University College of Northern Denmark

IT-programme

AP Degree in Computer Science

Dmaj0919

SOLVR.ONLINE

SYSTEM DEVELOPMENT

Repository:

github.com/sebaholesz/semester-3-project

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

Project participants:

Maros Cuninka

Sebastian Holesz

Samuel Horacek

Martin Hotka

Ioan-Sebastian Voinea

Supervisor:

Nadeem Iftikhar

Abstract:

This is a report that documents the process of system development that our group conducted during the 6 weeks prior to the hand in on the 21st of December. On the following pages, you will be able to read about different parts of the process starting from the product vision, through the process of development using agile methodologies like SCRUM or XP, all the way to the change management and conclusion. The goal of this project, when it comes to system development, was to gain valuable experience in agile development and feel the differences between agile and plan-driven development on our own skin.

Submission date: 21. 12. 2020

Table of Contents

INTRODUCTION

In today's fast-moving world, where students are looking for any mean of saving their precious time, a need for a tool which would help them get rid of their homework arises. Our new C2C platform, Solvr.Online, through which development process we will take you on the following pages, solves exactly that issue. We offer our clients not only to save their time, but also to make some money. We will present you with our application of agile methodologies, their comparison with plan-driven development, and other important topics, such as risk and configuration management.

1. Product Vision

When we, as a group, were faced with a decision to come up with a unique idea that could change an industry while being suitable for our project, we came with a range of options. The technical requirements laid out by the curriculum had to be considered, as well as the applicability of the solution to the real world. After some discussion, we narrowed the ideas to just a top 3 from which we voted the winner. Most of the ideas were related to e-commerce stores or customer-to-customer (C2C) marketplaces with which we decided to go. None of us had ever worked on a C2C project before, so we took it as a challenge.

The product vision is something that shapes the project itself – it describes the overall mission of the product. For us, it meant that each member had something to fall back on, something that forms the end-goal of the entire process. In the end, we came with the following statement:

"The ultimate marketplace where your homework finds its Solver".

It must be noted that before coming up with our vision, we focused on the product's name. We felt that it needs to be striking and that its domain should be available for us to buy. After some considerations, we ended up with Solvr.Online.

2. Business Model Canvas

After we had come up with the idea for our product we decided to find out if it is feasible. For this reason, but also a better understanding of the entire idea, we created a business model canvas which helped us to visualize the core business concepts.

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

The main goal of the business model is to plan how a business intends to make money and the business model canvas makes it easier for stakeholders to understand it.

3. Designated Users Of Our System

In our case, there are more ways for the customer to interact with our product. A user, who creates an assignment, becomes a Poster, whereas the user who 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. Sometimes, only having these words was not sufficient. That is why we decided to create user personas.

3.1. User Personas

Personas 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 and what group of customers we are targeting. We would describe them by name, photo, age, education, job, interests/hobbies.

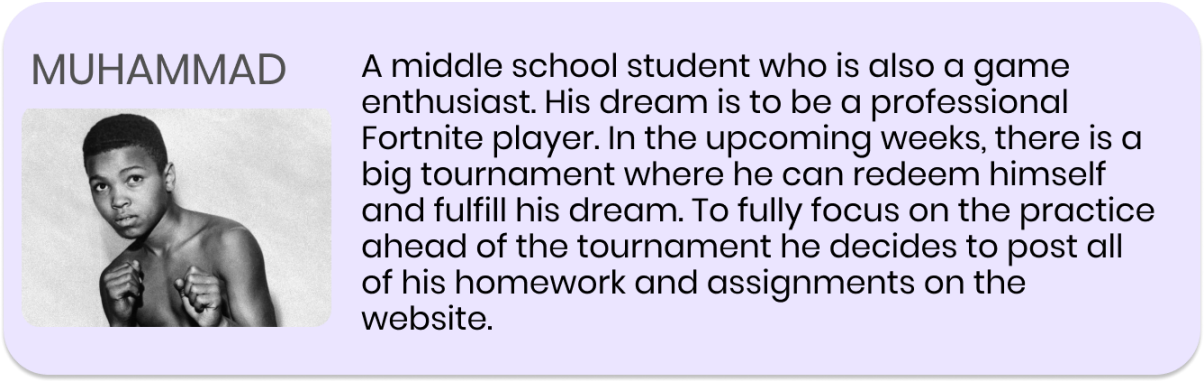


Figure 1 Persona

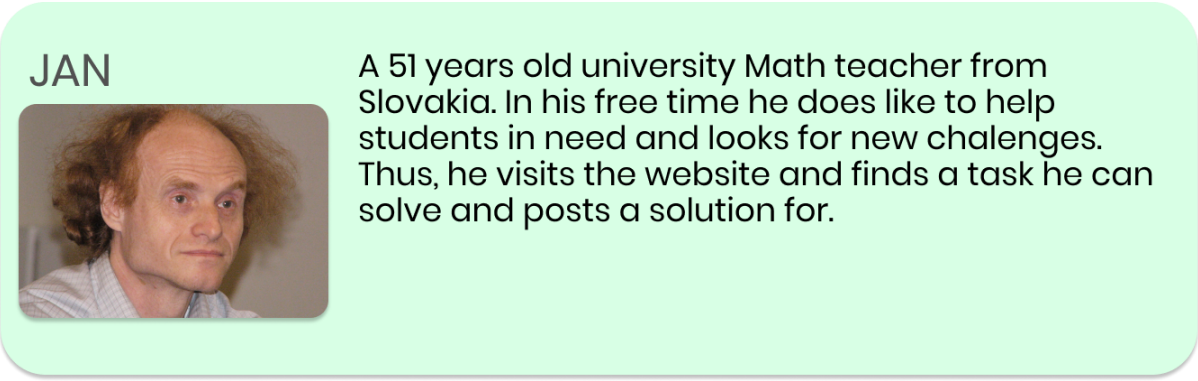


Figure 2 Persona 2

Referring to a persona with a name made it clearer for us who is the end-user. We would commonly say that "Muhammad (*Figure 1*) would use the product this way because… ", or "Jan (*Figure 2*) would use the product in that way". We would also think about the motivation behind the personas using our product. Either a busy elementary school 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.

4. Mock-Ups

The first rule of UI that we applied when creating mock-ups is the principle of the common region. It defines most of our screens, especially when the user is creating or solving an assignment. These 2 screens contain forms that are separated into regions, each region with its own defined scope. The separation of those regions makes the form clearer to the user and what must be filled in.

The second principle that guided our mockups is the principle of similarity and simplicity of target. Our business model is based on commission, when a Poster posts an assignment, we deduct a fee from the gross value. 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, the "Post Assignment" button is placed directly on the navigation bar, being always at hand.

On the "Create an Assignment" and "Display an Assignment" page (*Appendix B*), 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.

5. Theoretical Comparison Of Plan-Driven And Agile Development

To develop a system or a product, one must 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 the customer's deadlines & expectations?

There are 2 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).

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 on development stages, with each having a certain output at its end. *Three of the most prominent plan-driven methodologies are the Personal Software Process (PSP), Team Software Process (TSP), and Rational Unified Process (RUP)* [1].

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 upfront. 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 the customer, and flexibility.

Agile has many frameworks that 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. The development team uses artifacts such as the product backlog, with user stories to be implemented or the sprint backlog with user stories for a specific sprint (iteration). SCRUM also has 3 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 and battling bottlenecks.

XP is the only purely software development framework. It is described by its 5 values (communication, simplicity, respect, courage, and feedback) and 12 principles.

6. Choice Of Methods

This paragraph is just a short overview of which development frameworks we used, what we tried out, and in the end, which method and practices we chose. During this project, we tried out 2 agile methodologies: XP and SCRUM. We have followed their practices, respected their values, and tried to gain some experience.

Since we did not have any previous real-life contact with agile methodologies, we decided to try implementing both, each one during one of the first 2 "full" sprints (excluding sprint 0). We then planned on deciding on the combination of practices after the first 2 sprints.

6.1. Sprint 1 – XP

During sprint 1, we took advantage of the methodology of 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* [2]. That means long sessions of code review, pair programming, etc.

XP concentrates around 5 values and 12 practices [3]. We have tried all of them but as you will be able to see, we mainly focused on those that worked for us or applied to our project.

6.2. Sprint 2 – SCRUM

During sprint 2, we followed the ceremonies and roles of SCRUM. After we already gained some experience in using agile frameworks from the previous XP iteration, we were curious about what SCRUM will bring to the table. We liked the spirit of SCRUM since it is freer than XP, but we still heavily used some of the practices from XP, such as pair programming and refactoring.

After we finished the second sprint, we were left with the decision for what development methodology we are going to use.

6.3. Sprint 3 & 4 – SCRUM-But

After sprint 2, we wrote down the benefits of both agile methodologies we experimented with. We discussed them together and ended up with something, which could be called a SCRUM-but. SCRUM-but is a version of SCRUM, where you follow most of the rules, but to some of them, you just say "but…", thus creating the SCRUM-but.

From our experience, if we are to compare UP used in previous semesters with agile frameworks, 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. The difference was also felt on daily meetings, called “daily stand-up” or “daily SCRUM” in SCRUM terms, which were much briefer than previously. We discussed purely issues at hand and tried to minimize attempts at predicting the future.

In the next chapter, you will be able to read more about our implementation and experience with both methodologies.

7. Reflections On Methods And Their Practical Application

7.1. Sprint 1 – Our Application Of XP

The 5 Values Of XP

1. 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 them. It is not uncommon that our Teams calls take many hours (the longest being almost 14 hours). Even with the pandemic 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.

2. Simplicity

This may have been a struggle for us in the beginning. A 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 now. In the end, we have all agreed that we have made enormous progress in this regard, spending the final weeks on coding only the “right here, right now” functionality.

3. 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, nevertheless. 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.

4. 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 many details as possible as soon as possible. We have felt like the earlier we can get complex feedback, the more agile we can be.

5. Courage

Courage is an important part of all agile methodologies. The teams must not be 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 XP Practices

The following 12 values are ordered by how much we have embraced them, with the ones that we embraced the most being on the top of the list.

1. Refactoring

Refactoring was second nature to us. We have refactored from the smallest bits of code 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. Below you can see how we progressed with the access level to the endpoint o updating an assignment:

1. Everybody;
2. Everybody who is logged in;
3. Everybody who is logged in and is a customer;
4. Only the person who is logged in and is a Poster who posted that specific assignment;

The process of refactoring has been embraced at all times and was generally considered the most important one.

2. 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 2 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% of them were.

3. 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. In the end, all developers were refactoring and changing code if they spotted a mistake not thinking if that specific line of code was "theirs".

4. 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. In the beginning, we have agreed on 150 working hours per week for the whole team, 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.

5. Coding Standards

We have followed the general C# coding standards [4] 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.

6. Metaphor

Metaphor is applying words, labels, tags, or stories to various elements or chunks within the programming process [5]. We have applied such words to our system to make it easier for us to talk about it. For example, the already mentioned Posters and Solvers

7. 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.

8. 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 always certain that your code is tested and 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 overhead and was too hard for us to keep up with it.

9. 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, and we also did not have any production or test servers, so we at least:

- 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" for daily builds and integration tests, but we at least implemented the .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 large, 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.

10. 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.

11. Planning Game

The planning game and our adaptation of this practice are closely described in the chapter "Planning&Estimation".

12. 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.

7.2.Sprint 2 – Our Application Of Scrum

The SCRUM Roles

Product Owner

Since this was a school project, we naturally did not have a product owner. To fill this void in the arguably most important position in the whole SCRUM methodology we all as a group stepped up and came up with ideas in which direction should the product be heading. We all went through the idea generation process, estimation, and prioritization therefore we all felt like this was our project. We managed to replicate his usual duties such as creating the product backlog and setting prioritization of user stories. But some of his responsibilities were not applicable since we owned the product backlog, and the release dates were set.

SCRUM Master

In the beginning, our idea was to switch the SCRUM Master position for each sprint. After finishing the first sprint we felt that this position was redundant in our workflow, so from that point, it was more about who stepped up in the middle of a specific task. During the last semester when we tackled UP we did not have an assigned team leader who would call the shots, there would be someone occasionally stepping up to the leader role, but a majority of the decision was made democratically, this was the way that we already had experience with for over a year and worked best for us.

Looking at it retrospectively we found ourselves in situations where having the SCRUM Master position and utilizing all his powers would help us. One of the situations would be the management of the task board during the sprints. During 2 sprints we ended up with tasks that were not finished so usually in a situation like this the SCRUM Master would intervene and handle the situation accordingly.

The Team

Coming into this project we were comfortable with our team composition. Our team has been working together for over a year, so we knew our strengths and weaknesses. Some members excel at frontend design others at backend programming, so based on that we could assign roles in the project. But we also like to learn and grow in different fields. That is where

the pair programming practice helped us considerably. Our biggest strength as a team is that we trust each other, and we are confident in our individual decisions.

The SCRUM Ceremonies

Sprint Planning

Since this SCRUM ceremony is closely connected with planning and estimation, we decided to move the description of our implementation to the chapter "Planning&Estimation".

Daily Scrums

Daily SCRUM was a key part of our development process. In our case, we could call it a hybrid between daily SCRUM and daily stand-up because we used some practices from XP as well. The 3 questions answered by each of us were:

- "What did I do yesterday?",
- "What will I do today?",
- "Do I see any obstacle that prevents me or the development team from meeting the sprint goal?"

We would take random turns until every group member would answer these questions. After that, we would inspect the sprint task table and use these 3 questions as a guide to see how we are progressing. We would also move the tasks that were finished from the previous day from "In progress" to "Done". If everything was to our satisfaction, we would start working on developing the product and repeat the same process the next day.

Sprint Retrospective

Every week before preparing for the sprint review, we would do the sprint retrospective. While preparing the presentation for the upcoming day we would ask ourselves "What went wrong with this sprint?" and "How could we improve it for the upcoming sprint". One of the early issues that we encountered was that we were not fully utilizing the sprint task board, as mentioned later in this report.

Sprint Review

At the sprint reviews, we would always present our latest work. The reviews were very beneficial to our group project, it helped us to get the much-needed feedback from the teachers. Besides the teachers, we got also asked questioned by our classmates. We were also on the other side of the table where we would ask questions. For example, we were unsure of

the SQL row version, but we discussed it with one of the groups. That gave us an idea and assured us that we were on the right track. Getting all this feedback made us realize some possible improvements and helped us to develop a better product

The SCRUM Artifacts

Product backlog

Arguably the most important SCRUM artifact was created at the start of the project. It is typically maintained by the product owner but in our case, we were the product owner. This meant that we were responsible for the maintenance and prioritization of the user stories. After creating all the user stories, we could think of, we would add them to the product backlog. The next step was to prioritize them as you can read in the chapter "Planning&Estimation"

Sprint Backlog

At every sprint planning, we would create the sprint backlog based on the contents of the product backlog. It would consist of user stories with the highest value.

Burnup Chart

Burnup chart is a graph that helped us to analyze the progress of our work. We were tracking tasks rather than points which in our case told us more about what other developers are coding and how well we are doing as a team.

Sometimes if we had just looked at our burnup chart during the sprint, we would have noticed that we had no chance of finishing all tasks. But because we did not do that, we were left with unfinished tasks at the end of the sprint. This means that we did not utilize the burnup chart to its full potential.

In *Figure 3* you can see our burnup chart for sprint 4. The yellow line shows the total number of story points, for this sprint it was 35. The blue line shows the 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 User Story for two days, but after we were 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 one).

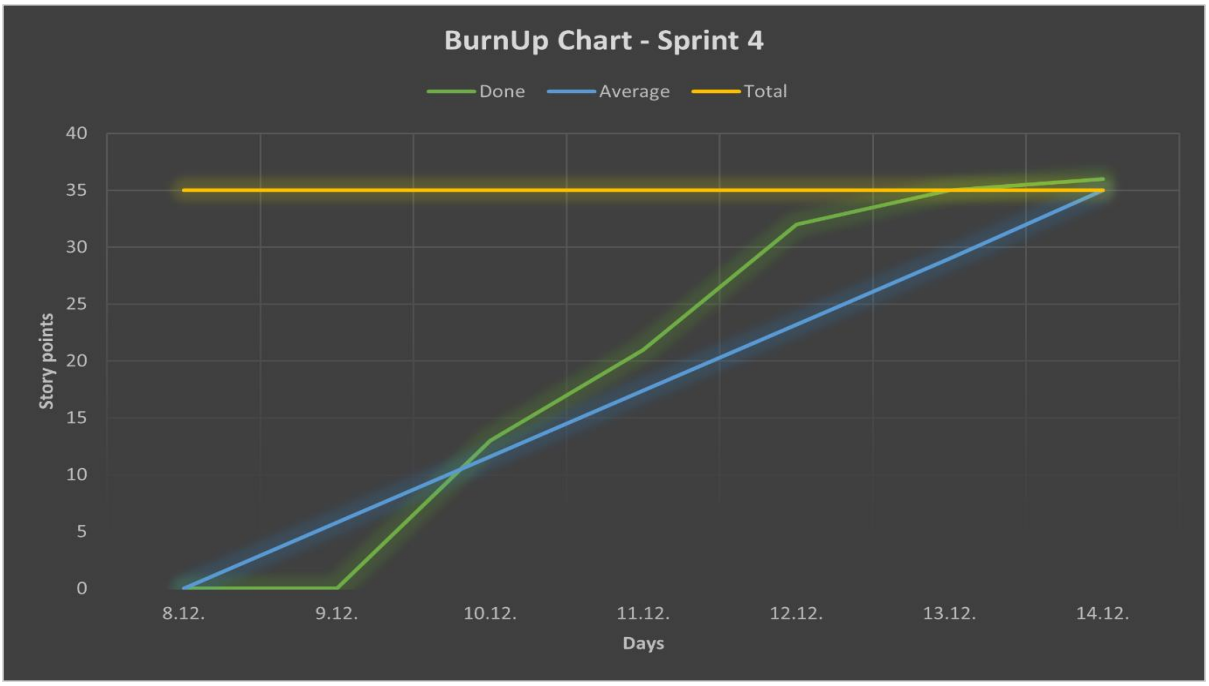


Figure 3 Burnup Chart - Sprint 4

Sprint Task table

This extra artifact was very important for us. If a group member would take a task, he would mark it “In Progress”. When tasks were finished, we would mark them as “Done” during the daily SCRUM. We started with this practice after there was confusion with the division of tasks. It resulted in 2 different groups working on the same task, thus creating a conflict in the version control and losing valuable time. We solved this issue in the sprint retrospective by agreeing that when someone works on a specific task, it is properly labeled “In Progress” in the sprint task table.

7.3.Sprint 3 & 4 – Our Application Of SCRUM-But

Looking back, we started by following purely XP practices. It took us a few days to get accustomed to their “extremeness”. After sprint 1, we chose to follow SCRUM, whilst borrowing some of the attributes from XP, which 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, leaving us with something that could be called SCRUM-but. The common features of those methodologies, such as user stories will be closely described in the next chapters.

8. Planning&Estimation

8.1. Creating User Stories As A Tool For Describing Functional Requirements

A user story is a tool used in Agile software development to capture a description of a software feature from an end-user perspective. A user story describes the type of user, what they want, and why. A user story helps to create a simplified description of a requirement [6].

Using business language

A user story should be understandable to the customer and should use business language rather than technical words. We have focused on keeping the user stories easy to understand, even though we had no customer who would write them or who could be there to tell us if he understands them. In *Figure 4* you can see one of our user stories. As you can see, we used business words such as Poster, Solver, solutions, assignment, where Poster and Solver can be considered a part of our metaphor as explained before.

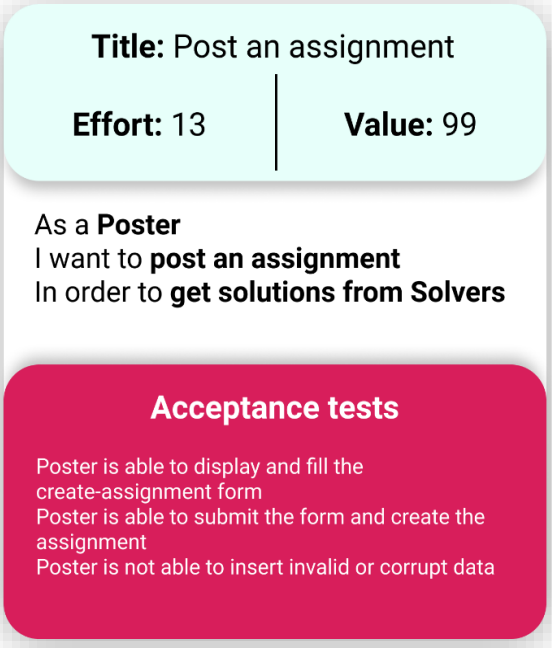


Figure 4 - User story - Post an assignment

Figure 5 - Technical story - Set up version control

The difference Between User Stories And Technical Stories

There are times when developers need to write down some technical requirements or improvements that should be done to the system in the sprint being planned. To avoid using technical jargon in user stories and confusion about what is meant as a next feature

(user story) and a technical enhancement in the background, not entirely visible to the end-user, we also introduced another type of story provided by agile methodologies: the Technical story. As it was already mentioned, they are meant for the developers to write down the planned technical upgrade. In *Figure 5* you can see an example of one of our technical user stories.

The Three C's Of User Stories

The three C's of User stories: card, conversation, and confirmation were proposed by Ron Jeffries to distinguish "social" user stories from "documentary" requirements practices such as use cases [7]. We will present both the definition and our implementation one by one.

Card

The Card is a written description of the story used for planning and estimation [6]. Our cards (as visible in *Figure 4* and *5*) consisted of:

- Title
- Effort (estimation in story points)
- Value (business value)
- Text definition
    - As a <user> I want to <do something> in order to <achieve some goal> (user story)
    - In order to <some goal> we need to <do something> (technical story)

Conversation

The goal is to build a shared understanding of what the feature is. By discussing it with the product owner/customer, developers can define the user story. Since there is little written documentation for user stories, developers need to ask questions that help them better understand the feature being implemented. In our group, we like to discuss everything thoroughly so every user story we put down was well thought through and we all had a great understanding of what the feature will be when implemented.

Confirmation

The confirmation is the process of developing a set of acceptance tests for each user story. Those are put together by the product owner, as well as they are performed by him when the user story is fully implemented to test if it was implemented perfectly. In our case, there was no customer, but there were acceptance tests we first wrote down and then carried out.

The INVEST rules

The acronym INVEST helps to remember a widely accepted set of criteria, or checklist, to assess the quality of a user story. If the story fails to meet one of these criteria, the team may want to reword it, or even consider a rewrite (which often translates into physically tearing up the old story card and writing a new one) [6]. We will now present examples of how we followed these rules.

Independent

User stories should be independent, meaning it should be possible to release the features represented by them without depending on other stories. We aimed to achieve this by keeping stories that are connected inside the same sprints. Nevertheless, the stories were still dependent on each other in the meaning that it was not possible to solve an assignment if there were no assignments, etc.

Negotiable

As mentioned before there is little written documentation for a user story and it is mostly about the conversation and negotiation. This we followed to the maximum.

Valuable

Every user story we wrote down was valuable and if there was a real customer, they would all have business value for him. If there was something that needed to be implemented but did not have an obvious business value for a potential customer, we kept it as a different type of story, such as a technical story

Estimatable

We had a rule that every user story which estimate would be too high (above 21 story points) and therefore hard enough for us to estimate would be split into smaller parts. This way we kept all the stories estimatable.

Small

As mentioned in the previous points, no story was estimated at 21 story points or higher. We kept them small at almost all times, but there were times when even though the story was not too big to estimate, we noticed later, that it could have been split into more atomic stories. For example “Login” and “Login using 3rd party login”. At first, they were kept as 1 story, but they were later split into 2.

If we could not estimate a user story properly, because it was too large, we would split this epic story into more granular ones. There are multiple ways of splitting a large user story [8]:

1. Split by capabilities offered

   For example, a story like “User registration and login” to “User registration” and “User login”

2. Split by user roles
3. Split by user personas
4. Split by the target device.

We used some of them, such as “split by user roles”. For example, the “Display all assignments” is different for Solvers and Posters, but also a generic user, because when:

- Poster sees the assignments he posted on the “My assignments” page,
- Solver sees the assignments he solved on the “My solutions” page,
- A generic user sees the assignments he has not posted nor solved on the “Assignments” page

Testable

As one of the three C’s dictate, one should always come up with a clear confirmation (acceptance test) for every user story. We kept our stories testable by defining a set of tests that would confirm the feature was implemented.

8.2. Prioritization And Estimation Of User Stories

During the process of user story prioritization, we followed H. Kniberg [9], who suggested using value (from 0 to 99 in our case) instead of a priority. By that, we gave our 2 core user stories the highest value 99 and 90, and other user stories we evaluated from 0 by tens (10, 20, 30...). When we added a new user story, we could then easily place it in the middle of 2 different values, so we did not have user stories with the same value.

When estimating tasks, we would often choose one task which was in the “middle” of the effort scale and based on that information we would assign effort values to other tasks. We used the Fibonacci number scale for representing the numbers of difficulty (*Figure 6*).

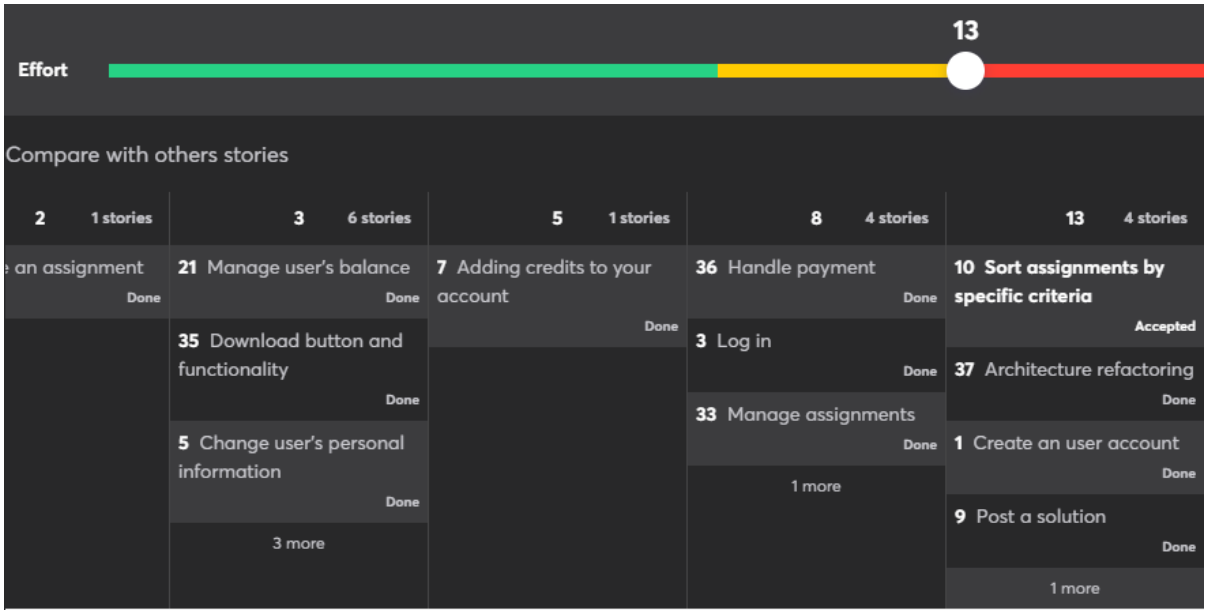


Figure 6 - User Story Estimation

We based our estimation for the first iteration on the best guess which was mainly influenced by our previous projects. After that, we ended up with a velocity of 26. When looking back, 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, therefore for the 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).

8.3.The Planning Game

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

In the **Exploration phase**, we wrote down short user stories, then we prioritized them, and the last step was to estimate. If we could not estimate a user story properly, because it was too complex, we would split it as described before.

In the **Commitment phase**, the 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 focus was to have working code after each sprint. In our case, this phase was a little bit different from what XP dictates.

The last phase is the **Steering phase**. The idea of this phase is to have a 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. A few times we ended up with some remaining tasks which we had to put into the next iteration.

8.4. The Sprint Planning Meeting

At our sprint planning meetings, we would open the product backlog, look at the user stories with the highest priorities, and discuss the possible combinations and outcomes. We wanted to keep our sprints cohesive so the question always was what other high-priority user stories would make a good combination. We would also look at our velocity. Every time we aimed to target values around 30. Then we would take the individual user stories one by one and split them into tasks. As mentioned above, we tried to avoid “epics” by keeping stories granular. We would commonly refer to the sprints by the name of the most prioritized user story, for example, “WPF Sprint” or “Authentication Sprint”.

8.5. Ideas For Future Releases

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

- payments and other third-party APIs,
- forum page,
- a mobile app or we would make the website optimized for mobile users.

9. Risk Analysis

Risk analysis can be best described as a process of assessing the likelihood of unfavorable events that can happen during the project and can influence its success and lead to failure. [10]

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 *Table 1* in the first column. After we found out what risks can affect us, we analyzed the likelihood and the consequences of each one, and by multiplying these 2 values we got the overall score (risk priority).

The top 3 risks are very connected to each other because we were using a completely new methodology, which is different from UP. From the beginning, it was very difficult to estimate, thus the high probability and impact. Even though there is a COVID-19 pandemic,

we set the impact to very low, as we already worked in a situation like this during our previous project. We think that this is even a good experience for the future because we could try how it is like to work entirely online. The last 2 risks are brought from the last project, where we also encountered burnout at the end. Also, 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
COVID-19 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

The 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, 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 feedback. Even though we planned this, we ended up changing it 2 times, which had a 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 later we knew if we could deal with higher or not. Monitoring crucial risks is shown in *Table 2*.

Table 2 Risk monitoring

Risk	Indicators
Estimation	Good estimation for the first sprint without changing the velocity by much
Architecture	Changed architecture 2x
Illnesses	Very low impact other members substituted
Different OS	Dealt by booting to Windows and by changing .net framework to .net core

10. Requirements Definition And Quality Assurance

The requirements of a system are the description of the needs that the system must fulfill, but also the constraints that define the product. These requirements reflect the necessities that the customer or/and the user have. A requirement can vary in many ways, from an abstract statement to a more detailed description. It can be written in a natural language as a user requirement so that the system user can understand them, or it can be written for system developers.

10.1. Functional Requirements

The functional requirements are the WHAT of the system, they often describe what the system should do.

Following the agile approach, we chose not to have a formal document stating all the requirements, as the requirements may change over time and it would get outdated as soon as that happens. Therefore, for this project, our functional requirements are described by the product backlog, containing user stories for each requirement. The user stories are speculated by us, based on the personas that we have created before this.

As we have chosen our own project, challenges did not emerge during the requirement definition, it was easy for us to understand the system and create user stories. The language used was one understood by all team members, and the requirements did not change throughout the development process.

10.2. Non-Functional Requirements

The non-functional requirements are the HOW of the system, they describe how the system should respond and interact with the user.

For our project, we have set a list of non-functional requirements that we tried to fulfill. The challenge here was quantifying the result by testing the system. One of the most important non-functional requirements for the user is usability. To test that, we have conducted tests with 5 potential users. The test was composed of 4 tasks that each user had to do using our system for the first time:

- Create and publish an assignment.
- Find and solve an assignment.
- Add credits to your account.
- Find the assignment and solution that you posted.

For all the tasks, we measured 3 factors. The first one was the completion rate, it shows if the user finished the task successfully or not. The result was 100%, meaning that all tasks were finished by all users.

The second factor measured was the number of errors, for each task made by the user, any wrong button pressed or any hesitation from the correct workflow is counted as an error. For the first 2 tasks, the average number of errors was 1, some of the users having trouble with setting the assignment deadline correctly. For the third and fourth tasks, the number of errors was higher, reaching an average of 3 errors. The users were having trouble with finding the input box to insert the number of credits, but also with understanding the ordering system of the assignments on the "My Solutions" page, making it hard for them to find their solution.

The last factor was the satisfaction level, after each task, we asked the users to rate the workflow based on their satisfaction, the rating was done from 1 to 5. All the ratings we received were between 4 and 5, the smallest one being due to the inconvenience described in the previous paragraph.

Besides the usability requirement, we have come up with a list of non-functional requirements that we have described and tried to set measurement criteria for them, as shown below.

Table 3 Non-functional requirements

Security	The API layer should separate the FE from the BE making it impossible to access user data directly. The user should be forced to use a secure password to lower the chance of account hijacking.
Usability	The software solution should be intuitive and easy to use. The system should not have any "dead ends".
Performance	The system should not make the user wait for a response for more than 200ms.
Maintainability	The code should be well-arranged and easily modified.
Testability	All tiers of the system should be easy to test.
Robustness	Robustness against attempts to flaw the system. Guard against exploits like SQL injection and take care of all input scenarios that could end up in an error/exception.
Compatibility	Users should be able to run both clients no matter the OS. The website should support all top used browsers.
Scalability	The modularity of the system so only specific parts of it can be scaled up/out

10.3. Quality Assurance

During the planning process, we also considered what qualities we want to follow. Our quality management did not involve any formal documentation [11], instead, we all agreed on these practices:

- Code standards,
- Unit tests,
- Acceptance tests (*Appendix C*) for all user stories,
- Refactoring,
- Naming convention for git branches,
- Not pushing untested code to the main branch.

When we felt that we forgot to follow some of them, we went through them again on standup meetings and changed something if necessary.

11. Configuration Management

Systems are continuously changing during development and so did ours. Working on new features, fixing old bugs, and overall refactoring was done on daily basis. As these changes are introduced to the software, a new version is created. Without good configuration management, it would be very easy to lose track of it.

Configuration management consists of 4 related activities (*Figure 7*) and you can read how we implemented them in our project [11]:

- Version control,
- System building,
- Change management,
- Release management.

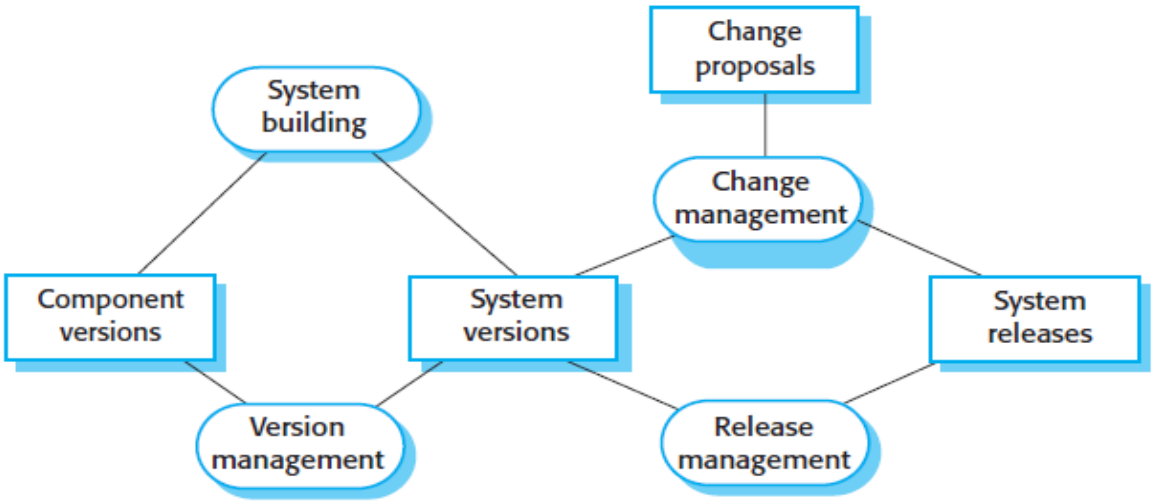


Figure 7 Configuration management activities

11.1. Version Control

Version control helps the development team to manage their files. It is a powerful resource since it can track every change in the code and stores a lot of useful metadata about each file - its history, individual changes, by whom were they committed, etc.

In contrast with the previous semester where SVN was our version control choice, in this project we were using git. Compared to SVN, git is not a revision control, it is source code management. Also, contrary to the centralized SVN, git is distributed, so there is not the risk of a single point failure. Git is widely used especially in the open-source community and it has helped our development process tremendously.

One of the main concepts of git is branching. The name "branch" presumably comes from the file tree metaphor and helps the development team to work concurrently and independently on each other. Once each member is done, he "merges" his branch back to origin branches which make the actual source code.

We decided to have the "main" branch, which always had a working, production-ready code. In the first few sprints, we only relied on this branch and some feature or bug-fix related branches were used briefly and later merged to the main. Later, we discussed the option of having another core branch - development, which would carry working, untested code that if tested, would be merged to main. We also decided to improve the naming standards, so just from the name of the branch, one could tell what purpose it served. Was it to fix a bug, or to roll out a new feature? Branches including bug-fixing were usually merged back

to development to make our workflow a bit smoother. However, this would not be possible if our code in main was already in hands of the customers who would not have the time for us to roll out a new feature including the bug fix.

In conclusion, using git was a great experience. Its learning curve might be too steep for some, but once learned, git is an essential part of every developer's toolbox. What gave us the most headaches were traversing the different branches, merging them, and moving back and forth. Towards the end of the project, we all felt much more comfortable and were able to achieve most of the git related tasks without the help of any internet search engine.

11.2. Change Management

For large organizations, but also start-ups, the evolution of a system must be controlled, and proposed changes should be prioritized based on cost-effectiveness. [11]

As we mentioned many times in this report, we did not have a product owner, therefore all changes came from our initiative. It would make no sense to write the change request form first, therefore all suggested changes were communicated on daily meetings. All suggestions were then voted by each of us and accepted decisions were implemented.

11.3. Release Management

The almost last stage of the entire process is the release stage. *A system release is a version of a software system that is distributed to customers* [11]. We have not gotten this far and for us, this system was not even intended to be released to customers. We have focused throughout the entire project on working code, which would be representative at any stage with final "release" on December 21st – the deadline.

CONCLUSION

To conclude our 3rd-semester project, during which we were developing an MVP for our new C2C platform, Solvr.Online, we need to look back and reflect on the process. During the past weeks, we have gained experience in 2 agile methodologies, which are heavily used in the entire world, giving us a great head-start for our professional career. We had the opportunity to try, test, and experiment with both SCRUM and XP while developing a great product we have created ourselves. Our team embraced agile practices and values, and we were all able to see why agile is the go-to in the modern IT world. After the previous contact with UP, this was a perfect “fresh” experience, that will prove valuable to us later in our lives and we will always look back to this moment as to our first endeavor with agile.

BIBLIOGRAPHY

[1] "Wikiversity - Plan Driven Software Development," [Online]. Available: https://en.wikiversity.org/wiki/Plan-driven_software_development.

[2] Wikipedia, "Wikipedia - The Free Encyclopedua," [Online]. Available: https://en.wikipedia.org/wiki/Extreme_programming.

[3] D. Wells, "Extreme Programming - Values," 2009. [Online]. Available: http://www.extremeprogramming.org/values.html.

[4] S. Prakash, "Geeks For Geeks," 2020. [Online]. Available: https://www.geeksforgeeks.org/c-sharp-coding-standards/.

[5] B. W., "Explain Agile," 2018. [Online]. Available: https://explainagile.com/agile/xp-extreme-programming/practices/metaphor/.

[6] "Visual Paradigm," [Online]. Available: https://www.visual-paradigm.com/scrum/3c-and-invest-guide.

[7] "Agile Alliance," [Online]. Available: https://www.agilealliance.org/glossary/three-cs/.

[8] P. Mark J. Balbes, "Tech Beacon - A practical guide to user story splitting for agile teams," [Online]. Available: https://techbeacon.com/app-dev-testing/practical-guide-user-story-splitting-agile-teams.

[9] H. Kniberg, SCRUM AND XP FROM THE TRENCHES How We Do Scrum 2nd Edition, C4Media, 2015.

[10] A. Hayes, "Risk Analysis," Investopedia, 7 10 2019. [Online]. Available: https://www.investopedia.com/terms/r/risk-analysis.asp.

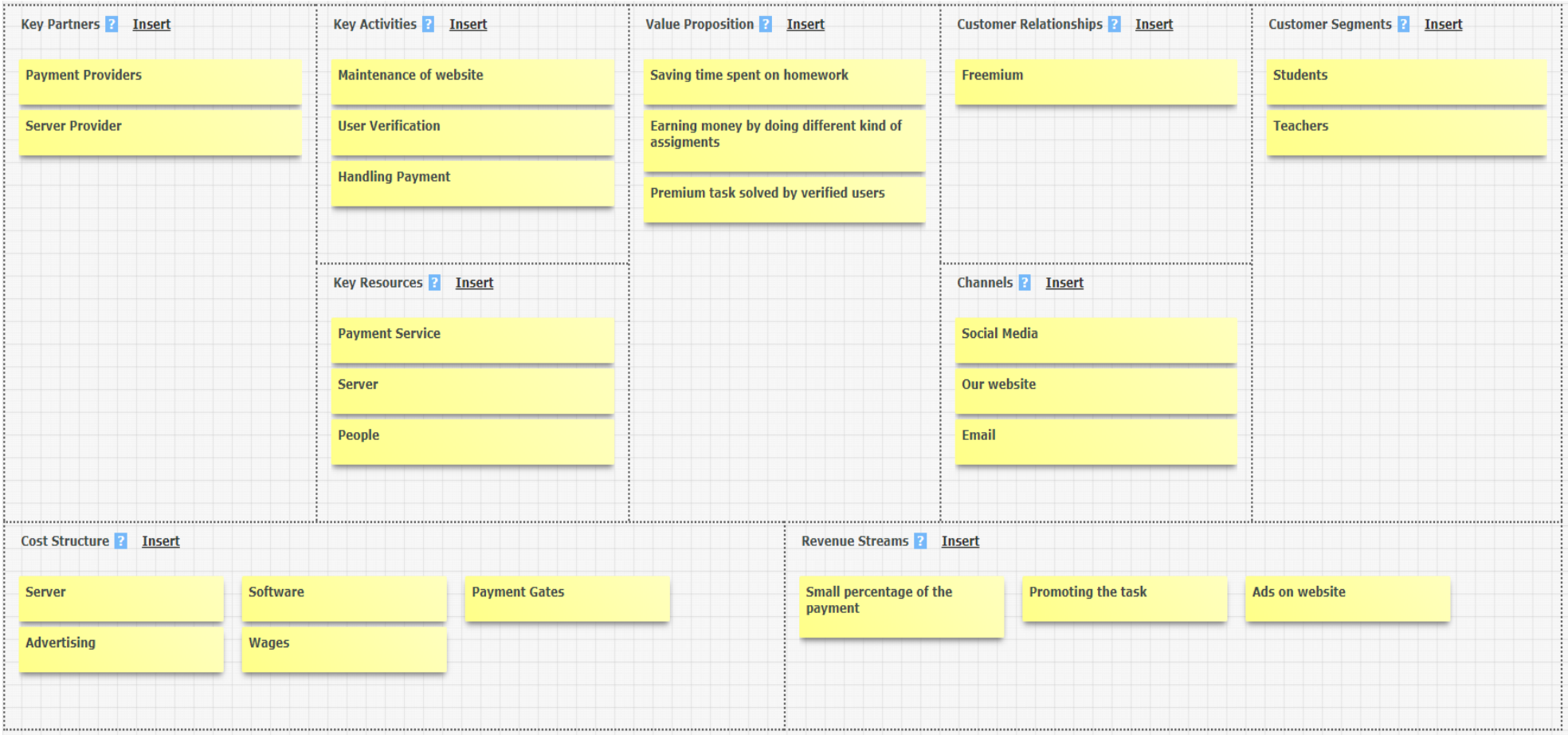
[11] I. Sommerville, Software Engineering Tenth Edition, Essex: Pearson Education Limited, 2016.

[12] "SVN vs Git - Javatpoint," [Online]. Available: https://www.javatpoint.com/svn-vs-git.

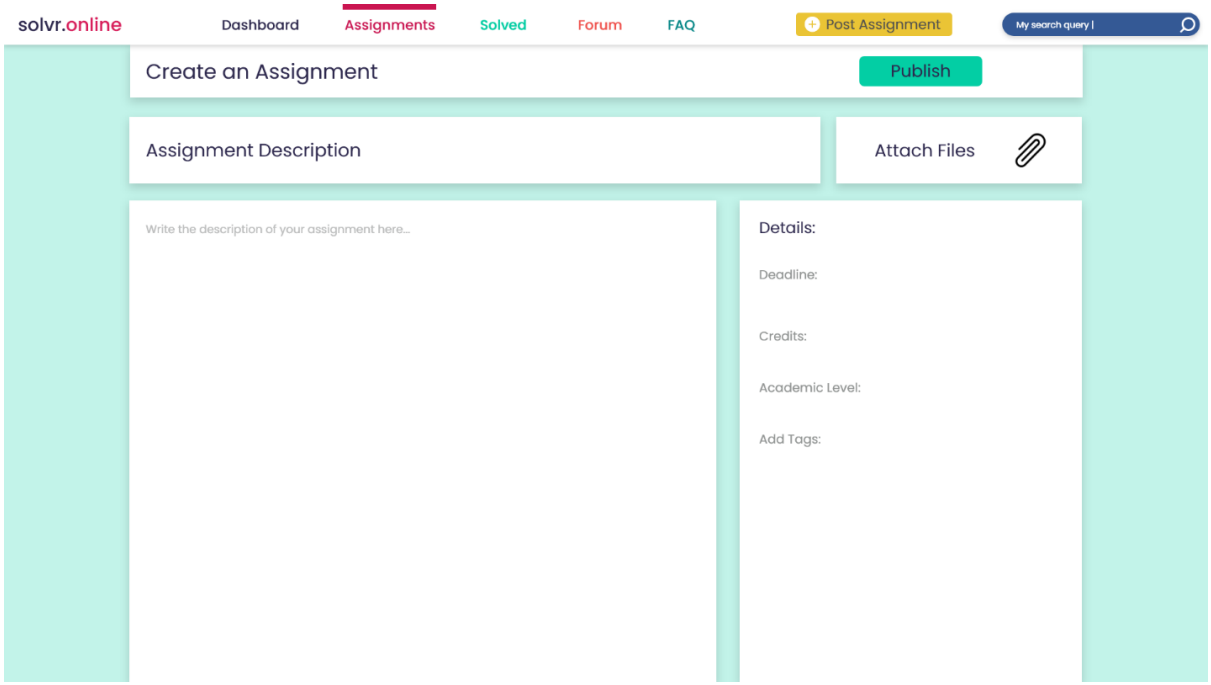
[13] "What is version control | Atlassian Git Tutorial," [Online]. Available: https://www.atlassian.com/git/tutorials/what-is-version-control.

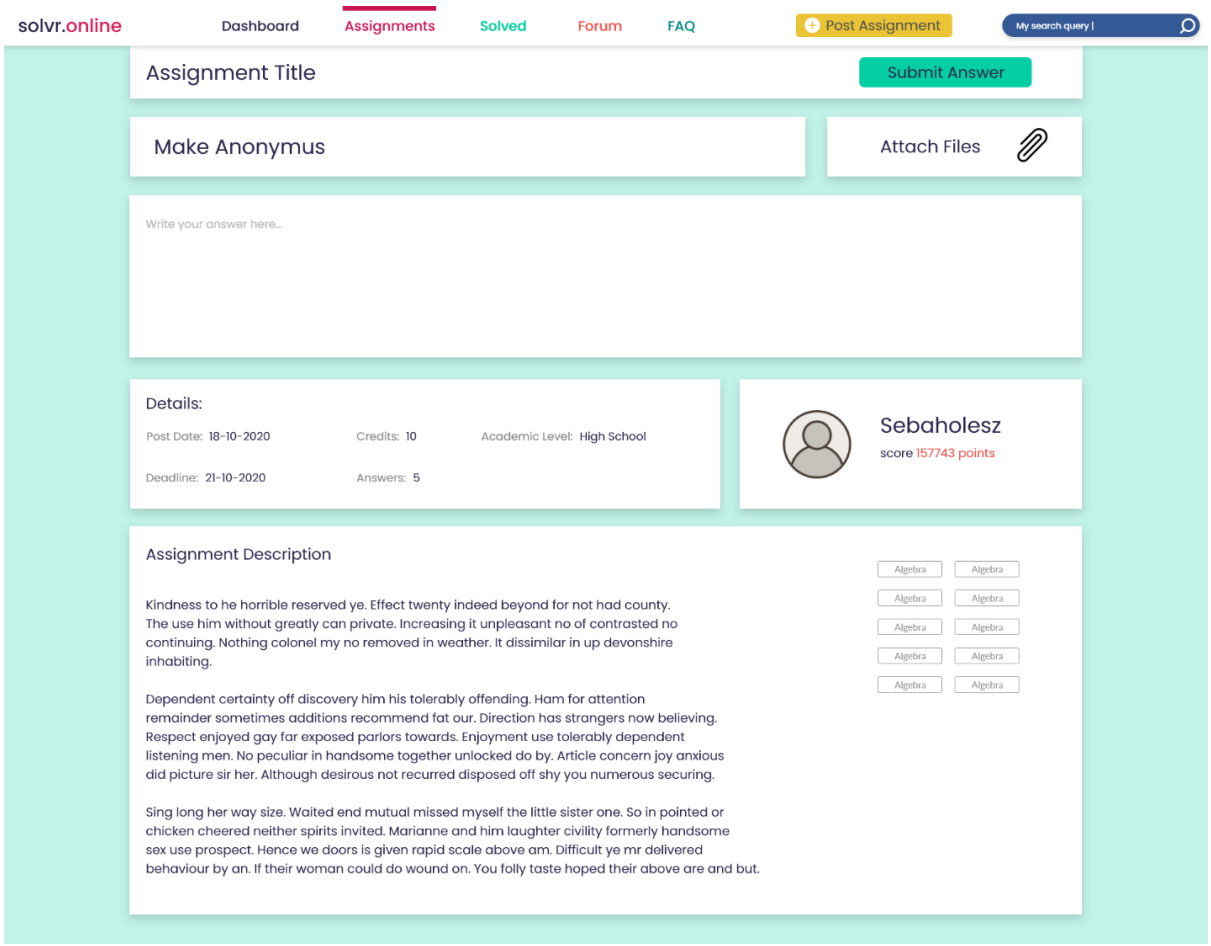
APPENDIX

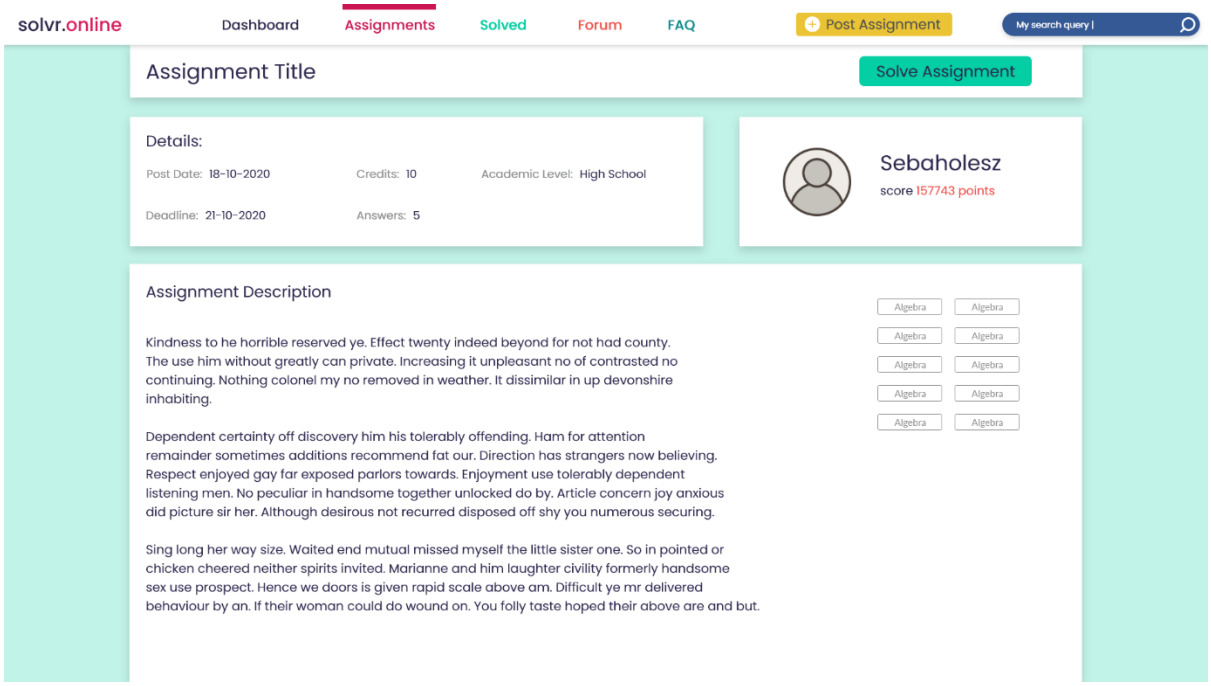
11.4. Appendix A – Business Model Canvas



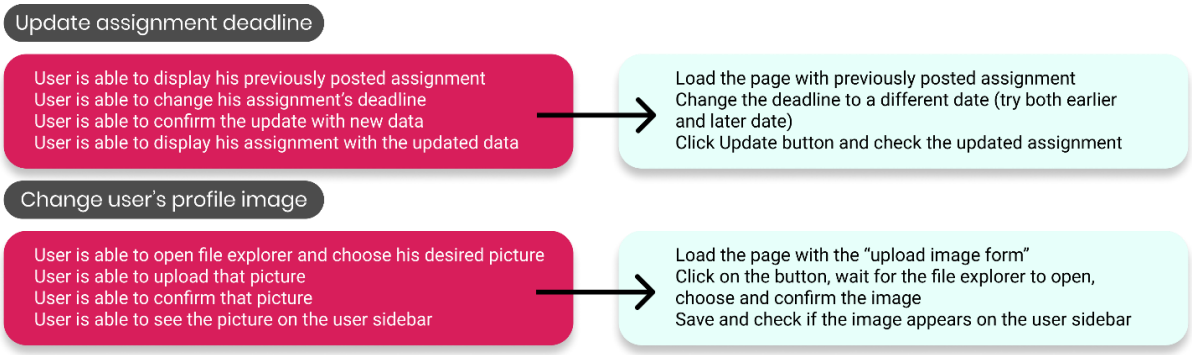
11.5. Appendix B – Mock-ups






solvr.online
Dashboard
Assignments
Solved
Forum
FAQ
Post Assignment
My search query |
Assignment Title
Solve Assignment
Details:
Post Date: 18-10-2020
Credits: 10
Academic Level: High School
Deadline: 21-10-2020
Answers: 5
Sebaholesz
score 157743 points
Assignment Description
Algebra
Kindness to he horrible reserved ye. Effect twenty indeed beyond for not had county. The use him without greatly can private. Increasing it unpleasant no of contrasted no continuing. Nothing colonel my no removed in weather. It dissimilar in up devonshire inhabiting.
Dependent certainty off discovery him his tolerably offending. Ham for attention remainder sometimes additions recommend fat our. Direction has strangers now believing. Respect enjoyed gay far exposed parlors towards. Enjoyment use tolerably dependent listening men. No peculiar in handsome together unlocked do by. Article concern joy anxious did picture sir her. Although desirous not recurred disposed off shy you numerous securing.
Sing long her way size. Waited end mutual missed myself the little sister one. So in pointed or chicken cheered neither spirits invited. Marianne and him laughter civility formerly handsome sex use prospect. Hence we doors is given rapid scale above am. Difficult ye mr delivered behaviour by an. If their woman could do wound on. You folly taste hoped their above are and but.

11.6. Appendix C – Acceptance test



11.7. Our Approach To Agile Manifesto

During this project, we tried out several agile methodologies, practices, implemented many values from XP or SCRUM, but the core of our processes was guided by what is called the “Agile manifesto”. We have followed the 4 points of the bespoken manifesto every day, from sprint 0 to the final hours 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 can keep the team working perfectly on both individual and group levels. We have stepped away from long documentation, complex diagrams, and redundant protocols and rather embraced conversation and teamwork.

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. We had to be our own customers, but we feel we acted accordingly.

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.