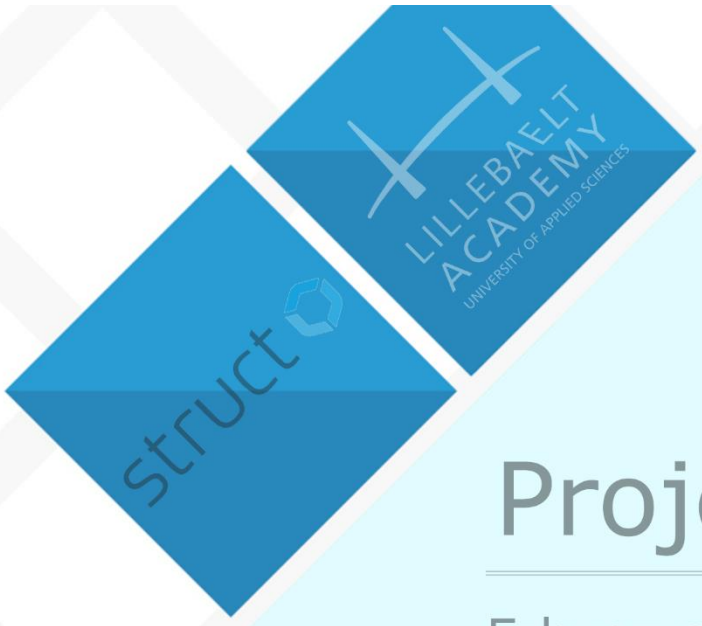

Project Report

Erhvervsakademiet Lillebælt
OEADM16EC Computer Science
3rd Semester
Group E

Product

Updating system

for Struct A/S

Group participants

Bogdan-Iulian Vasile	bogd0268@edu.eal.dk
David de Lima Fraga Alves	davi7816@edu.eal.dk
Elitsa Miroslavova Marinovska	elit0452@edu.eal.dk
Luca Treamamunno	luca9110@edu.eal.dk

Project advisers

Lene Andersen	lean@eal.dk
Simon Stochholm	sist@eal.dk
Steffen Mogensen	stmo@eal.dk

Hand in date: 15/12/2017

Preface and acknowledgement

This report is developed with the purpose of concluding the System Development course of the third semester in Computer Science education at Erhvervsakademiet Lillebælt. It summarizes the team's working process during the period from week 43 to week 49, based around various System Development methodologies. The chosen methodologies have reflected ways to determine the best means of managing the following project.

This report has been reviewed by fellow students chosen for their different perspectives and technical knowledge, with the purpose of having independent reviews, providing many constructive comments and suggestions which assisted in considering many aspects of the process.

The underlying work would have been impossible to complete without receiving support and help from our lecturers - Lene Andersen, Simon Stochholm and Steffen Mogensen. We would like to express our gratitude to them for the valuable and interesting discussions throughout the project.

Furthermore the team would like to give thanks to the company *Struct A/S* for investing their valuable time working with us and providing lots of recommendations based on their broad experience accumulated throughout years in the field.

Contents

Table of Figures

Introduction

This project was developed in the scope of the final third semester examination of the Computer Science course and was composed by a team of four elements from the international class. The team was firstly introduced to the project's requirements defined by the lecturers, which consisted mainly of three optional programming topics. From these the team had to find a company which problem will fulfil the project's requirements and collaborate with them during the project's timeframe. This led the team to contact *Struct A/S* which was a company, some of the team members previously experienced working with.

Struct A/S specializes in creating E-Commerce and PIM solutions for their clients. The software used to setup the solutions is based on a CMS called Umbraco, but it's been adapted and changed to suit their needs. They offer both basic solutions with already pre-developed features but additionally can provide any additional features that their customers desire and will tailor the solution to the clients' individual needs.

Problem definition

After meeting with *Struct*'s *A/S* CEO, Peter Melchiorsen, the team was given a project case to work upon. This case's intention was to obtain a distributed system, using a client and a server for managing product's data for distinct suppliers. Therefore, a description of the project while comprising the project's requirements was created and contained specifications describing a Web API that would allow to manage the information of products in a database, as well as indications to develop an ASP .Net MVC application that would provide a graphical user interface.

Reading guide

At the beginning of every focus area an explanation of the chosen SDM will be provided, presenting after it the main documentation produced during this phase with its corresponding description and arguments.

Through the whole report the company with which the team collaborated, *Struct A/S* will be referred to without A/S for the simplicity of reading.

When a text is with a superscript index, it's because it is a word, concept or abbreviation that requires an explanation in the footnotes. It will only be explained the first time it appears in the report.

Appendixes will be referenced like this (*Appendix X.II*).

Images will be named "Figure" followed by a numeration that refers to the chapter that it's located in, and the second numeration refers to the order of the images in the chapter. For instance, "Figure 3.1", would be located in chapter 3 and it is the first image in that chapter. Additionally, images will contain a small description of their content.

When writing "team", an association to the project group and all its members is made.

The following list exemplifies the meaning of abbreviations which might occur in this report:

SDM	System Development Methodology
RAD	Rapid Application Development
CRUD	Create, Read, Update, Delete
HLD	High-Level Design
LLD	Low-Level Design
SSD	System Sequence Diagram
SD	Sequence Diagram
DBMS	Database Management System
IRL	In Real Life (a previous project)
OOP	Object-Oriented Programming
RDBM	Relational Database Model

Project development process

In this section, the development process of the project will be detailed, in areas such as Process model, SDM, Traceability and Documentation.

Process model

Just after getting a general idea of what the project scope would be, the team started working on the decision of the process model. This was done mainly by analysing the complexity and uncertainty of the project, but also by evaluating the general structure of the requested system.

The team identified the project as being of high uncertainty and low complexity.

The project is considered to be of high uncertainty because it requires the team to work with technologies that none of the members have had experience with, nonetheless the project is small enough to be considered of low complexity.

Given the uncertainty of the project, the team discarded the possibility of applying a waterfall or incremental process model, since it would be difficult to produce high fidelity of both artifacts and code from the beginning.

Additionally, given the scope of the project, the team realised it could be split into two subsystems, the first completely usable and independent from the second one. Provided this, the team decided to use an approach in-between agile and iterative, depicted in figure below.

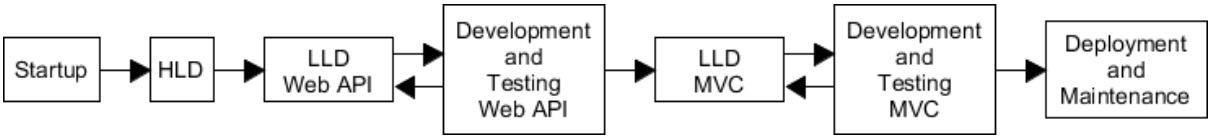

Figure 1.1 - Process model execution plan

The Startup and HLD phases are done for the entirety of the system, but afterwards the team worked on the LLD and Development and Testing phases for the Web API subsystem, iterating through these two focus areas until the whole subsystem was completed. This process would be then repeated for the MVC subsystem, after followed by the Deployment and Maintenance stages performed again for the system as a whole.

One of the main advantages of this approach was that in case any delays occur, the team can deploy at least a working part of the system to the client, while continuing to work on the other subsystem.

SDM

The main objective of the project was to reflect upon how the team made decisions concerning their SDM choice, including actions which were taken along the process, understanding why something is important, analysing different alternatives, evaluating consequences and finding new innovative ways to achieve something. All previously mentioned statements were documented throughout the whole report and the actual SDM choice was specified during the Startup phase, referred in the "Chosen SDM" section, found later in the report.

Quality assurance

Quality assurance has been one of the major focuses of the team in the initial system development phase. After brainstorming the possible artifacts and considering how the chosen methodologies would contribute to the project's development decisions, the team established a set of criteria that can be easily accessed and checked against. This allows for each team member to work independently on several tasks while keeping them within certain predefined standard guidelines. In addition, the team realized how import and beneficial those criteria can be when aiming at a good system quality.

Quality criteria

The team has prepared criteria specifically for the acquisition of feedback coming from independent commentators, leading to a more on point critical suggestions for improvement on the system's quality.

The criteria were composed by revising reliable materials and discussions with specialists from the area. This next excerpt was extracted from the table containing the team's overall quality plan that is present in *Appendix A.I.*

Item	How	Who	When
Rich picture	Criteria: Make sure it shows: various systems and the way they interact together different perspectives and the concerns that stakeholders have everything that is relevant in our system (but not detailed, details belongs to design phase)	The team	During the production process of every item
Workflow diagram	Criteria: Ensure it depicts all the actions in the process Ensure we evaluate all the various conditions		

	Ensure to have a start event which initiates all the following activities Ensure every workflow is reaching and outcome point (end event)		

Figure 1.2 - Excerpt of Quality plan table

FURPS

Another way of assessing a certain quality was by categorizing the requirements by the FURPS model, which were later discussed with *Struct* to make sure they were in agreement with the team. The team interpreted this model according to the following statements.

In terms of *Functionality*, our system must be able to provide the basic CRUD operations on a database, which might be used by either human users or scripts. Another factor to take into account is that a Login system is required, in order to provide custom features for the different users.

Regarding *Usability* of the system, *Struct* did not have any specific requests, therefore the team put more emphasis to the other FURPS' categories.

In relation to *Reliability*, the team assumed that since the system is not critical for any other external component there would be no requirement regarding maximum downtime in case of failure, nonetheless, the team focused on creating a reliable architecture for the system.

Our system should be able to respond to any request from a user within a matter of seconds, thus providing a good system *Performance*.

Finally, for *Sustainability*, the system should be easily testable and extensible while maintaining the ability of being able to move from server to server, or from one database provider to another.

Struct relies mostly on *Functionality* and *Sustainability*, therefore it was important for the team to strictly obey these two categories.

PDCA

PDCA management method came as an ally for the team in the sense that the four steps (Plan-Do-Check-Adjust) model suited perfectly the team's working preference. During early stages of production, an artifact creation has been based on the PDCA cycle, meaning that first the team has planned the artifacts that will be used, followed by the actual creation. They were then thoroughly checked against the quality criteria that the team had set, and in the eventuality of having conflicts, the team would revise them and make adjustments in order to fulfil the criteria.

Traceability

Traceability was a topic that the team kept very much in mind, especially because it came as a fresh topic in the beginning of the project. The team has acknowledged the benefits of

having a strong plan for traceability and decided to set up a handful of tools and strategies that will help reaching a good level of traceability.

To reach this goal the team used a variety of tools, including Github, Google Drive, decision logs and Trello, each of them used for a different purpose.

GitHub - This was mainly used to keep track of the versions of the code and diagrams; it was used in conjunction with Github issues to keep track of the problems encountered, noting when these occurred and when they were solved. Like this, the team had a clear idea about the amount of time spent on each problem and offered an easy way to track down some weak parts of the system.

Google Drive - The team used this tool to keep track of a variety of documents, since it provides better support to rich text files than Github, and also granting a history of the documents and the changes that occurred during the lifetime of the file.

Decision logs - In some cases it was impossible and/or inappropriate to use Github to keep track of some changes that might occur in the diagrams, since in some cases the diagrams were initially being designed on the whiteboard by the team. To keep track of the changes that happened during this phase, decision logs (one for each of these diagrams) were created. On this document the team kept pictures of the different stages of the diagram, with a short description of the diagram at that stage and, if there were some parts of it that were discarded, why it was changed. The decision log documents are kept on the Google Drive, in order to make it easy for the team members to access them at any time, while being able to redact them. This artifact can be consulted in the *Appendix B.I*.

Trello - A Trello board was used to keep track of the tasks that have been created. Using different labels to represent the various stages of the process and different columns to represent different stages of the development, had a great impact of the team's ability to track down the completeness and correctness of the task.

Our SDM is also supporting traceability in a number of different ways which were described throughout the report.

Documentation

Due to prior projects the team has managed, the importance of creating a thorough documentation for the project was considered an essential. The team tried to conceptualize ways that a suitable documentation can be produced that will fulfil both the team's and *Struct*'s needs.

Some of the following statements' documentation can be observed along the whole report.

The first way of documenting the team's decisions was through various diagrams, which allowed the team to represent and understand the system in a unified way. It also enabled the creation of a proof of concept that could be discussed with anyone that might have doubts about the system.

The team included documentation inside the program's source code - namely code comments and summaries - which could help future developers to better understand the functionality that was implemented.

Further on the report, a section is describing artifacts that were discussed but not included, that helped the team to analyse other possible ways something can be achieved, evaluate their consequences and gather the best possible outcome for the team.

Another documentation type that was used, was through a decision log and Github issues to record any changes in the system which established a level of traceability that the team was pleased with. Further reading can be found in the previous section.

Using Google Drive the team kept a history of meetings with the client, advisors or fellow students in the form of transcripts that were used in a way that the team could go back and check in case there were any doubts about fundamental recommendations and discussions.

Tools and methods

The following topics will present the tools and methods that were adopted throughout the project's development along with a brief description about their importance to the team. Furthermore, additional details for some of these topics can be found throughout the report.

MS Visual Studio, C# (Web API, ASP .Net)

Visual Studio was used during the implementation, this was a requirement since Web API and ASP .Net are part of Microsoft's Framework.

SOLID

These principles helped to ensure that the solution's source code was clean, extensible and modular.

MS SQL Server Management Studio

Even though databases were not part of the initial requirements, the team decided to take advantage of it, so the usage of SQL Server Management studio was essential.

Github

This website was used to keep the most up-to-date versions for both artifacts and source code, with which traceability was obtained.

Rich picture

Although this artifact was recently introduced to the team, it provided a better grasp about possible ways other applications will be dependent on our system.

Workflow diagram

This diagram provided a fast and intuitive way to represent the current working procedure and how our implementation will make it more efficient.

Wireframe

Practiced in the previous year, this artifact proved to be handy in depicting a potential user interface.

Experiments

Even though it was the team's first time implementing experiments, it gave an insight on some technologies, helping everyone to better understand any future implementation.

KPIs

The team applied this method in their process in spite the fact that it typically occurs when doing Business analysis, for measuring the effectiveness of the developed solution.

UML (Umlet)

This standardized modelling language was chosen due to the advantages it provides when analysing and documenting the behaviour of the system. For easiness of producing these diagrams, the team used a UML design tool named Umlet.

- **Use case diagram**

  To achieve a better visualization of the program's desired capabilities and features that can be presented to product owners and stakeholders.

- **Object model**

  From the OOP best practices, this diagram was created to illustrate in format that everyone can understand, real-world objects depicted in a model.

- **Domain model**

  To demonstrate how the above-mentioned diagram can be interpreted in a domain, all the fields were placed into entities and linked, establishing their cardinalities.

- **Base class diagram**

  This diagram was developed by the team to present the base structure of the system that would be later implemented in both Web API and MVC.

- **Design class diagram**

  The Design class diagram reflects a more in-depth view on the system's classes and the inner architecture of the application.

- **System sequence diagram**

  This diagram illustrates how certain tasks are performed between users and the system, regarding a single use case.

- **Sequence diagram**

  This type of UML artifact demonstrates what will happen inside the system once a call is invoked from a user to it.

Operation contracts

The team created this artifact when there was the need to reveal the effects of an operation to the system by describing pre- and postconditions, as well as defining changes of the state of the overall system.

Relational Database Model

All the relations that an entity is associated with were analysed and after that, a relational database model was created. One of the strengths of this model, was to ensure that our design is free from anomalies, after normal forms were obeyed.

Task board

An essential part of organizing the team's work was an electronic task board used as a visualizing management tool for following a task's path towards completion and achieving overall objectives. The team was able to stay on the same page, prioritize their work, review their progress and discuss project changes using the online board since it was accessible from anywhere with an internet connection. The board used during the whole project period is present in the *Appendix C.I.*

Kanban

The team decided to adopt this new method for them, to be used during the Development and Testing stages of the project after an accurate research amongst other suitable methodologies, analysing and adjusting the method with the purpose of accomplishing the settled goals.

CL structures

Some of the "flipped classroom" methods that were taught and used during first year's classes were revised once more and afterwards the team has selected several CL structures that can be applied in order to improve the efficiency and workflow of the team. The most preferred one was "*Boss and Secretary*" used for artifacts creation, then "*Round Robin*" was also picked whenever a discussion arises, "*Think Pair Share*" also contributed to generating new ideas. When the team began writing documents which were used later to compose this report, a structure where two members are working and exchanging ideas together was created.

Focus Areas

Requirements Gathering

After establishing a contact with *Struct* the team needed to set the project's direction by gathering the user requirements. An email with all the specifications was sent to the team, from which all the requirements were carefully assessed, and a meeting was held after to assure the clear interpretation of the system's goals. Their importance derives from the fact that they would be used later to guide the development of the project and to ensure the team is pursuing the right direction.

To accomplish the specified tasks, the team had to record the requirements in a way that is clear and well-defined for the whole team. The customers' needs were refined into Use cases, which would form a consensus on how the system should behave in detail, reducing uncertainty by avoiding misunderstandings that could be problematic in future phases. Furthermore, revising them constantly had a huge role in identifying some unforeseen issues, for instance, a button to logout manually was missing in the wireframe. The most important benefit undoubtedly was the fact that the team was able to identify numerous exceptions and alternatives of the main flow of events, before they became critical in the next phases.

Prioritization method

When the team evaluated the overall project's schedule, a differentiation between the requirements' importance needed to be done. The MoSCoW method was chosen to classify the significance of critical features to the system, as well as to help in prioritizing what is valued the most according to the user organization. Further details about MoSCoW method together with the prioritized use case list can be found in *Appendix D.I*.

Prioritizing the Web API was done after considering the priority of MVC, as MVC is built on top of Web API, the functions that allow some MVC use cases to work should have the same (or higher) level of priority.

START UP

Chosen SDM

For the System Development Methodology choice, it was decided that it would be advantageous to follow *Struct*'s approach for developing an information system. This decision would allow to adapt to new processes regarding a project management and at the same time it will make it easier for the company, should they want to continue to develop the project. There were also other aspects about *Struct* taken into consideration to make the decision to implement their project management flow, such as: their team sizes are very close to our team size so this way an already proven to work aspect of working was used,

and some parts of the process can be improved, which they might find useful and implement on their end. After the decision was made, the team made some research regarding their SDM and came up with some similarities with other SDMs, thus helping to understand them better and improve some of the processes, which will be further described throughout the report. Additionally, during a prior project some team members cooperated with *Struct*, where observations and analysis were done in order to provide some valuable suggestions on how their methodology can be improved. This exam project can perfectly contribute in demonstrating why and how those suggestions can be integrated in their current working process.

Startup Methodology

As previously mentioned in the Chosen SDM section, *Struct* follows the policy of conducting workshops during the Startup phase for better defining the user requirements and program's UI, which the team compared with *RAD*'s first two broad phases. Another similarity was found in *Prototyping* SDM, where *Struct* creates a basic prototype with the purpose of showing it to the user so everyone is on the same track about the project.

Despite the fact that the requirements planning phase and the design workshops in *RAD* are held in different time periods, *Struct* is doing them combined. The reason for it is that *Struct*'s preferred way of gathering requirements, both UI and system requirements, is through workshops where the company communicates with the client through a more practical way to get the best common agreement. These are participation intensive and normally the client is closely involved, which allows for better communication resulting in improved documentation that both parties can understand.

The team evaluated the second similarity and determined that the way *Struct* applies it could be adjusted. The intention of presenting a prototype to the client works for them, since they always start with a base project and implement minimal functionality, that can be demoed for users' feedback. This approach would also benefit our process, but due to time constraints on the project the team decided not to pursue it, instead it was established that experiments would be considered. These will still allow to get some functionality and to acquire feedback for it.

Risk analysis

Identifying the risks the system is more prone to have, was considered a major area of interest, being that it is the cornerstone of developing a good and reliable system.

Software risk analysis

The team had considered three areas in which the system may be exposed to failure and tried to formulate plans for each of them in order to overcome any problem that might arise.

The topic of major concern for the team was the risk of developing the wrong software, this can be caused by a poor requirements gathering phase and misunderstandings between the team and the client. Therefore, the team thoroughly evaluated the client's requirements, thus creating artifacts that illustrated the requirements into an easy to understand shape, such as a rich picture, use cases or workflow diagram. Subsequently, the artifacts were presented to the client and further discussion took place, leading to drastically reducing the risk of creating the wrong software.

Another risk factor that might have been present was creating an improper user interface. The team has foreseen this problem and came up with a wireframe that has been created to overcome the risk in this area. The wireframe has been presented to the client and modelled in such way that will fulfil their needs.
Gold plating represented another topic of interest for the team in order to assess the risk of the system. Filling the system with unwanted features would have been time consuming and unnecessary, therefore the team tried to overcome the problem by constantly requiring feedback from the user upon different artifacts. Doing so, the team has managed to reduce the risk of gold plating the software and focus on creating only the required features.

SDM risk analysis

The chosen SDM was composed of different parts from various well-known SDMs, which could cause that the wrong artifacts would be developed and applied to a not so appropriate situation, either by them being not useful for the purpose of creating the desired system, or by not having the necessary artifacts to ensure good quality of the system in general.

Even after obtaining the right artifacts there may be other issues, such as enforcing poor quality criteria, fact that would lead the team in a wrong direction, therefore the team would have to thoroughly document the quality criteria before the creation of the artifacts.

Another risk that the team may be exposed to would be not updating the artifacts after modifications have been made to artifacts related to it. As such, before performing any changes the team had to analyse the relevant actions in order to have a clear view upon the artifacts that would be affected by the change, and modify them in the same order as they were created.

Concluding the analysis, the team also assessed the risk of deploying an unwanted feature which would conflict with the rest of the system. This could be diminished by creating backups before deploying the system ensuring there was a possibility to go to a previous working state.

Rich Picture

After getting a general idea of what the project consists of, the team has created a "Rich picture" in order to visualise its objectives.

This drawing has the purpose of highlighting the main components of the system and the way it interacts with other existing systems. The artifact below helped everyone involved in the project to take a consistent view of the problem situation, thus evaluating its consequences. Moreover, it helped to discuss and identify the beneficial users and other participants in meetings together with the client. A brief of what this artifact is representing can be found in *Appendix E.IV*.

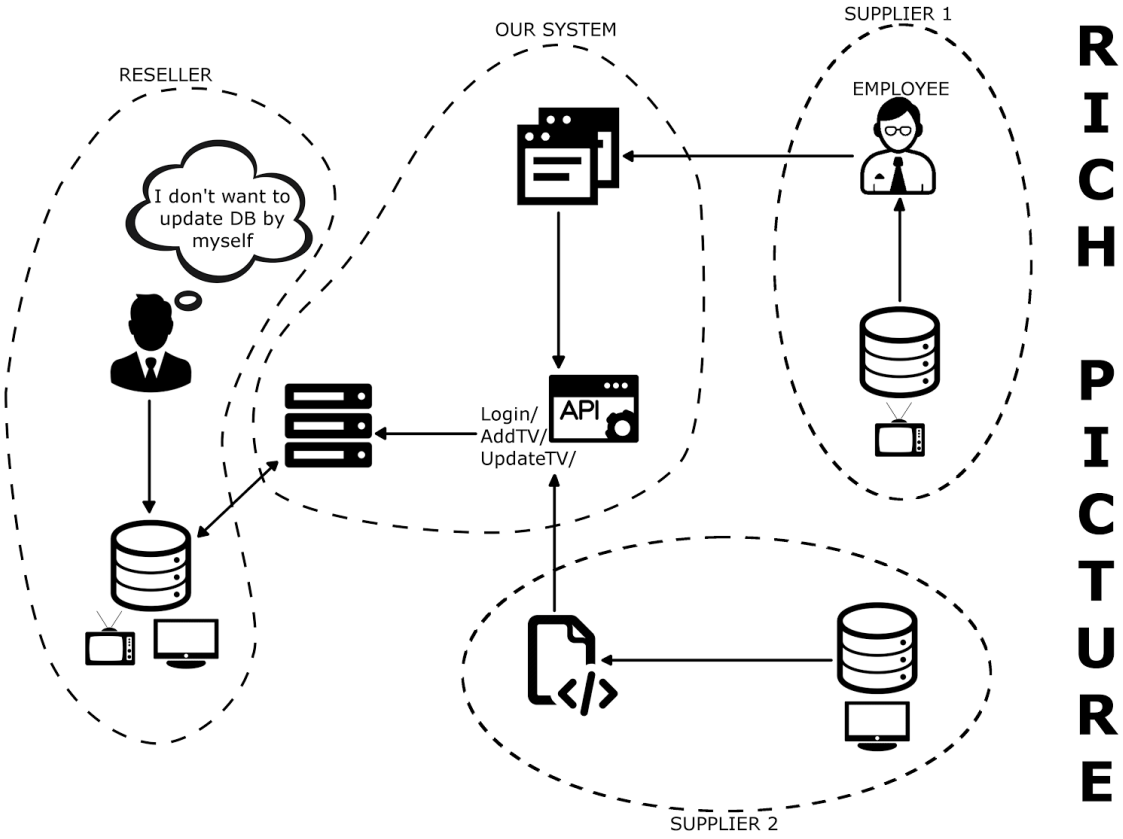

Figure 2.1 - Rich picture

Workflow diagram

A workflow diagram was worth considering before starting to work on the actual system design. This diagram was used for understanding purposes, namely elaborating more on our client's business, and showing the difference between the current and the desired way of working. With it the team defined a series of steps in the reseller's process that must be executed consistently, thus getting a general overview of the business process and afterwards consulting the stakeholders about its accuracy.

Each step is represented with an abstract shape like a box. The steps are connected with dotted arrows that indicate the flow from beginning to end.

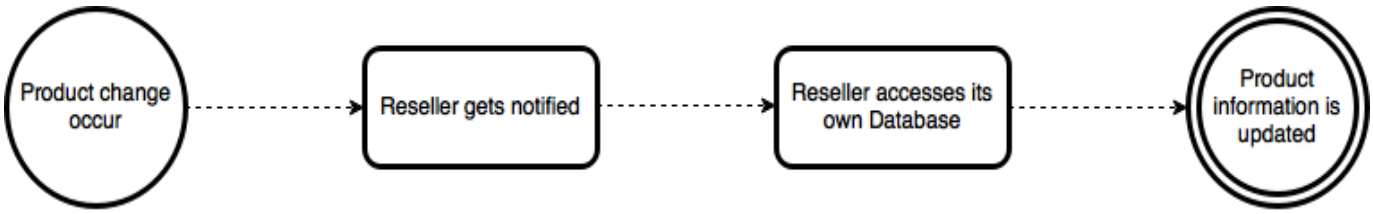

Figure 2.2 - Workflow diagram before implementation

As *Figure 2.2* indicates, a reseller will be needed as a mediator between the changes that the supplier provides and the reseller's database that needs to be updated with the last items' specifications.

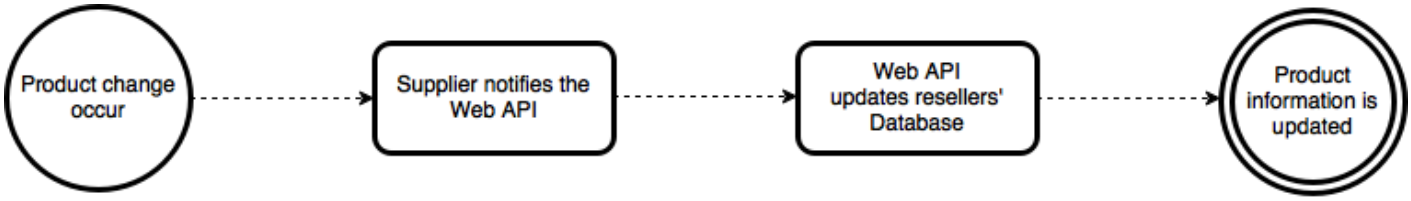

Figure 2.3 - Workflow diagram after implementation

However, *Figure 2.3* depicts the after state of their process, where a reseller's manual work would no longer be needed. Instead, there will be a Web API which will receive the updates and pass them simultaneously to all different reseller's databases.

In both figures the end event will be the same, resulting from the prior actions - the item's specifications will be up to date.

Use cases

For this project, the team was given a lot of specifications which were first interpreted as Use cases and later expanded by Use case descriptions, to help capture the user requirements in a more detailed way. Out of these, in the report, two will be emphasised together with the reasons that brought the team to such choice. The use cases will be arranged by their numerical order taken from the common document with all the use cases (*Appendix E.I*). Considering the two main parts of the application, i.e.: use cases related to the MVC part will have the prefix "UC1.", while the ones associated with the Web API respectively will begin with "UC2.".

More examples of the same approach can be found in the above-mentioned appendix.

Note:
The possibility that the actor could at any time go to a different page or close the browser resulting in cancelling the action was not included in the following descriptions.

UC2.1 - Login
Short description
A user wants to start using the Web API actions.
Primary actors
• ***Bob***

Secondary actors

- *None*

Overview

Bob wants to use the Web API, but he needs an authorization token.

Precondition

- *None*

Primary Scenario

Actor	System
1.Bob prepares a request with his username and password 2.Bob sends the request 7.Bob receives the token	 3.The system checks the credentials 4.The credentials are valid 5.The system creates a new token 6.The system sends the token to Bob

Alternative scenario

Actor	System
	4a.The credentials are not valid 5a.The system sends an error message to Bob

Post-conditions

- The new token is stored in the system

UC2.3 - Update items

Short description	
A user wants to update information about one or more existing items.	
Primary actors	
• ***Bob***	
Secondary actors	
• *None*	
Overview	
Bob wants to change information about one or more items already present in the system.	
Precondition	
• Bob owns a token.	
Primary Scenario	
Actor	**System**
1.Bob prepares a request with items' information 2.Bob sends the request	3.The system checks the items provided by Bob 4.The items are all valid 5.The system performs the action 6.The system sends a "200 OK" message to Bob
Alternative scenario 1	
Actor	**System**
	4a.At least one item is not valid 5a.The system prepares a "400 Bad Request" message, containing the error details 6a.The system sends the message

Post-conditions
• The system updates the item(s)' information

The team has selected the two use cases that stood out the most. The reason for this choice was that the UC2.1 - "Login", was the most different from every other because it created an interaction with the system that is developed by default. The remaining use case, UC2.3 - "Update items" was chosen due to its similarity but elaborate complexity to the unselected use cases. The complete list of Use cases' descriptions can be found in the *Appendix E.II*. Next chapters will explore the specific way these use cases were accomplished.

Scenario

The next step consisted in going through the Use cases' descriptions and producing a use case instance, better known as Scenario, for the Use case with most alternative paths, which brought a real value of representing the different branches in which this Use case can be performed. The team previously has created actors (only one of the actors is referred below) which made the Scenario more personalized, thus increasing the understandability for the reader. The creation of the Scenario UC1.3 helped the team to foresee user tests and their expected outcome.

Scenario: UC1.3

Alice needs to change the specifications of a television that was inserted into the database with some errors (she is already logged in). She searches for the item in the list, and clicks the corresponding "Edit" button. She will then be redirected to an editable page with the current details of the item where she fixes the wrong information, but inserts invalid data without noticing. Alice then clicks to confirm, but the system displays an error message, explaining the mistake. After fixing the mistake, and confirming the changes she is again getting prompted with a request for confirmation. Before clicking on "Yes", she notices a typo, so she cancels the request, fixes the problem and confirms again, this time with success.

Wireframe

The team chose to conceive a wireframe mainly to get a general look on how the user interface should be designed and to create a greater level of understanding upon what is being built. It also allowed for early feedback to be received from the client, which optimized the workflow by reducing the possibility of requiring some changes further in the process. Throughout the whole project, wireframes were used to give the team and the client organization an easy overview about the final product's appearance.

An alternative to the wireframe would have been a Mockup, but the latter is more focused on the look of the user interface and not about the elements, thereby there was no necessity of creating it.

The wireframes created can be found in the *Appendix E.III*, including wireframes that were considered for the further development of the process.

Experiments

One of the tools used during the Startup phase was experiments, which were meant to rapidly develop a small system that wouldn't actually be put to use in the final product. However, its objective was to improve the quality by investigating the parts of the final system that the team was lacking experience on.

Before starting the experiments, it was necessary to define in a detailed way what they would be about, similar to a requirements list, then it would be necessary to explore how to achieve them. Eventually the team began working on the actual development of the experiments. In some cases, there could be the need to have more iterations between researching and experimenting, until the team gets satisfied with the knowledge attained.

From the experiments the team obtained more knowledge about the researched fields, which resulted in a more manageable designing phase, also yielding better outcomes. Moreover, it reduced the probability of having to review parts of the design after development has started because of unnoticed flaws.

The team has performed an experiment focused on authentication on Web API with the purpose of exploring and testing the API's template login system, specifically what does it offer and what level of security it grants. The requirements were as follows:

- The experiment will allow to register new users;
- The registration of new user can only be performed by an "Admin" user, that is formerly stored into the system;
- The experiment will allow only registered user to execute the Web API's actions;
- The experiment will have one Web API action which will return a different result depending on the user requesting to perform it.

By performing this experiment, the team realized that the system will probably use two databases, in order to separate the sensitive data about the user from the inventory items.

High-Level Design

Posterior to the Startup phase, the team began working on the High-Level design.

This focus area concentrates mostly on collecting knowledge regarding the abstract, high-level view of the system, thereby presenting how the major pieces relative to the application will interact collectively.

Before moving forward, it is worth mentioning, that even though the following decisions towards the system's general environment and architecture are usually made during the High-Level Design, at the beginning of the cooperation with *Struct*, the team was given specific requirements about the system's general environment and architecture, such as:

- The application will be composed by a Web API component providing actions to manipulate data and a ASP .Net MVC element built on top of Web API, presenting a graphical user interface to a human user;
- The system will have to handle a database provided by an external source.

To comply to the initial strategy, the team moved their focus onto finding a methodology that would be a good analogy for the *Struct*'s High-Level Design process. The choice resulted in an *Object-Oriented Methodology*, more specifically - *Object-Oriented Analysis and Design*. Analysing and designing the system by applying object-oriented concepts perfectly matched *OOAD*'s specifications. Approaching the problem domain in the form of objects made the design of the system's architecture more coherent. The team firstly identified the components of the system, analysed and classified design patterns to determine what components would be used repeatedly or would share characteristics and then arranged those components based on similarities and differences. After performing this analysis, the team began modelling the system using UML diagrams.

In addition, the entirety of the artifacts may be displayed to the eventual users of the system, since they are straightforward and uncomplicated.

The proceeding subsections will include the artifacts produced at this stage, as well as their corresponding contribution to this design phase.

Use case diagram

The rationale behind creating a Use case diagram from the list of Use cases, generated during the Startup phase, can be seen in the following section.

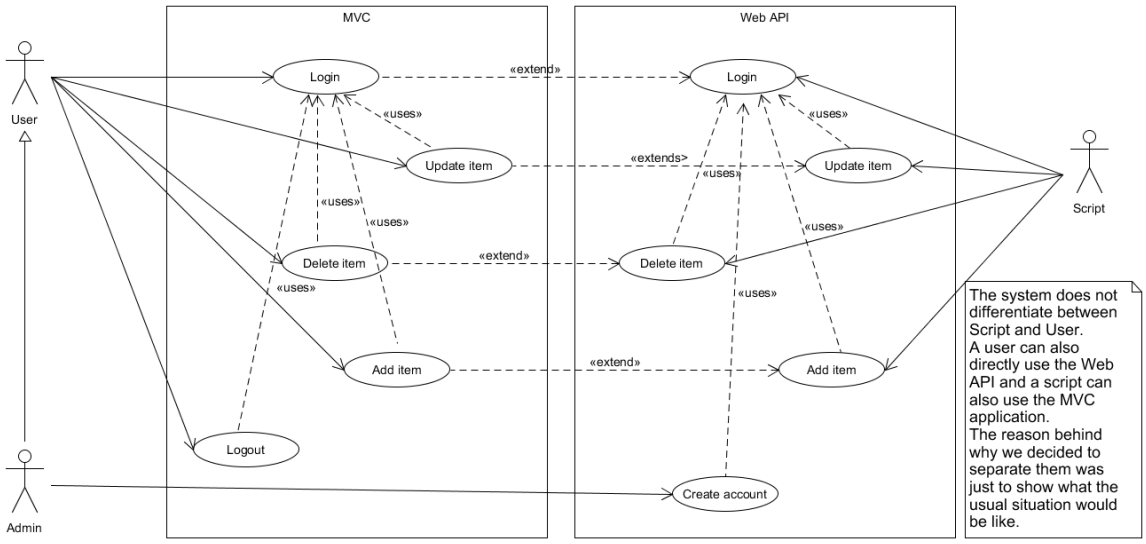

Figure 2.4 - Use case diagram

While analysing the Use case descriptions, the team spotted the disadvantage of not being able to provide a simplified way for the client to imagine the abstract operations in the system. Thereby the team began working on a Use case diagram. Its purpose was to give a visual representation of the Use cases which helped to depict the involved actors, and the interaction between them. The diagram has enabled easy communication with the customers, which contributed to discarding some possible flaws in the Use case descriptions. The team also used a notation that is not very diffused, but that perfectly suited the case. This explanation is as follows:

The "uses" arrow is drawn from a Use case X to another Use case Y to indicate that the process of doing X always involves doing Y at least once (although it may involve doing it many times, "at least once" is the only relationship guaranteed by this symbol).

KPIs

Less time consuming

Why measure?	This KPI's purpose is to measure how much time the reseller is saving by using our solution
How measure?	We will measure how long it will take them to collect data related to changes given by the producer. Then comparing the time spent with and without our solution.
Who is responsible for the measurement?	The measurement had to be done by a reseller who used the old solution so that he can compare both.
Expected date for measuring	Not defined.
Expected values measured	The transfer of the updated data should be almost instantaneous.
Measure	(Measurement still to be conducted).
Plan of action in case the measure is outside the range of the expected measure	If that is the case, our solution is not working as intended so, the team has to figure out if the problem is occurring on the solution itself or it is caused by something external. From this, the team would research and implement a solution.
Responsible for action	The development team.

Figure 2.5 - KPI for "Less time consuming"

The team analysed the consequences that our system will have on the resellers' work by measuring the time a reseller would take to complete the action of updating their own database against the time it will take to update it without human interaction. After a thorough research the team focused on a specific KPI /*Figure 2.5*/, which was the one that the team found to be more relevant. With this artifact, a quantitative time measurement about the impact our system would have in the organization, was determined. Similarly, it assisted the team in the review meetings with our client, presenting our objectives and strategies about the overall system performance.

Object model

The Object model helped to get an overview on how a real-life object can be perceived in a model, this was achieved by revising the use cases and extracting the most relevant components from them. Moreover, by exemplifying the domain in individual objects benefited in the communication with the user organization, since real data is something that can be easily related to. From this comprehensive picture several discussions were raised of whether the cardinalities and connections between the entities made sense.

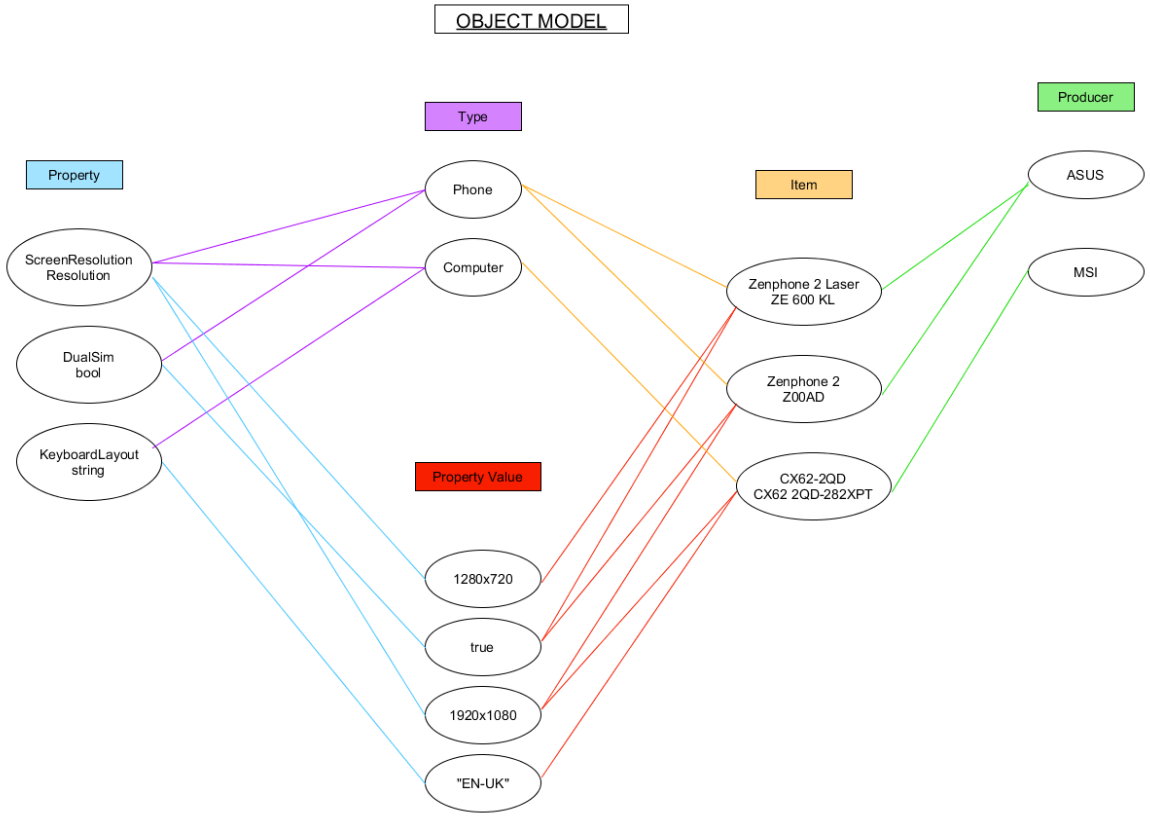

Figure 2.6 - Object model

With this diagram the team was able to clarify the essential components that would be part of the system, as well as giving a great outline to further develop additional artifacts.

Domain model

The domain model was easily derived from the Object model */Figure 2.6/*, where the conceptual classes were deducted from the various components that the system would operate with. This diagram additionally depicts the cardinalities between the denoted entities in the artifact, which made clear how the individual elements can be logically related. Every entity is comprised of attributes relevant to it, these were extracted from real-life models and previously demonstrated in the aforementioned diagram.

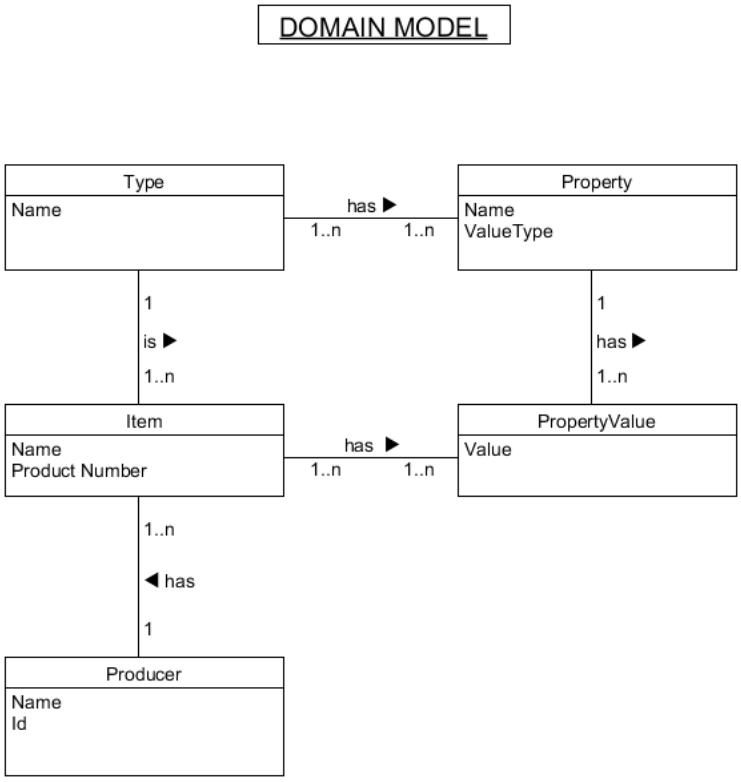

Figure 2.7- Domain model

The purpose of this artifact was to get the entire team in the same level of understanding about the domain, this helped to formulate a general idea upon which the implementation will be carried.

Base Class diagram

During the concluding stage of the High-Level Design, the team has decided to create a simplified version of a Design Class diagram, in order to explore the best possible way of extracting the classes that make up our system from the Domain model, thus reducing the time spent on the initial creation of the Design Class diagram. It also indicated the relationships among those classes, using annotations to give more information about the kinds of relationships. Another advantage of this artifact was reviewing the Domain model again, which allowed for fixing some of the diagram flaws at this early point of the design before causing problems in future stages of the process.

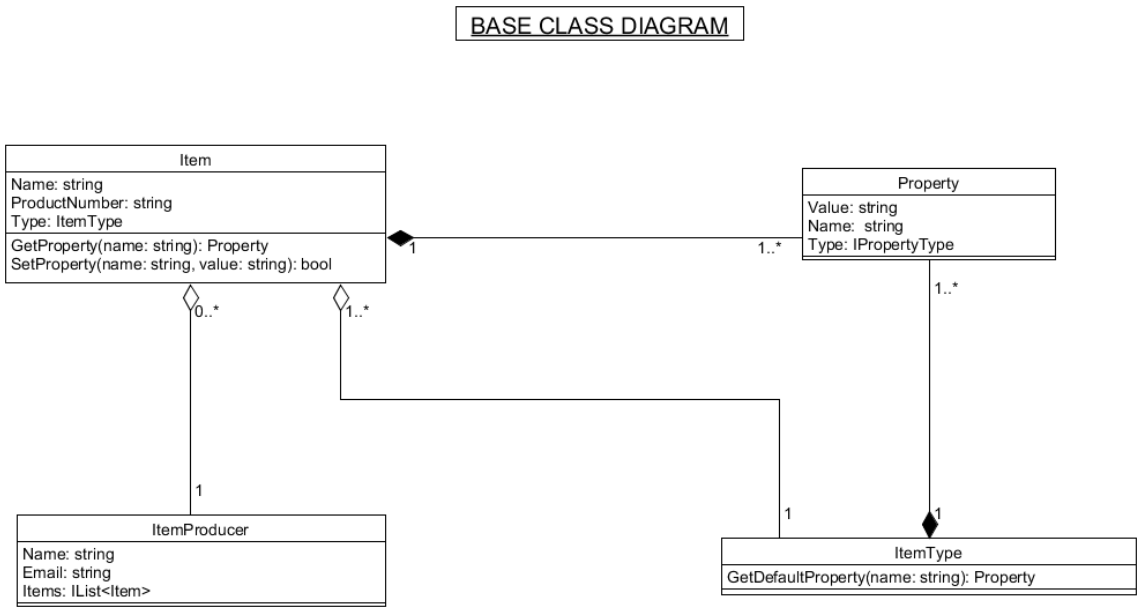

Figure 2.8 - Base class diagram

Initially, the team worked on a whiteboard, in order to research on the possible designs of the diagram. Several possibilities were discarded or modified numerous times, however they were recorded in a decision log, where the team has put all the versions of a diagram and explanation about its modifications. Documenting those, improved traceability and helped to kept track of the process that led us to the final diagram.

Design patterns

During the HLD phase the team looked into which Design patterns could be appropriate to implement. These patterns, if performed correctly can lead to improve maintainability of the system, given the fact that these are well described, and most developers practise them. Additionally, since these are providing solutions for common problems, they would allow the team to reduce implementation and design time by following the most proven techniques. The quality of the code is also largely improved by these in a sense that they enhance the reusability of code, as well as providing a loose coupling between different parts of the system.

The team chose to work with the following patterns: Repository, Façade, Factory, Singleton, MVC and Decorator. All of the before mentioned patterns will be further explained and detailed in the DCD section, furthermore, the combination of these patterns will result in a flexible system that will be easily modified and will have a reduced chance of system breaking bugs.

System Sequence diagram

The team needed to consider further the architectural design of the system before proceeding into the in-depth implementation analysis. There was the necessity to predict how the system will behave and to discover various responsibilities a class may need to have in the process of modelling our system. The decision of composing System Sequence

diagrams was advantageous for representing how the user and the system interact in a sequence of actions and interactions. The previously created Use Cases suggested system operations that could easily be discussed by having them simplified in a diagram form.

The following figures were chosen due to showing the most diverse actions with the system. Further examples of SSDs are provided in the *Appendix F.II*.

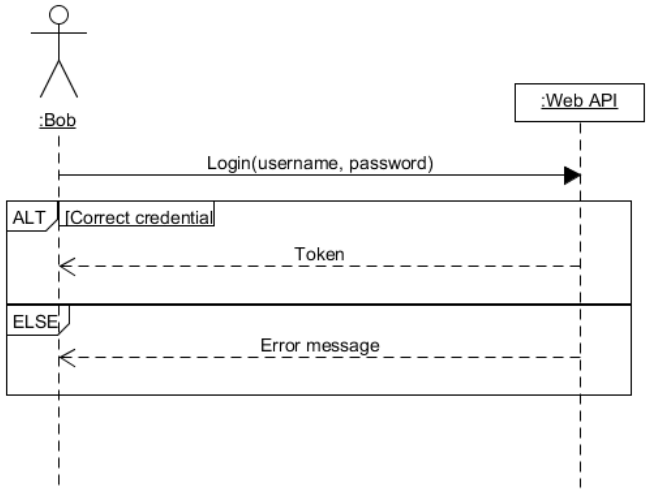

Figure 2.9 - SSD UC2.1 Web API - Login

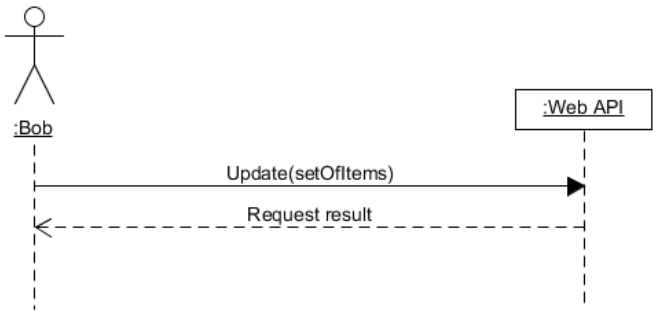

Figure 2.10 - SSD UC2.3 Web API - Update

These diagrams contributed to the process by giving a reflection of the decisions the team made about the design.

Low-Level Design

This stage of the project the team devoted to get some in depth understanding on how the system will actually be implemented and created, just before the actual development. Despite the fact that *Struct* does LLD by using artifacts only when it was fundamental to understand the connection between the current system and new features, to help them in the

thinking process of how it should be implemented, the team decided to follow their own suggestions created during the *IRL project* for using a more documentation extensive approach. In this focus area, as well as in Development, the team took only one module of the system to be described. The one analysed in the report will be the Web API.

Similar to the HLD, the methodology used - *Object-Oriented Analysis and Design* - was preferred due to the interconnection between the two design phases. The team remained creating artifacts commonly used in an Object-Oriented architecture since the programming architecture itself will be the same, this means that when the implementation phase starts the team only had to look at the artifact to create the application's structure.

SOLID

During this phase the system design was created taking into consideration the SOLID principles as general quality criteria. By applying them the team gained the ability of producing good and robust code and diagrams, also helping in keeping the system easy to maintain and extend over time.

SOLID stands for 5 general design principles, starting with the first two, Single responsibility and Open/Closed principles, which are enforced in our system by the Decorator Pattern.

The next one to take place is Liskov substitution principle, that is applied in our system by all our interface cases, which can be easily replaced by their subtypes.

Interface segregation principle is roughly followed in our system, except in the *IDBFacade* where not every class uses all of the methods provided by the interface.

The last one to be evaluated is Dependency inversion principle - applied to our system in several places, such as using *IList* instead of a *List* or in more complex situations such as connecting the system with the database using the *IDBFacade* interface.

Design Class diagram

Prior to the Development phase the team has decided to go one step further with the Base Class diagram by adding additional layer of information to it, resulting in a Design Class diagram. The purpose of creating this structure was to determine the different software specifications by representing all the classes, attributes, methods and relations among the objects to a level of detail close to the actual implementation.

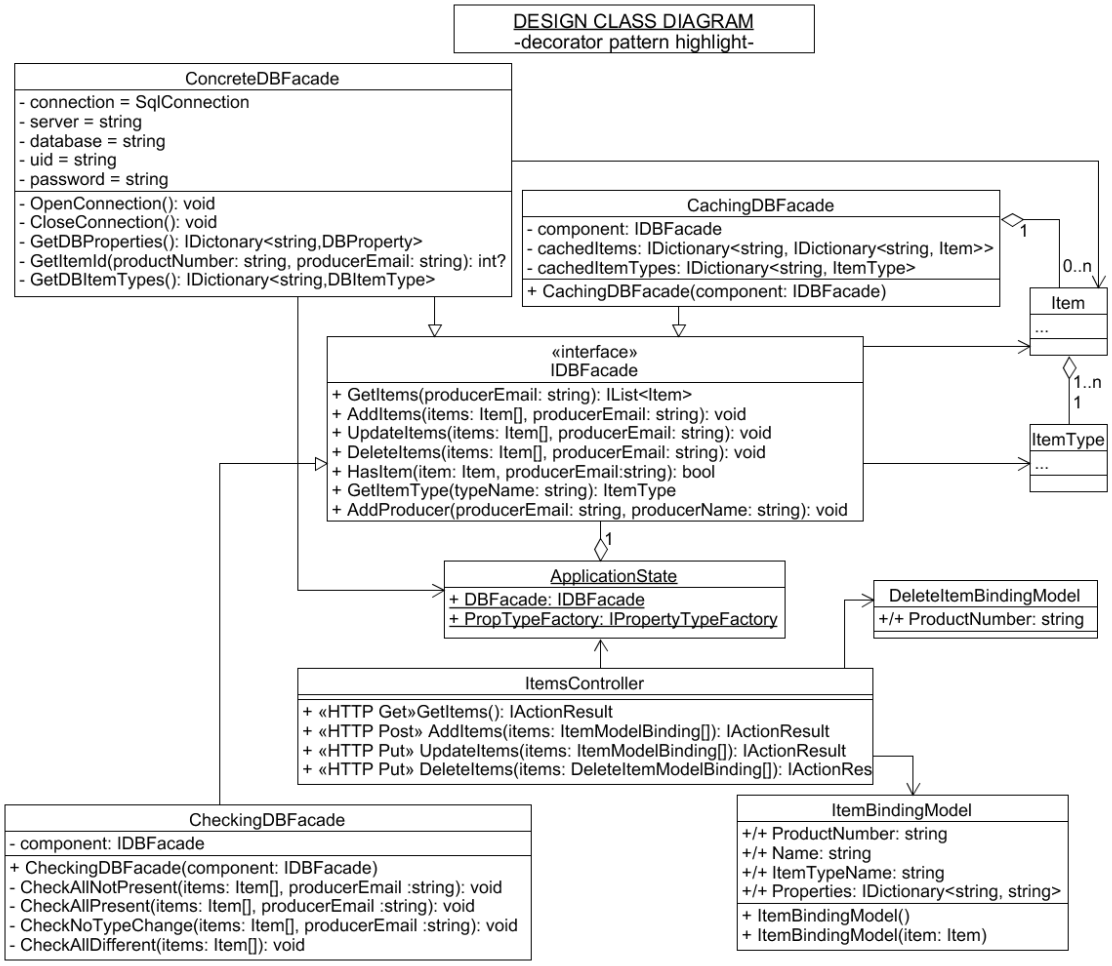

Figure 2.11 - Highlighted section of DCD (Appendix G.I)

Having the DCD helped to have a faster implementation phase by reducing the time spent on refactoring, since the majority of possible cases were already discussed and refined.

The Design Class diagram was built in steps. First the team identified the classes that will be present in the system, and their relationships. Then the team moved on by identifying first the public properties, and methods of the represented classes and after, the private ones. These steps were repeated multiple times, until the team was satisfied with the resulting Class diagram.

The team started by drawing on the whiteboard some possibilities, and passed through many different versions of the diagram which were catalogued using pictures, before starting to use a digital tool to draw them, versioning them via git.

The team used an unconventional notation to show the access level of C# properties in classes, since the access level to read the property can be different to the one for writing to it. The notation is explained as a note in the diagram, in order to allow people from outside of the team to understand it.

Design patterns implementation

In the Design Class diagram some of the pattern analysed in previous phase were applied to the classes. The design patterns used in this diagram are:

- **Model View Controller**

  Even though the system does not present any data to the user, hence the *View* part being inexistent, there is a clear separation between the *Model* - which represents the data - and the *Controller* - which defines how data is accessed and manipulated. The advantage of this separation is that it will loose the coupling between different parts of the code, making it easier to have different programmers working on separate parts of the system at the same time, moreover, it will be easy to perform any future changes.
  In the diagram the *Model* part is the one on the right, while the *Controller* is the one on the left.

- **Façade to database pattern**
  The Façade pattern is used in our system to provide an interface to the database, without having knowledge of the actual DBMS used.
  This allows to loose the coupling between the system and the database, making it suitable to use any type of database with the implemented system.
  In the diagram the façade to database is done through the *IDBFacade* and *ConcreteDBFacade* class.
- **Repository pattern**
  The Repository pattern is used to reduce the number of accesses to the database - which is resource expensive - by caching the items after a request to database.
  In the diagram the repository pattern is represented through the *CachingDBFacade* class.
- **Decorator pattern**
  The Decorator pattern allows to add new functionalities to already existing items, or to separate functionality across multiple classes.
  The usage of this pattern enables the team to improve the overall quality, by adhering to the Single Responsibility Principle, which states that each class should have responsibility over a single part of the functionality provided by the software, as well as the Open/Closed Principle which states that functionality of an entity can be extended without modifying its source code.
  In the diagram the Decorator pattern is applied to the classes implementing the *IDBFacade*. For example, there can be a *CheckingDBFacade* decorating a *CachingDBFacade*, decorating a *ConcreteDBFacade*.
- **Factory pattern**
  The Factory pattern is used for encapsulation, providing methods to get instances of a specific class without exposing the actual class implementation.
  This usually allows to have code not tied to specific classes, reducing the need for refactoring in case some changes are needed.
  In the diagram the pattern is applied to the *PropertyTypeFactory*, which provides instances of *IPropertyType*.

Database model

The Database model and the Design Class diagram were created at around the same period, laying at the same level of significance, in the sense that they are specifying the project's specific design infrastructure. The first mentioned diagram provided insight about the permanent storage data's layout which is based on the formerly created artifact, Domain model.

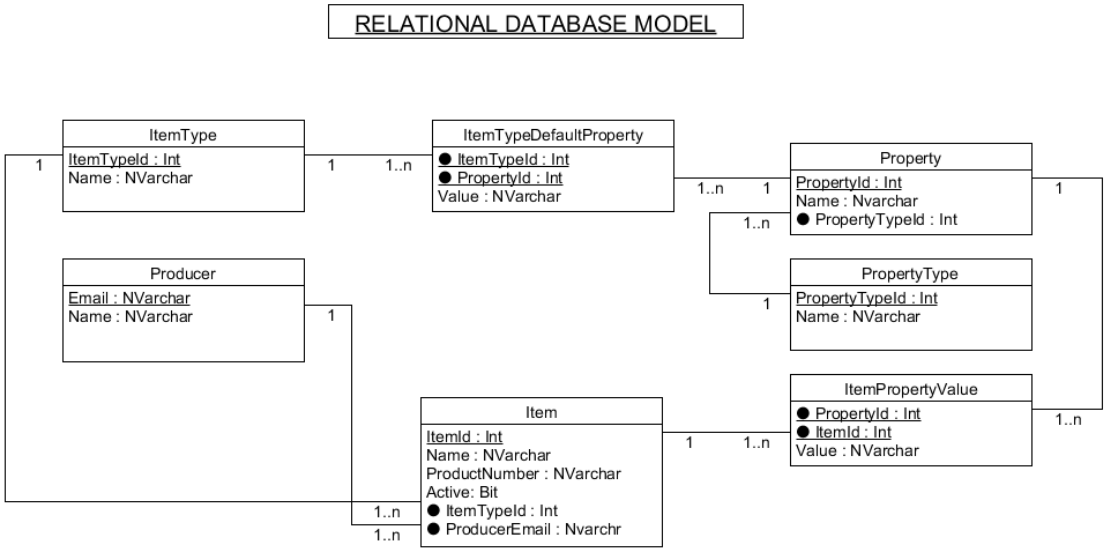

Figure 2.12 - Relational Database Model

Even though, a database is not included in the project's curriculum scope, one had to be created in order to fulfil the requirements gathered from the client. To obtain this model, the team started by looking into the diagrams created in the previous focus area to get a greater comprehension in terms of which entities would need to be permanently stored in a database. The following step was to analyse the relationships between the entities and figure out whether they made sense in an entity relational database perspective. These induced the team to verify the normalization state of the entities, which resulted in a number of changes to this model's design. Taking into account the normal forms, the team managed to eliminate the occurring anomalies and prevent future redundancy. As a result, this diagram was used in the means of illustrating the logical structure of the database.

Operation contract

During the creation of the SSDs, at the latest stage of the High-Level Design, the team considered the possibility to further analyse the system operations being invoked from input system events through Operation contracts. This artifact served the value of giving more detailed or precise descriptions of the system behaviour, since some of the Use cases didn't fully specify it. The team looked into how the internal state of the concepts from the Domain model might change assuming the preconditions before and postconditions after the execution of the operation.

Contract UC2.3 - Update items

Operation:	UpdateItems(items: Item[])
Cross References:	Use case Web API: Update items
Responsibilities:	The operation method is responsible for updating the information of items already present in the database. This method should first check if the updated items are valid, and if the user can modify the specified items. Afterwards the method will update the specified items in the cache and in the database.
Preconditions:	The user has a valid access token, and there is at least one item in the system he has access to.
Postconditions:	The Item objects in the CachingDBFacade were updated The items in the database were updated

The contracts assisted in providing fine-grained detail and precision of what the outcome of the operation should be, thus delivering an additional documentation to potential developers, who will be able to easily understand and further develop their ideas.
Moreover, the team only produced one Operation contract since from the chosen SSDs only "UpdateItem" was complex enough to have the need to further explore on it.

Sequence diagram

The team created a relation between a previously mentioned in the HLD artifact - SSD - and the following diagram, thus making a transition between two different focus areas. The relation can be described by mentioning that the Sequence diagram will explain in a greater detail how the actions, activated by an actor on the SSD, would impact the system.

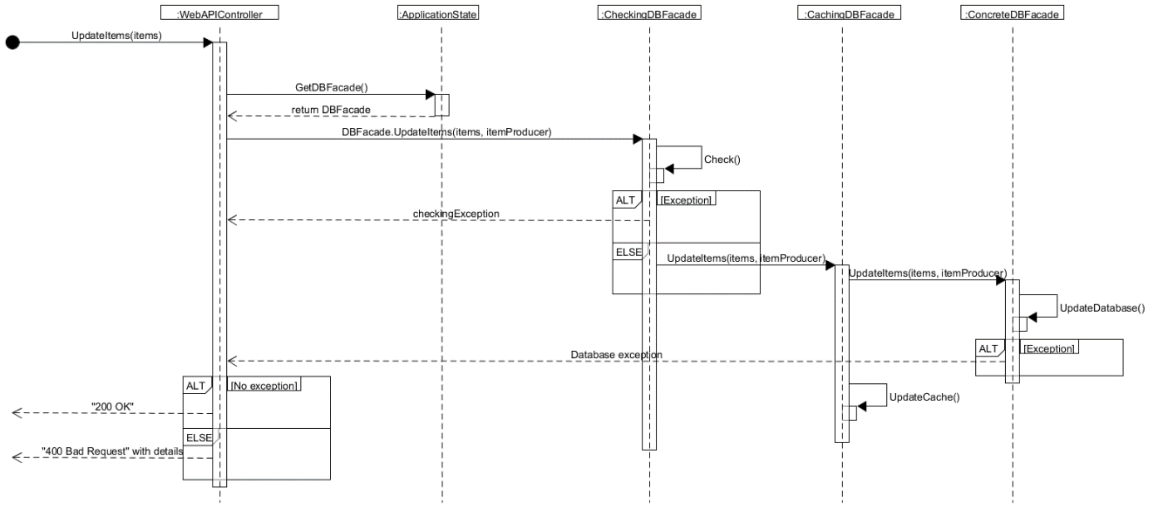

Figure 2.13 - SD UC2.3

The main advantage was that it gave an insight to the developers on how to actually implement the system and which calls would be made throughout a request. Furthermore, it

helped the team to analyse if what was done in the DCD was feasible in programming terms.

Development and Testing

When reaching this stage, the team decided to look into the *Iterative model methodology*, which was found to be appropriate for the team's needs, applying it to the following focus areas by firstly adjusting it. One of the adjustments was that the team started the Development phase only after the LLD decisions were made, which goes against the standard methodology practice, more specifically - the Development should begin right after HLD phase. Another modification concerning the terms commonly used in the methodology was that the team refers to "*Analysis*" as a junction of system analysis and testing. This means that when the analysis was being held, the team would make sure the system is implemented as intended, keeping track of the modifications that would need to take place, as well as performing the necessary tests.

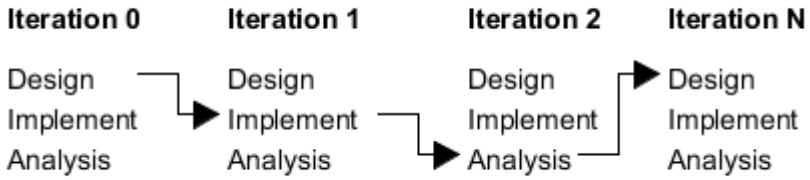

Figure 2.14 - Iterative methodology model

As it can be seen from the above diagram, the team will proceed with the cycle "*Design*" → "*Implement*" → "*Analysis*" for as long as there are modifications required in the system.

It was decided that the project would be developed into two different modules - Web API and ASP .Net MVC - which provided the ability of focusing in one of the modules at a time.

Regarding the Testing and even though *Iterative model methodology* suggests getting feedback each iteration, it was decided that this would cause too much strain to *Struct*, therefore the team agreed to only require feedback from them at the end of each module.

Using this methodology was very beneficial to the team's working process since it provided a reliable way of developing each module while assuring the quality and functionality of the system.

Development

The Development phase was the least expensive phase from the point of view of time consumption, since most of the architecture was already thought of from the previous LLD stage. This resulted in the team focus going mainly to translating the DCD into code, while implementing functionality. One other factor that contributed to the short time-span of this

stage was that Microsoft already provides default behaviour for a Login system, therefore saving the team time.

At the start of this stage it was decided that the best approach for code production was to have one of the team members' computer connected to a projector or a TV, so the actions can be seen by everyone, and have each member program a section of the code. This approach was created by the team by adapting the "*Boss and secretary*" CL structure and it helped to level every member's understanding upon the underlying technology. Even though subdividing the work between all of the team members would have reduced the time spent on the implementation, the group decided that since the implementation was not too complex, to use the method above. This second method would have brought further disadvantages as for example: investing more time in subdividing the tasks within the team and additional explanation would be needed to get everyone up to the same comprehension of the program.

After the initial implementation was done by having all the classes and functionality present in the system, the team created the database while referring to the RDBM, including as well all the necessary queries needed for testing purposes. The team then began to evaluate the newly created code, checking for places where some of the application components could be improved. When these were noticed the testing of new ideas would firstly be applied to the DCD, if no problems were identified in the new version of the diagram a new code implementation would be done.

Realising there would be no further modifications to the system, the team proceeded with testing the application, which will be additionally explained later in this section.

Kanban board

As a supplement to efficient process management during the Development phase, the team came across a tool used in the *Kanban methodology* that will facilitate the work and give insight in many aspects of the workflow. This tool is a board, mostly known as the Kanban board, which brought the advantage of having freedom when it comes to mapping the uniqueness to our team workflow. More specifically, it made the process transparent and helped the entire team to easily see the status of its work.

The avail of working with Kanban was the fact that it reminded the team that there is no right workflow or a right way to categorize client's requests. Kanban doesn't prescribe a specific path or procedure to evolve an item from request to completion. Considering that, the team adapted Kanban properties according to the desired style of working, aiming at reaching an optimal performance as a team.

The Kanban board was helpful in the sense of representing a shared visual[1] language which team members and stakeholders can use to rapidly communicate information in a way that is frictionless and transparent. Similarly to *Struct'*s Helpdesk platform, the board is accessible in a virtual form, which provided additional collaboration features for remote

[1] The term Kanban can be interpreted as follows: "Kan" means visible or visual and "ban" means a card or board.

workers and stakeholders, as well as feedback from customer reviews. Another benefit of this online source was the ability of generating a report about the history of every card, labelled as audit trail[2].

Like in *SCRUM*, the work was divided into tasks, where each of them was written on a card and placed on a column in the board, which represents different steps in the workflow. Before an item could be ready to move forward, it had to meet certain criteria. These constraints quickly illuminated problem areas in the flow that can be resolved after. A five-colour scheme was used to demonstrate the type of work. Further explanation about the symbolization of the chosen colours and the actual board can be found in *Appendix H.I.*

The way of placing the cards was according to the their priority, this eliminated any possible confusion about which card to be taken next - the highest card had the most priority. The team agreed upon keeping a card in a column as long as its task is completed, if the development of the feature is incorrect or uncertain, the card was moved back to the previous column, contrary to this, if there were no issues, the task was moved to the next pool.

The team observed how *Kanban development methodology* differs from *SCRUM* by evaluating its consequences. The following statements briefly mention what the team found as the most relevant discrepancies:

Unlike *SCRUM*, the tasks in the *Kanban methodology* are relatively larger in scope and fewer in amount. The work is concentrated around tasks, instead of the successful completion of a sprint. And the main idea is targeted to scaling down the amount of work in progress (i.e. the team doesn't commit to too much work at once).

Testing

The next point in the process required a greater understanding of the system itself. After the team had performed deeper system analysis and design, likewise followed essential conventions for a good software architecture, a method to guarantee that the system is reliable enough was needed.

Upon prior analysis of the chosen and followed SDM by *Struct* during *IRL project*, the team became aware that the Testing phase was not broadly documented during their projects. To surpass the aforementioned obstacle, an adjustment in the methodology was required. The team continued researching into various information sources, however among the current system development methodologies there was none that suited the project needs for the Testing focus area. Therefore, the team established their own strategy of ensuring that the product works in accordance with its specification, without undesirable side effects when used in a real development environment. As the possibility of expanding the software application in terms of complexity and components exists, for instance, when being introduced to different platforms and devices, it was of great importance to have a testing methodology that would assist in delivering the highest possible quality of the program at the end of a project, as well as securing it well enough. For this reason, the team aimed their attention on the functional testing part of the application.

[2] A system that traces the detailed transactions relating to any item in an accounting record.

This phase began with verbal discussion to assure what is required and how it should be documented to ease the technical part of the collaboration with the client. The agreement for documenting the software tests grew stronger when facing the need to provide documentary evidence of exactly what was tested. In that scenario the team included test cases in their testing methodology.

The tool introduced during the Development phase for mapping out the process steps using Kanban cards was applicable to indicate what should be covered in tests and to track these throughout the project lifecycle. The other benefits that the Kanban board carried were specified in the previous section.

As previously mentioned the team relied on specific test documentation in the form of Test cases, thus examining possible scenarios for optimal software quality. The latter were gathered in a table, together with pre-conditions, a summary of what the respective test case is going to do and expected results. Every test case was identified by a unique number (*i.e.: #3*) which can be easily referred to the corresponding test in the code, those unique numbers were present in the code, surrounding the test section which was relevant to the case. The reason it was decided to write Test cases was because it helped in the process of identifying defects in the code, they uncovered some bugs in the system and improved the overall quality of the software.

The team proceeded by doing the actual software system testing, i.e. performing Unit tests, as well as considering integration tests further in the process. Unit tests took place after code has been produced, thus ensuring what the system functions according to the design and functional specifications. Having all the tests passed enabled for applying the same conditions to refactored segments of code.

The table illustrated below depicts a portion taken from the created Test cases throughout the project's timeframe. More examples of likely Test cases can be seen in *Appendix H.II*.

CachingDBFacade

Ref №	**Test scenarios**	**Test Case**	**Pre-condition**	**Expected result**
...				
#4	Check AddItems Method	Check if the method is adding the items in the cache even if the specified user had no entries	The added items are not duplicated	The list returned by GetItems returns the newly added items

#5	Check DeleteItems Method	Check if the method is removing the items for the specified user from the cache	The aforementioned items for the specified user are present in the system and not duplicated	The list returned by GetItems method does not contain the newly removed items
...				
#7	Check UpdateItems Method	Check if the method is updating the values for all and only the specified items	The items passed as arguments are equivalent to already present items in the system for the specified user	The next GetItems method call will return the same amount of items as before and the modified items will have the values updated while the others remain the same

A side effect of following this method while conducting the testing focus area was providing an evidence why above-mentioned testing approach might benefit *Struct*'s future projects.

Fiddler

As part of the Testing stage, the team decided to use a tool named Fiddler in order to test Web API Controller, which would have been way harder or even impossible without it. This impossibility comes from the fact that the Web API Controller is receiving and processing network requests. The tool was mainly important while testing the server-side authorization, as well as if the parsing of objects sent in the request body was being properly done.

To entirely test the authorization method the team performed the following tests:

- Check if a user with an access token can perform any action.
- Check if a user with registered credentials can request an access token.
- Check if a user without an access token receives the "Unauthorized" response from the server.

Afterwards, the team proceeded to test all the remaining functionalities provided by the system, and compare the results with the expected values.

All the above-mentioned tests were stored in a text file in a way that will allow for future tests to be completed faster. These can be seen in the *Appendix H.III*.

Deployment and Maintenance

The team initiated Deployment and Maintenance stages by considering an investigation amongst existing methodologies that can be applied, thus benefiting the project. This was helpful to get inspiration from various techniques that the team structured in order to create their own methodology. The consequence of following this approach is that since it is not a proven concept there is a higher risk of getting mislead, thereby inducing the team on a wrong path for the process. Nevertheless, taking into account that the newly created methodology was based on well-defined and widely used SDMs, it helped to guide the team throughout the Deployment and Maintenance areas. Further explanation will be provided in the following sections about the choices and procedures taken during the last stage of the project.

Deployment

Even though this project's deployment responsibility would belong to *Struct*, the team developed a plan that depicts our intentions should the actions would have been our obligation.

The first thing taken into consideration was that the system would have to interact with existing databases, this would require modifications on the parts of the system that will directly interact with them. For this reason, and since this can result in several problems, a backup of the database's current state should be created, so if anything goes not according to the plan, a rollback can be used, allowing the old system to still function properly. Another concern the team had was about how to deploy system updates, which can be resolved by following *Struct*'s approach, using *AppVeyor* to automate and improve the deployment experience. The deployment itself will be comprised by several stages taking into account that the project is already split into two modules. From these, the Web API service would be deployed first, and secondly the ASP .Net MVC application would be able to be deployed being that it depends on the previously mentioned Web API. The last step of this stage would involve user training, which considering the dimensions of the system can be achieved by writing a user manual. Given the case that there is a need for additional explanation, workshops can be planned and executed in the user organization.

Deployment flow diagram, as can be assumed from its name, was created to visualize the deployment plan of the whole system. This was used to graphically map each step of the process and to have the final user confirmation on the defined deployment plan.

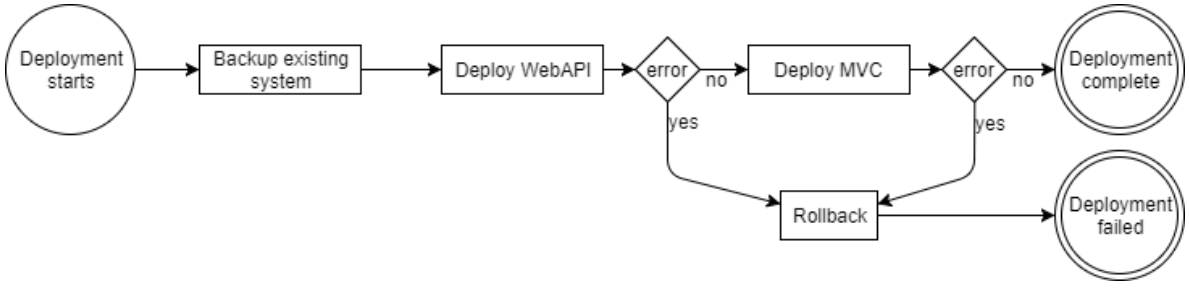

Figure 2.15 - Deployment flow

Maintenance

Proceeding from Deployment, the Maintenance stage aims at keeping the system at an optimal working state. On an initial stage, the maintenance would be mostly corrective, in other words taking concern about adding new features and fixing possible problems that could arise. These can be tracked and documented using an existing web-based system, such as GitHub.

The team also realised that two of the previous project stages, namely LLD and Development, some parts of the system could have been done better. With this in mind it was thought that the first maintenance to occur in the system should fix them, which can be related to predictive maintenance. This will help to further ease the process of maintenance.

The system was also designed with maintenance in mind, therefore its architecture is flexible to changes and can rapidly be adapted to use other components.

Evaluation

SDM

Evaluating the chosen SDM in the Startup focus area, made the team realize whether they would be able to assess the right decisions further down the focus areas. The team feels like choosing to follow *Struct*'s SDM was one of the best decisions they could have taken, for it helped not only to develop a project designed for them with their own way of managing it, but also a way to demonstrate how previously made suggestions to the company's SDM can be put into action on a real project. Having the ability of observing the team's performance while using specific tools and methods, provides *Struct* with the opportunity of analysing them and deciding if anything will be worth implementing on their own way of developing a system.

One thing the team had concerns about was what methodology will be followed by the Deployment and Maintenance focus area. Since the team was adapting their SDM depending on the focus area, there was some research that had to be held, this resulted in the team not being able to find a proper SDM for our current system. Also, since *Struct*'s projects aren't too related to what was intended with the project, the team couldn't apply any of the company's methods.

The latter can be overcome by using feedback to verify, how the whole deployment and maintenance process worked. This will lead the team to find the main issues that can be then thought and reflected upon, in order to come up with an execution plan that will surpass and improve the current one.

Artifacts not produced

In this section an exemplifying documentation that was considered but not produced will be described while arguing about the consequences of not doing it.

BPMN

The reason for not creating a BPMN was mainly based on the fact that an artifact that could be easily grasped by the team and the end customer was needed to be presented during the workshops, therefore the team decided to use a workflow diagram depicting the client's business process. The latter does not include any pools and lanes which might lead to confusion or uncertainty if the person does not completely understand the BPMN artifact.

Interviews with users

Interviews with users were not part of the team's working process because, our client *Struct*, assisted the team in contacting the end users regarding the project by the means of surveys, which helped to gather the main vision that was subsequently shared throughout the workshops.

Workflow

Looking back, the team has managed to cooperate well, every team member has promptly fulfilled their duties within the group and shown professionalism and dedication for the project.

The only possible improvement would have been to withhold a better log about what was done during the day, in order to have accomplished a greater level of traceability. The current ways of keeping track of the tasks that were *On hold*, *In progress*, *Needs review* and *Approved* was by the means of using a Trello board. Although it provides activity information about, for example, the time when some tasks were started or finished, it doesn't do that in an easy to read way. This could cause some confusion about what was done in a previous meeting if some team member was not present.

The creation of a Gantt chart was a possibility the team didn't consider in the initial phases of system development. Although the team had a plan about what tasks should be performed, a plan specifying the time frame for each of them was never created, and a Gantt chart would have proven to be useful for this. A time schedule could have helped the team in managing better the work and also identifying possible delays, allowing to catch up with the schedule, without having to resort to much longer working periods during the concluding phases.

Blossom's Taxonomy

The following hierarchical model was used to classify the team's overall project evaluation into levels of complexity and specificity.

Create	Generating a new methodology while combining it with newly gathered information through the project helped the team to develop their own way of managing a system development process.
Evaluate	Justifying the team's SDM choice was helpful to get the team a moment of reflection on what can be improved.
Analyse	Determining how different artifacts were related to each other, allowed the team to make interconnections between focus areas.
Apply	With the obtained information the team applied their previous knowledge to the system development areas that emerged this year.
Understand	Demonstrating understanding of the memorized facts and ideas was provoked by explaining these concepts to each other with the purpose of getting into an agreement.
Remember	At the beginning of the project the team started by recalling some of their first year's knowledge to figure out what can be reused in this project.

Figure 3.1 - Blossom taxonomy

Conclusion

The assigned project with the purpose of developing a Product Updating system was of enriching experience for the whole team, in the sense that everyone involved in this project has evolved both in a professional and personal level. During this project all the team members have learned and utilized several new tools that they have never crossed paths with.

Even though the project was not completely finished, since the team didn't manage to deliver the MVC system, they still feel like they produced a very good architecture and software as it was intended. A thorough analysis was made throughout every focus area together with the consequences of applying certain techniques and tools.

Overall, the team is very pleased with the final result of the project and is looking forward to collaborate further with *Struct*.

Bibliography

Books:

- "Beginning Software Engineering", Rod Stephens, 2015, ISBN 978-8126555376
- "Applying UML and Patterns", 3rd ed., Craig Larman, Prentice Hall, 2004, ISBN 0-13-148906-2
- "On track of the good system", Lene V. Andersen

Online Articles:

- Selection of Software Development Methodology
- SOLID principles
- Rich picture
- Bloom's taxonomy
- Asp .Net Authentication