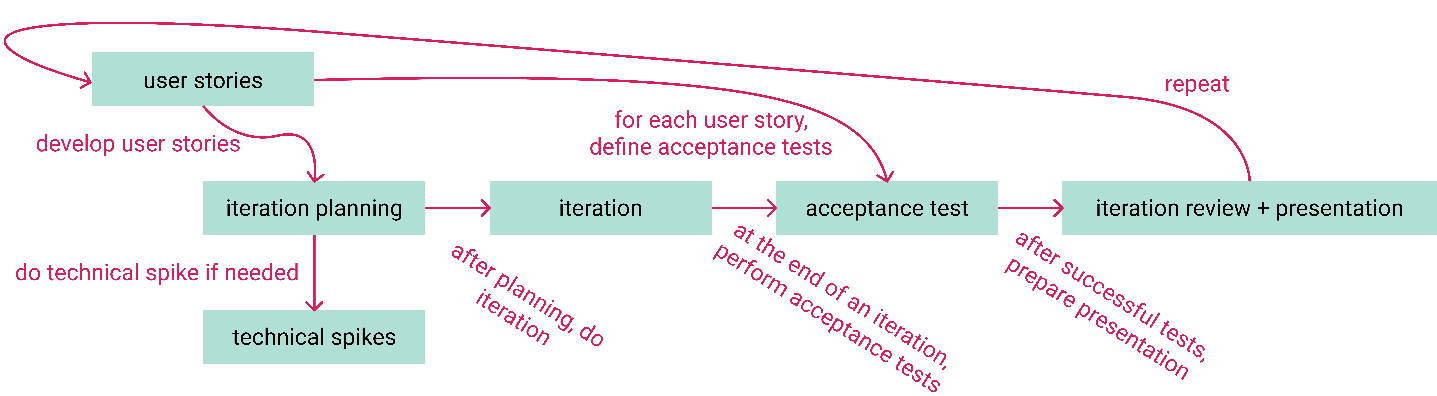
Last phase is the Steering phase. The idea of this phase is to have possibility to change accordingly to customer needs (to our need for us). We chose what to do in the following iteration and divided selected user stories into tasks. Few times we ended up that we had to reconsidered shifting some tasks or user stories into next iteration.

**Small Releases**

We did not do any small releases. During the whole project, we were developing what one could call an MVP (minimal value product). It would not make sense to make any releases in the meantime, as the system would not work properly in the production as is.

Our iteration timeline

Each of our iterations (or sprints if we are talking in SCRUM terms) consisted of couple of common steps and processes. Below, you can see the figure XXX, where you are able to see those steps.



1. Developing user stories

At the beginning of every iteration, we have developed our user stories. The process of creating those user stories is described in a greater detail later in this report. We have mainly focused on the 3 C’s and the INVEST values.

1. Iteration planning

We planned every iteration, also estimating how many user stories (in story points) are we able to develop during the next iteration. You will also be able to read about our estimation later in the report. We have mainly focused on the estimation techniques of analogy, and breakdown and calculation.

1. (Optional) technical spikes

Sometimes, there was a need for a spike, especially when working with technologies with which we had no previous experience. Those spikes, which result we usually threw away, were a helpful tool what brought us closer to understanding some more complex technologies, such as Asp.Net Identity, JWT or others.

1. Iteration

During this, in time the longest, step of all, we spent several days on developing the defined user stories.

1. Acceptance test

After each iteration, we have performed the acceptance tests we have created together with our user stories at the beginning of the iteration. Since there was no customer, who usually both defines and performs them, we had to do this ourselves. Successful acceptance tests meant fully implemented user stories and a green light to proceed to the next step.

1. Iteration review + presentation

When the acceptance test was concluded, on every Tuesday we had a presentation and iteration (or sprint) review, where we showed our progress, what we have developed, how did we improve our development process and answered questions to the outside world, in our case, the supervisors and other groups. This was a great check for us, as it had an enormous influence on many of our greatest decisions.

## **How we used SCRUM**

## **Our conclusion**

Looking back, we started off by following purely XP practices. It took us few days to get accustomed to their “extremeness”. After sprint 1, we chose to follow Scrum, whilst borrowing some of the attributes from XP, we thought will help our development process. In the end, the formal structure of the project was shaped by Scrum, but the coding itself by the coding principles of XP.

From our experience, if we are to compare unified process used in previous semesters compared to frameworks used in the project, the biggest difference could be felt at the start of the project. We were not spending considerable amounts of time on planning. We have assessed potential risks and worked up some crucial artifacts, but most of our time was spent on the actual development. Difference was also felt on daily meetings, called “daily stand-up” in Scrum terms, which were much briefer than previously. We discussed purely issues at hand and tried to minimize attempts at predicting future.

# **Planning & Estimation**

When we had finished with defining almost all user stories, we had to prioritize them, in order to start with the most important ones. During this process we followed H. Kniberg [5], who suggested using value (from 0 to 99 in our case) instead of priority, where the priority number one is the most crucial. By that, we gave our two core user stories the highest value 99 and 90 and other user stories we evaluated from 0 by tens (10, 20, 30...). When we have added a new user story, we could then easily place it in the middle of two different values, so we did not have user stories with the same value.

Before each sprint, on a sprint planning meeting, we picked the user stories with the highest value and the process of estimation started. We thought about tasks it will include and then we used the Fibonacci numbers to estimate them (Figure XXX).

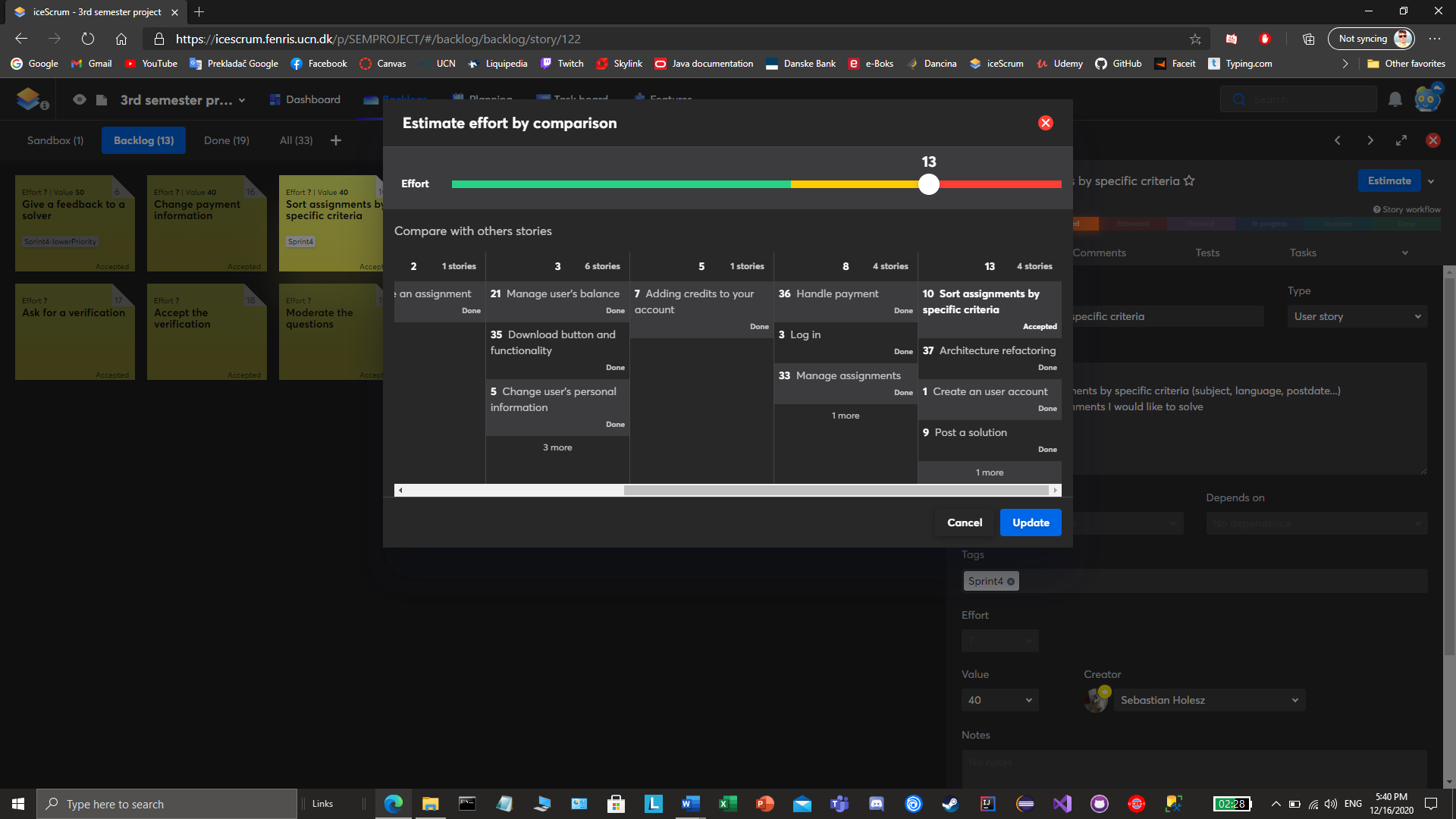


Figure 2 Estimation process

We used a breakdown and calculation method when we ended up with a user story with value higher than 21, we split it into two different stories. It was very difficult for us to estimate the first sprint, therefore we based our estimation on the best guess which was mainly influenced by our previous projects. After that we ended up with a sprint velocity of 26. On the sprint retrospective we agreed that we finished everything but did not have time to implement a new user story. On the other hand, we wanted to challenge ourselves, thus for next sprints we used an analogy planning method and planned the entire sprint with very similar, but slightly higher velocity (sprint 2 - 29, sprint 3 - 32, sprint 4 - 35). As you can see, we increased the velocity by nine between sprint 1 and sprint 4, which was caused by changing the overall architecture in sprint 4. Unfortunately, we did not play planning poker, because as we mentioned above, we got rid of epics and during estimation we all had similar opinions.

Next step during sprint planning was to divide whole user story into small tasks, which were then grouped up into 3 categories – TO-DO, In progress and Done, so we could divide them among us and work separately without conflicts. XXX shows tasks for the Post an assignment user story.

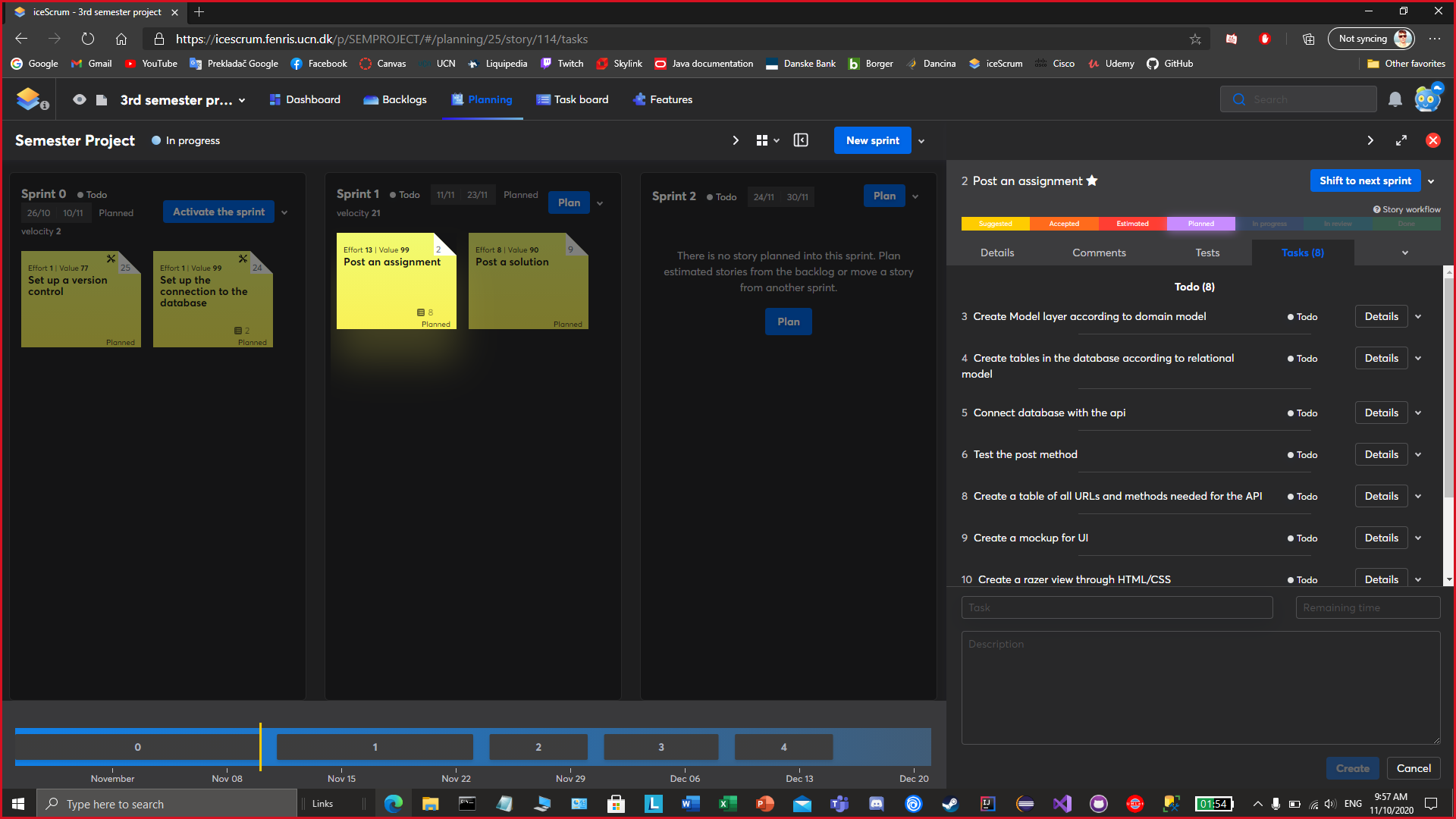


Figure 3 User Story - Post an assignment

During each sprint and on sprint retrospective we used Burnup Chart. It is a graph that helped us to analyze the progress of the work in relation to the total story points for the current sprint. In the figure XXX you can see burnup chart for sprint 4. The yellow line shows total number of story points, for this sprint it was 35. The blue line shows average or the ideal progress which should be made. The green line shows our actual progress. As you can see, at the beginning of the sprint we were behind the schedule, which was caused by changing the overall architecture. We were dealing with this user story for 2 days, but after we finished with it, we were constantly above average, therefore we added one more user story to this sprint – Change profile picture (changed the overall velocity by 1).

Chart, line chart

Description automatically generated

Figure 4 Burnup Chart for Sprint 4

At the end of the project, we had also in mind what would be the next features we would like to implement. For next releases we would add:

* payments and other third-party APIs,
* forum page,
* a mobile app or make it optimized for mobile users.

# **Quality assurance & Quality management**

## **Quality criteria and architecture**

Reflections on methods and their uses in practice

# **Risk analysis**

Test Risk analysis can be best described as a process of assessing the likelihood of an unfavorable events that can happen during the project and can influence its success and lead to failure.

We started with risk analysis at the beginning of the project, even before sprint 0, by identifying the risks. You can see the result in the table XXX in first column. After we found out what risks can affect us, we analyze the likelihood and the consequence of each one of them and by multiplying these two values we got the overall score (risk priority).

The top three risks are very connected to each other, because we were using completely new methodology, which is different from plan-driven, and our planning was divided to sprints. From beginning it was very difficult to estimate, therefore the high probability and impact. Even though, there is a corona pandemic, we set the impact to very low, as we already worked in a situation like this in our previous project. We think that this is even a good experience to the future as well, because we could try how is it to work only online, instead to face it our jobs for the first time. Last two risks are brought from last project, where we also encountered burnout at the end of sprint and not all of us are using Windows.

Table 1 Risk analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Probability** | **Impact** | **Score** |
| Wrong total estimates | 8 | 7 | 56 |
| New system architecture | 6 | 9 | 54 |
| Wrong sprint estimates | 8 | 4 | 32 |
| Fluctuant sprint velocity | 8 | 3 | 24 |
| Illnesses | 7 | 2 | 14 |
| Corona pandemic | 10 | 1 | 10 |
| New programming language | 1 | 9 | 9 |
| Problems with new SD framework (methodology) | 2 | 4 | 8 |
| Burnout | 2 | 4 | 8 |
| Different OS | 1 | 5 | 5 |

Next step was to plan how to minimize the effect of these risks as a part of qualitative risk analysis. We did not plan everything ahead, but we were monitoring it continuously during the entire project. We have only written down the strategy for the most crucial risks. For the new system architecture, the strategy was to find out what suits us the best and ask for a feedback. Nevertheless, we planned this, we end up with changing it two times, which had great impact. With the estimations, we decided that in the first sprint we will not overestimate ourselves, based on that we chose the starting sprint velocity and after that we knew if we could deal with higher or not. Monitoring crucial risks is shown in table XXX.

Table 2 Risk monitoring

|  |  |
| --- | --- |
| **Risk** | **Indicators** |
| Estimation | Good estimation at first sprint without changing the velocity by much |
| Architecture | Changed architecture 2 times |
| Illnesses | very low impact other members substituted |
| Different OS | dealt by booting to Windows and by changing .net framework to .net core |

# **Requirements definition**

## **Functional**

## **Non-functional**

# **Configuration management**

## **Version control**

# **CONCLUSION**

# **BIBLIOGRAPHY**

|  |  |
| --- | --- |
| [1] | Wikipedia, "Wikipedia - The Free Encyclopedua," [Online]. Available: https://en.wikipedia.org/wiki/Extreme\_programming. |
| [2] | D. Wells, "Extreme Programming - Values," 2009. [Online]. Available: http://www.extremeprogramming.org/values.html. |
| [3] | S. Prakash, "Geeks For Geeks," 2020. [Online]. Available: https://www.geeksforgeeks.org/c-sharp-coding-standards/. |
| [4] | B. W., "Explain Agile," 2018. [Online]. Available: https://explainagile.com/agile/xp-extreme-programming/practices/metaphor/. |
| [5] | H. Kniberg, SCRUM AND XP FROM THE TRENCHES How We Do Scrum 2nd Edition, C4Media, 2015. |

https://www.productplan.com/glossary/product-vision/

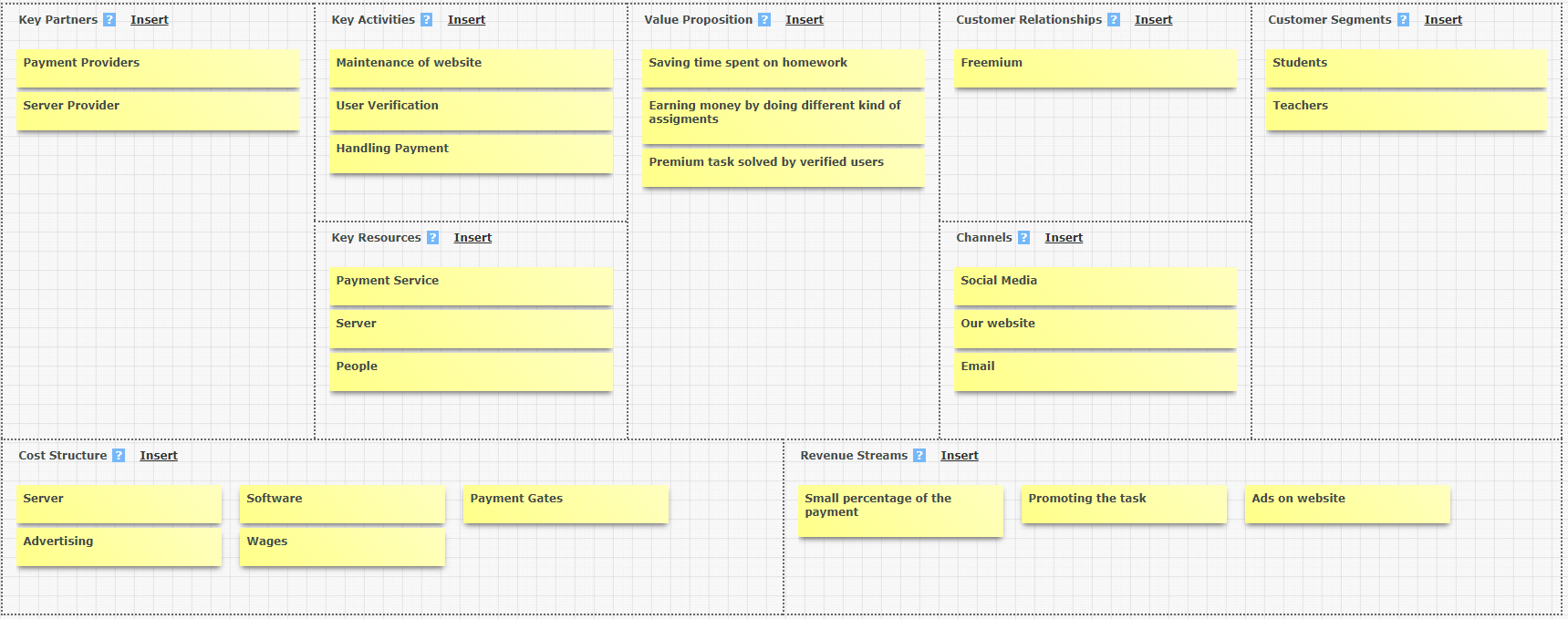
https://www.bmc.com/blogs/scrum-vs-kanban/

https://www.visual-paradigm.com/guide/software-development-process/agile-vs-waterfall/

https://en.wikipedia.org/wiki/Extreme\_programming

# **APPENDIX**

Appendix A – Business Model Canvas



Appendix B – Mock-ups

