Abstract

In this report you will be able to read about our scrum-based development of our program “Slice of Pie”. Slice of Pie is a proof of concept piece of software that contains two clients, one being web-based, and the other being a local client. They both provide near the same functionality, which is allowing a user to create, edit and share documents through and between both.

We developed the program using the agile development method Scrum, which dictates rules about how and when development should be done. This report serves as documentation to both document the software we have developed, and the development itself.

# Indholdsfortegnelse

[Indholdsfortegnelse 2](#_Toc343489783)

[Software Analysis 5](#_Toc343489784)

[Use cases 5](#_Toc343489785)

[1: Create new document 5](#_Toc343489786)

[2: Change the name of a document 5](#_Toc343489787)

[3: Delete a document 5](#_Toc343489788)

[4: Open a document 5](#_Toc343489789)

[5: Save a document 5](#_Toc343489790)

[6: Create a project 5](#_Toc343489791)

[7: Choose a project to work in 5](#_Toc343489792)

[8: Share a project with another user 6](#_Toc343489793)

[9: Insert picture to a document 6](#_Toc343489794)

[10: View a picture attached to a document 6](#_Toc343489795)

[11: Remove picture attached to a document 6](#_Toc343489796)

[12: Rename folder 6](#_Toc343489797)

[13: Move object in explorer 6](#_Toc343489798)

[14: Synchronize local project with server 6](#_Toc343489799)

[15: Add project from server to offline client. 6](#_Toc343489800)

[Domain model 6](#_Toc343489801)

[System Sequence Diagrams 7](#_Toc343489802)

[OfllineClient-Storage Diagram 7](#_Toc343489803)

[OfflineClient-Server Diagram 7](#_Toc343489804)

[WebClient-Server Diagram 7](#_Toc343489805)

[FURPS+ 7](#_Toc343489806)

[Functional 7](#_Toc343489807)

[Usability 8](#_Toc343489808)

[Reliability 8](#_Toc343489809)

[Performance 8](#_Toc343489810)

[Supportability 8](#_Toc343489811)

[//VI HAR INTET TIL NEDESTÅENDE 8](#_Toc343489812)

[// 9](#_Toc343489813)

[Software Design 9](#_Toc343489814)

[Class diagram 9](#_Toc343489815)

[Interaction diagrams 9](#_Toc343489816)

[Grasp 10](#_Toc343489817)

[Controller 10](#_Toc343489818)

[Creator 10](#_Toc343489819)

[Indirection 10](#_Toc343489820)

[Information Expert 10](#_Toc343489821)

[High Cohesion 10](#_Toc343489822)

[Low Coupling 11](#_Toc343489823)

[Polymorphism 11](#_Toc343489824)

[Design patterns 11](#_Toc343489825)

[Software Architecture Documentation 11](#_Toc343489826)

[Architechture analysis 11](#_Toc343489827)

[Scenarios 11](#_Toc343489828)

[Factor tables 11](#_Toc343489829)

[Logical and deployment views (4+1)? 11](#_Toc343489830)

[Development documentation (Scrum) 11](#_Toc343489831)

[The Scrum Process 11](#_Toc343489832)

[Capacity planning 12](#_Toc343489833)

[Iteration planning 12](#_Toc343489834)

[Backlog story planning 12](#_Toc343489835)

[Sprint planning meetings and daily scrums 14](#_Toc343489836)

[INSERT PICTURE OF SCRUMBOARD AND US TROLO HIIHIHIHIHIHIH 14](#_Toc343489837)

[Definition of Done 15](#_Toc343489838)

[Sprint retrospectives and reviews 15](#_Toc343489839)

[Scrum Review 15](#_Toc343489840)

[Testing 16](#_Toc343489841)

[Unit testing 16](#_Toc343489842)

[HUSKELISTE: 18](#_Toc343489843)

# Software Analysis

## Vision

Our vision was the very first artifact we wrote, and looking back, it describes our final piece of software very well.

*We want our front-end for our clients to include an Explorer-like system, with some directories in which the clients’ documents are supposed to be, clicking on the document would then open it in a different window and there would be options to save and delete the document, along with other trivial document editing services. We want to be able to let several users use the same document at the same time without problems.*

*We wish for our proof of concept to be simple rather than overly complicated packed with features, but instead an intuitive working prototype, demonstrating the basic features, though as we progress we might want to add additional features.*

## 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

We have chosen to not create any additional diagrams to map interactions in our program, as we felt our SSD’s covered the interaction of core parts of program in enough detail to not warrant other diagrams.

## Grasp

### Controller

Our system actually has 3 controllers, one for the offline GUI called Controller, and another one also for the offline GUI called ServerController which is used for the sync methods to the server. The last controller is actually the SliceOfPieService, which is used by the WebGUI class in our system. Our controllers delegate the assignments needed to be carried out when something is queried from the GUI, they serve as a middle-layer classes between our program-logic and the GUI.

### Creator

The creator in our system is the class Storage, which is the one that reads from the file system and instantiates new projects with folders and documents inside, so it has all the initializing information that is needed to construct the objects and pass it on to the rest of the system.

### Indirection

We use the term of indirection, in the way that our GUI, calls to the storage class, but through our controllers. So if we change something in the storage, we don’t have to touch on the GUI parts, since we can just change some parameters in the controllers. That way we make sure that our classes are very loosely coupled.

### Information Expert

Our Expert is the same as the Creator, the Storage class. It holds all the information needed to actually create Documents, Folders, DocumentStructs and Projects, so the classes calls the Storage methods with the correct parameters and it creates the objects, because it has the information needed.

### High Cohesion

Our system practices high cohesion, we don’t have any classes with different responsibilities, rather they all have a certain job to perform, without overlapping with the other elements in the system, a class takes care of the problem or delegates the problem to the appropriate element to handle it. We think that our classes make sense and are well understandable by the way we have named them to represent what their responsibilities are.

### Low Coupling

We believe that our classes exercise low coupling. We have planned our classes in such matter that they have been encapsulated well, and are easy to change without having to change the entire system.

### Polymorphism

We use polymorphism widely in our composite pattern, since we often produce code where we just ask for an IFileComponent, because it could both be a Document, DocumentStruct or a Folder. This meaning that when we have a Folder and want all the children of the folder we can return a list of the folders children as IFileSystemComponents.

## 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)

## The Scrum Process

For our scrum process we used a very great website called [www.scrumdo.com](http://www.scrumdo.com), in which we started a free trial for the period, it was a great tool to help us manage our scrum segments, it gave a nice overview and we would definitely use a tool like that again.  
Scrum roles

Since we are 3 people in our group we decided to have one guy being part-time scrum-master, and another guy being part-time product owner and the last guy being a pig. We mainly used these roles when we planned stories for sprints and decided story points and so on.  
For reference we appointed Kasra to product owner and Christian to Scrum-Master, and we kept these roles throughout the entire project.

Capacity planning

We decided due to our rather late start of the project, due to other hand-ins in the parallel courses, that we would use a lot of time every single day on the project.

|  |  |  |
| --- | --- | --- |
|  | Days to work each week | Time allotted each workday |
| Christian | 6 days | 7-8 hours |
| Kewin | 6 days | 7-8 hours |
| Kasra | 6 days | 7-8 hours |

This rather extreme capacity planning was possible, because we all moved in together in the duration of the project period.

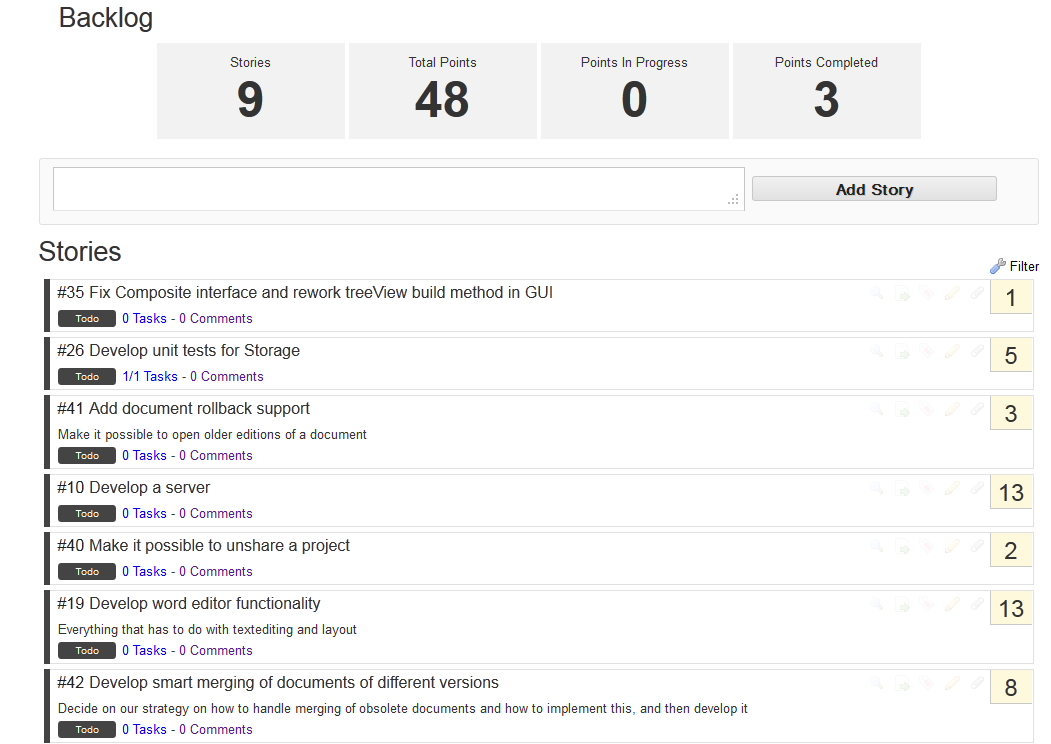
### Iteration planning

We decided that even though we only had two weeks from the time we began working on the project, we wanted to have multiple sprints to really try out the scrum method. Initially we planned on doing a total of 3 sprints, spanning 3 days each, starting on the 4th of December, ending the 13th of December and finally a release sprint spanning to the release date. However we had a one day break after each sprint moving the release sprint a bit.

Backlog story planning

We realized very late in the process that we didn’t quite write the stories exactly as they should have been, we wrote the stories as tasks that had to be implemented or diagrams to be made, where we in fact should have written them in the format like: As a \_\_\_\_\_ I want to \_\_\_\_ so I can \_\_\_\_ . Then when we put them into the sprintlogs we should create tasks and assign persons to each one. Instead we just created the tasks as stories.

Here’s and example of our backlog:



However, we used a feature on scrumDo called PlanningPoker, to decide how many points each story was worth, it worked as following:  
The scrum-master chooses a story and everybody chooses how many points they think it’s worth, then the numbers are revealed, and a discussion of maximum 1 minute is initiated where the story is discussed, finally the scrum-master decides the amount of points the story should have, and next story is voted upon.  
  
Burndown charts

ScrumDo has automatically produced burndown charts reflecting our progress as the time progressed.[[6]](#footnote-6)

As it shows, we actually kept a partly linear progression line. We started by adding every story we could think of, and gave them high amounts of points, which we reevaluated later as we progressed. At times we were not as good at setting the stories status to “Done” as we should be, and it shows in the graph certain places, but we believe it shows how our work effort progressed very well.

Sprint planning meetings and daily scrums

We tried to almost every morning, to start the day off with a standing meeting in several minutes discussing what we accomplished the previous day, and what we have in mind that we should be doing this day, and what might be troubling to these tasks.  
These meetings were always held under our big ScrumBoard as it can be seen in the picture below:

# INSERT PICTURE OF SCRUMBOARD AND US TROLO HIIHIHIHIHIHIH

In the start we very good at keeping the meetings very punctual but as we progressed the sharp meeting times got a bit more loose.

In addition to our stand up meetings we also had a meeting dedicated to each sprint the night before the day it should be started, where we planned what to do in the following sprint and played PlanningPoker again to reevaluate the points we gave the stories earlier.

### Definition of Done

Throughout the whole project, whenever we talked about the stories, we ensured that everybody had a consistent meaning of when we thought the story was done, we obtained this by talking the story out thoroughly, of course it’s not possible to get the exact same vision of the definition of done, as we experienced, but we tried to get as close as we could. In retrospective we should probably have written them down on each story, instead of just discussing them orally.

### Sprint retrospectives and reviews

After every sprint before we discussed the next sprint, we discussed how it went and what could have been better, and of course what we accomplished and furthermore what we didn’t. [[7]](#footnote-7)We all agreed that was a really nice way to catch up on what actually had been gone

Scrum Review

We all really liked to work in the scrum environment, even though sometimes we felt that scrum wasn’t really the right method to use in our case, because of different reasons:

Deadline-Oriented

Scrum is often not the best idea if you are having a fixed deadline with a strict requirement set, as we have in this case, that would call for a more waterfall-oriented approach.

No contact with stakeholders

Usually you would have a stakeholder representing the firm at the workplace, which we of course could not have, for the same reason we didn’t experience that much big changes to the project, which is one of the aspects that scrum is built for, being good at adapting to changes.

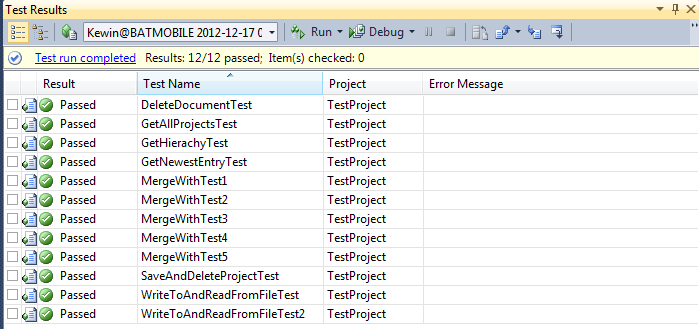
# Testing

## Unit testing

Before we started developing our program we had set up an architectural model of the interaction between the different modules in our program. And set up unit tests for the functions that were did not seem redundant to test ( getting and setting fields, adding and removing items from lists etc. ).

We have also overridden .Equals() methods for many classes to make testing easier, as it allows for easy comparison our custom classes. This was not done for all classes though, as comparison of fields like DateTime proved impossible because of the way the objects are created. When created by the DateTime.Now property the objects “Kind” property is set to local, along with having detailed information about milliseconds etc. These properties are not able to be saved and read when we write and read files.

As the development continued, and we neared the deadline with every passing day, unit testing was overlooked and instead intensive debugging was done to test corner cases, program states and more.  
This was done for all functionality in our program, in all later sprints, but we have very little documentation on this, and as a consequence, we cannot provide a satisfactory amount of unit tests for our project.



# 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:

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)
6. These can be found in appendix 4. [↑](#footnote-ref-6)
7. We have included the reviews and retrospectives for each sprint in appendix 7. [↑](#footnote-ref-7)