Abstract

In this report you will be able to read about our scrum-based development of our program “Slice of Pie”.

// NEEDS MOAR

# Indholdsfortegnelse

[Indholdsfortegnelse 2](#_Toc343485239)

[Software Analysis 4](#_Toc343485240)

[Use cases 4](#_Toc343485241)

[1: Create new document 4](#_Toc343485242)

[2: Change the name of a document 4](#_Toc343485243)

[3: Delete a document 4](#_Toc343485244)

[4: Open a document 4](#_Toc343485245)

[5: Save a document 4](#_Toc343485246)

[6: Create a project 4](#_Toc343485247)

[7: Choose a project to work in 4](#_Toc343485248)

[8: Share a project with another user 5](#_Toc343485249)

[9: Insert picture to a document 5](#_Toc343485250)

[10: View a picture attached to a document 5](#_Toc343485251)

[11: Remove picture attached to a document 5](#_Toc343485252)

[12: Rename folder 5](#_Toc343485253)

[13: Move object in explorer 5](#_Toc343485254)

[14: Synchronize local project with server 5](#_Toc343485255)

[15: Add project from server to offline client. 5](#_Toc343485256)

[Domain model 5](#_Toc343485257)

[System Sequence Diagrams 6](#_Toc343485258)

[OfllineClient-Storage Diagram 6](#_Toc343485259)

[OfflineClient-Server Diagram 6](#_Toc343485260)

[WebClient-Server Diagram 6](#_Toc343485261)

[FURPS+ 6](#_Toc343485262)

[Software Design 7](#_Toc343485263)

[Class diagram 7](#_Toc343485264)

[Interaction diagrams 7](#_Toc343485265)

[Design patterns 7](#_Toc343485266)

[Software Architecture Documentation 8](#_Toc343485267)

[Architechture analysis 8](#_Toc343485268)

[Scenarios 8](#_Toc343485269)

[Factor tables 8](#_Toc343485270)

[Logical and deployment views (4+1)? 8](#_Toc343485271)

[Development documentation (Scrum) 8](#_Toc343485272)

[Testing, strategy and results 8](#_Toc343485273)

[HUSKELISTE: 10](#_Toc343485274)

# Software Analysis

## Use cases

We defined most of our use cases very early in the development, and expanded existing as well as adding new ones as the development progressed. Many of our use cases are very simple, and we chose not to make a bunch of complex diagrams for something that in fact is not very complex. We have a single Use case diagram which was made early in the development stage, and expanded near the end of the development. The diagram consists of 9 use cases and their logical orders, as well as how they impact the program. [[1]](#footnote-1)

Below is the final list of use cases we worked with in our project.[[2]](#footnote-2)

### 1: Create new document

The user wants to create a new document.

### 2: Change the name of a document

The user wants to change the name of a document.

### 3: Delete a document

The user wants to delete a document.

### 4: Open a document

The user wants to open a document he has selected in the explorer.

### 5: Save a document

The user wants to save a document.

### 6: Create a project

The user wants to create a project.

### 7: Choose a project to work in

The user wants to choose which project he would like to work in.

### 8: Share a project with another user

The user wants to share a project.

### 9: Insert picture to a document

The user wants to attach a picture to a document.

### 10: View a picture attached to a document

The user wants to see a picture that is attached to a document.

### 11: Remove picture attached to a document

The user wants to remove a picture attached to a document.

### 12: Rename folder

The user wants to rename a folder.

### 13: Move object in explorer

The user wants to move either a document or a folder to another folder in the explorer.

### 14: Synchronize local project with server

The user wants to synchronize his local project with the servers version of the project.

### 15: Add project from server to offline client.

The user wants to add a project that is shared with him on the server to his local client.

## Domain model

In the initial state of our program we had problems identifying the different domains that were relevant to the end-user. As a result of this our initial software architecture ended up being flawed, and that could have been avoided had we been better at identifying the different domains at an earlier stage.

In its final version[[3]](#footnote-3), it is kept very simple, but it helped us identify the structure of how the user sees our program, and to build a program based on exactly that.

## System Sequence Diagrams

We have mapped the interaction of our program using several SSD’s, they can be found in our Visual Studio Solution in the project “UML Slice of Pie”.[[4]](#footnote-4)

### OfllineClient-Storage Diagram

This diagram describes all functions the GUI calls through the controller to the storage, this includes creating, editing and deleting documents and projects.

### OfflineClient-Server Diagram

This diagram maps the connectivity between the offline client and our server.

When the user presses the Synchronize with Server button in OfflineGUI, the program calls SyncWithServer in ServiceController, who then handles everything. This includes making a connection to the server, sending each document in the project to the server, and then receiving the updated versions back, and finally save them to the local storage.

### WebClient-Server Diagram

This diagram describes how the WebClient makes calls to the server which then uses our storage to store the changes, much in the same way it is handled by the OfflineClient, but with everything being handled by the server instead.

## FURPS+

### Functional

The following points describe functions we want from our program.

* The ability to include both text and images.
* Allow the program to support being able to arrange the text files into folder and subfolders.
* Show a list of previous versions of the document.
* Synchronization of local content to a “server”.
* Ability to work while not connected to server.
* Changes to documents should be merged for everyone who has access to it.

### Usability

Our program to be run exclusively in a GUI we set up, without any need for command line interaction.

The training time for a new user to be familiar with our program and its functionality should be minimal, since we want a simple and intuitive gui.

### Reliability

If the system happens to crash the saved data should be kept intact via physical storage, which can then be loaded next time the program is opened.

### Performance

The program should be able to support at least 3 users being connected to the server the time.

### Supportability

The product code will follow the coding standards as well as naming convention that is commonly used in C#.

# //VI HAR INTET TIL NEDESTÅENDE

Implementation

We do not have any implementation requirements to our program.

Interfaces

Operations

Packaging

Legal

# //

# Software Design

## Class diagram

Our class diagram[[5]](#footnote-5) has been a central part of our software design, it has been what we have gathered around every time we identified an issue that would require us to rethink the way we handled data. A prime example of this was when a group member realized that sending every single document to the serer every single time we synchronized was a poor solution. When that was mentioned, we all sat down and brainstormed for other ways for this to be implemented. That ended up introducing our Project class, which would be a top level folder that held documents and folders. The Project class would also take over job of keeping track of who documents were shared with, as we chose to instead share projects instead of single documents. The issue of the synchronization was then instead handled by sending single projects at a time.

Over the duration of our development, our Class diagram has been reiterated over and changed many times, and we feel it provides great illustration of interaction and architecture of our program.

## Interaction diagrams

## Design patterns

* Everything regarding software-architecture
* Design patterns.

# Software Architecture Documentation

## Architechture analysis

## Scenarios

## Factor tables

## Logical and deployment views (4+1)?

# Development documentation (Scrum)

* Everything that regards to Scrum

# Testing, strategy and results

# HUSKELISTE:

Server og klient “burde” ikke dele storage.

Vi har ikke kastet os ud i omfattende exceptionhandling :<

Et sprint i Scrum skal aldrig kunne have sin varighed eller sit mål ændret under selve sprintet. Selvom vi har tilføjet nye stories til sprintlog’en under vores sprints, har vi ikke ændret selve målet med sprintet, så noget har vi i hvert fald gjort rigtigt.

User kunne sagtens kun være en string frem for at være en hel klasse, men det er både mere scalable, mere ’typesikkert’ (somehow) og vi bryder os mere om at have User som sin egen klasse.

” Top Reasons To Not Go Scrum #5: you have a fixed deadline, with a fixed set of requirements”

SKRIV: HVORFOR VI IKKE LAVEDE COMMUNICATION DIAGRAM, VI MENTE SYSTEM SEQUENCE VAR BESKRIVENDE/LET FORSTÅELIGE.

1. Use case model can be found in appendix 5 [↑](#footnote-ref-1)
2. For their full descriptions, see appendix 1. [↑](#footnote-ref-2)
3. You can find the final version of our domain model in appendix 2. [↑](#footnote-ref-3)
4. The System Sequence diagrams can be found in appendix 3. [↑](#footnote-ref-4)
5. Our class diagram can be found in appendix 6. [↑](#footnote-ref-5)