Rent It

Software Development in Large Teams with International Collaboration,

Second-Year Project,
Bachelor in Software Development,
IT University of Copenhagen

Group 12
Jakob Melnyk, jmel@itu.dk
Frederik Lysgaard, frly@itu.dk
Ulrik Flænø Damm, ulfd@itu.dk
Niklas Hansen, nikl@itu.dk
Jacob Claudius Grooss, jcgr@itu.dk

April 26th, 2012

Contents

1 Preface

This report is the result of a project on the bachelor in Software Development at the IT-University of Copenhagen spanning from the 2nd of February 2012 to the 23rd of May 2012.
The project was given on the fourth semester of the bachelor and corresponds to 15 ECTS-credits.

The project ("Software Development in Large Teams with International Collaboration") is described as centered on developing and implementing a rental service for digital media. The is goal is to develop a client-server solution backed by a relational database system.
Students work in the teams of four to six people and collaborate and negotiate with students from Singapore Management University (SMU) in English.
This specific project was done by five ITU students who collaborated with three SMU students.

The challenges of this projects were:

- Dealing with cultural gaps and time differences in communication with team members from Singapore.
- Sharing a balanced amount of information to have successful collaboration.
- Reaching agreements that both ITU and SMU students found satisfactory.
- Conducting constructive feedback on fellow students work.
- Constructing a client/server solution in C#.

Whenever we reference the Bibliography (found on page 21), we use the signature [1] to indicate such. [1] refers to the link to a blog describing "The Way of Testivus".

1.1 Scrum tracker, version control and service info

We use Git (at GitHub) as our version control system.

In addition, we use a free tool called Pivotal Tracker to track our user stories for the Scrum[1] development model we have chosen to use.

We have set up a server for publish a test copy of our service (and database). We use this service to automate tests of the client and the service interface.

git https://github.com/itu-bswu/RentIt

PivotalTracker pivotaltracker.com/projects/492063

Service address http://rentit.itu.dk/RentIt12/Services/Service.svc

Test Service address http://rentit.dk:9000/Services/Service.svc

Release database http://rentit.itu.dk (username: RentIt12Db — password: Zaq12wsx)

Test database rentit.dk (username: RentIT.dk — password: Vand22kanon)

[1] Further detailed in our **??** chapter on **??**.

2 Project overview

In this chapter we discuss our take on the project "Software Development in Large Teams with International Collaboration", what we feel are the important parts of the project, the must-haves of the final product and the assumptions we make going into our requirement specification.

2.1 Problem analysis

Not too many years ago, media rental of physical media was a lucrative business to be in[1]. The last couple of years have been hard on companies making their business in physical media rental[3]. This is, at least partially, due to the increasing popularity of companies like Netflix[4] making it easier to rent media digitally, thus enabling users to do it from home and not spend time going to the actual shops.

To create a media rental service that would be seen as interesting (if not competitive), it has to be:

- Easy to use[2]
- Price competitive[3]
- (Optionally) Offer an expanded array of services compared to other services.

A wide array of media rental services already exist for books, movies/films, music and other media, so there are many sources to draw inspiration from. In addition, media rental services do not necessarily have to be run by private companies. Some institutions (like libraries[4]) offer similar services for citizens.
A service does not necessarily have to focus on one kind of media (like Netflix), as evidenced by Apple's iTunes Store[7].

In addition to subjects concerning normal users of the service, the project description also mentions administrators. Administrators can upload, delete and edit movie information on the service. Because these types of administrator users provide content, we have decided to refer to them as `Content Providers`. We make this distinction because we have a user type we called `Admins`[5].

2.2 Assumptions and decisions

In order to narrow down the focus and requirements for our system, we make some assumptions and decisions in addition to the points raised in the problem analysis.

2.2.1 Choosing a service type

As described in the problem analysis, the existing types of media rental services can be narrowed down to a) free public library rental type and b) paid media rental. They can be very similar (depending on development choices) and both types present some security issues (user information, credit cards, etc.).

[1] Blockbuster LLC[2] is an example of a successful company in the media rental industry.
[2] Piracy is a major concern, and if the service does not provide something that is (at least) just as easy to use, people would rather be inclined to download illegally rather than pay for content from a service[5].
[3] Rarely a problem with piracy, but if one services provides the same amount of media, support and access, price is certainly a factor.
[4] Roskilde Bibliotek is an example of a danish library providing similar functionality[6].
[5] `Admins` are explained in further detail in our Design chapter (page 14).

We decided to develop a paid media rental system, as we felt it had more options (such as payment models) in terms of functionality that could be implemented. While adding payment options is not necessarily a core requirement, we feel we should design our system with payment options in mind.

2.2.2 Choosing a media type

At first we wanted to make a streaming service for TV shows. This could involve paying for a single episode of a TV show or for a full season.
After doing some research on what streaming would involve (compared to just downloading and saving a file), we changed the way we let users access our content. Instead of doing streaming, we decided to just let users download movie files and store them on their system.

In addition we changed our media type. While we felt it could have been more interesting to do TV shows (compared to other types of media), we decided to pick a slightly less complex system and instead focus on designing the service to offer movie rentals.
We decided on this less complex system, because we wanted to make a compromise with the SMU students[6] and still something that had a close relation with TV shows, but simpler.

2.2.3 Digital Rights Management

Digital Rights Management (DRM) is an issue we will most likely run into. While we may limit how long users have active rentals on the service, there is a technical challenge in making sure users cannot view the downloaded files after rentals have expired. We do not consider DRM functionality core in the service, but we do have it as an optional goal for our service.

2.2.4 Author rights

Author rights is another concept to consider. When we give `Content Providers` the rights to upload movie files, they may be able to abuse this by uploading files they do not have author rights to. We do not consider this a central focus point in our system, but in order to ensure a great quality service in a broader perspective, some sort of validation of uploaded material should be considered.

[6] Described in our **??** chapter on page **??**

3 Requirements

This chapter describes the requirements (and optional features) for our system and project. We have translated the required features and some of the optional features into use cases. We create tests[1] and workflows for our system from the use cases, so that we can document and make sure that our system fulfills our requirements. The use cases are described on page 8.

3.1 Required system features

The project descriptions lists a number of requirements for the design and implementation of the service and client.

- The service must use a SQL server database.
- The system must run in multi-user environments.
- The service must be implemented in c# using Windows Communication Foundation(WCF).
- The client must enable users to access, administrate, upload and download medias.
- The graphical user interface of the client must be implemented in c# using Windows Forms, ASP or Windows Presentation Foundation(WPF).

Core Features The features listed below are the core features of our system. For us to deliver an acceptable system, we feel these features must be implemented, both on the service and the client, and must be thoroughly tested.

- User
  - Create a new user account.
  - Login.
  - Rent media.
  - Edit profile.
  - Download media.
  - View a list of all movies.
- Content provider
  - Login
  - Upload media.
  - Edit uploaded media.
  - Delete media.

3.2 Optional system features

Section 3.1 described the core features of our system. We consider those features the "bare bones" of our system. In addition to the core features, we have a number of optional features.
Some of these features are involve bigger design decisions than others. This means we may decide not to pick up a "High priority" feature before a "Medium" priority feature, due to time constraints or other other

[1] Further described in our Testing chapter on page 18.

reasons.
The optional features we have decided to implement are in **bold**.

- High priority
  - **Searching for movies**.
  - **View movielists with different sorting**.
  - **Movie release dates**.
  - **Logout**.
  - **View rental history.**.
- Medium priority
  - **Movie editions (SD, HD, Director's Cut, etc.)**.
  - Implement cost for rentals.
  - Let users rate/review media.
  - Store information (Service-end) for analytical and statistical work[2].
- Low priority
  - Stream media from the browser (no download necessary).
  - Social network integration.
  - Allow users to buy products instead of renting them.
  - Instructions and/or tooltips.
  - Age ratings.
  - User banning.
  - Trailers.
  - "Featured" movies.

High Our high priority features are largely quality of life, yet almost core features. Users are used to being able to search for what they want, so we feel this should be one of the first things we do beyond the core.
Additionally we feel that release dates gives the user more information and more ways to sort the movie lists.

Medium The medium priority features are expansions of our core features.
The implementation of rental costs is the most interesting one, yet also what we feel is the most advanced of the options. Because we want our system to handle paid media rental[3], rental costs should be a priority. On the other hand, we feel that if we go for implementing rental costs, we should also design payment options, GUI for the payment and more.
In contrast, the other options are much on the same scale, but do not necessarily have the same amount of extra design time.

Low Low priority features are nice to have but not necessary.
Integrating social networks and streaming media in the browser are cool features, but we risk taking development time away from the core features to add optional features that do not really add much to the system as a whole.
Instructions, manuals, online help and tooltips are nice usability features and they may improve the product as a whole, but if we take development time away from the core features to create these usability features, we may end up with a manual for buggy/non-functional software.

[2]We have implemented a distinction between a users current rentals and rental history.
[3]Discussed in section 2.2.1 on page 4.

3.3 Additional project requirements

In addition to our required and optional features, we have a number of project requirements to make sure we deliver a good product.

- Collaboration
  - Use a version control system.
  - Document design decisions.
  - Use an iterative development strategy.
  - Feature freeze May 8thAfter this date, no new "features" can be added..
  - Code freeze May 18th.
- Quality Assurance[4] (QA)
  - Use cases must be covered by tests[5].
  - Test code coverage must be thorough[6].
    - Minimum overall coverage of the service project: 50%
    - Goal for overall coverage of service: 85%
    - Critical sections of service: 80%
  - All code must be documented[7].
  - User interface must be usability tested[8].

3.4 Use cases for the system

These are the use cases for our system. The use cases only include requirements that we have managed

User management

- A user wants to create a new account.
- A user wants to login.
- A user wants to edit his profile.
- A user wants to logout.

Browsing media

- A user wants to view a list of all offered movies.
- A user wants to browse movies by their release date.
- A user wants to search for a specific movie title.
- A user wants to view all movies of a specific genre.

[4]The requirements listed here are what we feel need to be successfully covered so that we can say we deliver a well tested product.
[5]Functionality must be tested on both the service and the client if possible.
[6]See chapter 8 Testing for reasoning behind numbers.
[7]XML headers for classes, fields, constructors and methods. Additional comments if deemed necessary.
[8]

Media rental

- A user wants to rent a specific movie edition.
- A user wants to view all of his previous rentals.
- A user wants to view his current rentals.
- A user wants to download a current rental,

Content management

- A content provider wants to register and upload a movie.
- A content provider wants to register a movie.
- A content provider wants to upload an edition to an already registered movie.
- A content provider wants to edit information about a movie.
- A content provider wants to delete a movie.

4 Collaboration

4.1 Collaboration

This chapter will focus on how we worked together as a group, how we worked with the SMU team, what problems we ran into and what could have been done differently.

When we began the project, we decided on using the agile form for development known as "SCRUM". SCRUM is a development form where the team works in "sprints", which are a period of time in which the team is supposed to work on certain features. These features are called "user stories", which consist of a name, a short description and an estimation of how much time it takes to complete the story. Each user story descirbe a feature that the team is supposed to develop over the current sprint, and they can be prioritised by the product owner if the product owner wants a certain feature finished before another feature.

SCRUM also contains daily meetings/stand-up meetings, in which everyone from the team stands up and tells what they have been working on since the last meeting, what they intend to work on until next meeting and if anything can prevent them from doing this work. These meetings gives the team a good overview of what's been finished, and what still needs finishing. Furthermore, it gives the team an opportunity to discuss problems that have been encountered, and how to solve these problems.

Another important part of SCRUM is the so-called "retrospective". At the end of every sprint, each team-member writes down some good and bad things that happened during the sprint. These points are all gathered, and discussed by the team, after which the team decides on what points to improve during the next sprint. This allows the team to become improve over time, which, in the end, leads to a better project, both in terms of finished product, but also in terms of how the team works together.

4.2 ITU group structure

Having decided upon using SCRUM for our project, we had to distribute the roles we wanted to use. More precisely, we had the following 4 roles to distribute:

Product Owner The Product Owner is the person/company who has ordered the product. Normally, this would be a specific person or company, but in our case that isn't entirely true. Techinically, our lector is the one who has "ordered" the product, which normally would make him te product owner. However, we also had the Singaporeans to take into account, as they had a say in how the product was supposed to work, as they were supposed to be able to use it as well.

In the end, we decided that the Product Owner should be a combination of our lector along with out Team Leader, as they are the ones who confirm whether the product lives up to the demands.

Team Leader The Team Leader is responsible for making sure that people get their stuff done on time, making sure that people show up to the agreed time, making sure that the team is functioning well, etc. The Team Leader is basically responsible for the whole team. If the Product Owner has a problem with regards to the team, he should contact the Team Leader and let the Team Leader take care of it.

In our team we had two candidates for the Team Leader; Niklas Hansen and Jakob Melnyk. After some talking, Jakob decided to let Niklas get the Team Leader role, as it turned out that it wasn't something he was very interested in anyways.

SCRUM Master The SCRUM Master is responsible for ensuring that the SCRUM process is used as intended. A key part of his role is to keep the team focused on the user stories, and makre sure that the team isn't distracted by outside influences. In our group, we also decided to let the SCRUM Master take control during the daily meetings, the sprint plannings and the retrospectives.

For our SCRUM Master role, we decided upon Frederik Lysgaard, as he was the one who showed the most interest in the role.

QA Responsible The QA Responsible is responsible for taking a look at stories that have been finished, and making sure that they work as intended. This includes running tests that have been written for the story by the teammember who was responsible for the development of that specific story. The QA role was given to Jakob Melnyk because he is very good at making sure that things work as are they are supposed to.

The second thing we had to do with regards to roles, was to decide whether we wanted the roles to rotate on a certain basis, or just let people keep the roles until the end of the project. After some discussion, we decided to keep the roles static. Had we changed them every so often, it would cause confusion for both us and the other group, which is why we decided to keep the roels static.

4.2.1 Meetings

As mentioned in the descruption of SCRUM, meetings are an important part of SCRUM. This meant that we had a lot of focus on meetings, which will be discussed in this section.

In the beginning, we decided to meet on Mondays from 12.00 to 14.00, and Tuesdays and Thursdays from 12.00 to 16.00. Later on, we decided to meet every day except Friday and Sunday, from 10.00-16.00. When we met during these times, we'd start off with a stand-up meeting, where we told each other what we had been working on since the last meeting, and what we intended to work on until the next meeting. After the stand-up meeting we would begin working on our tasks.

Stand-up meetings weren't the only kind of meeting we had, though. Every second Tuesday we would have our retrospective meeting, in which we would discuss how the sprint had gone, and what we could improve on. Using these meetings, we kept improving our work, both with regards to how we worked, but also with regards to the quality of out product.

The last type of meetings we used, were the sprint planning meetings. On these meetings we would take a look at new stories, estimate them and prioritise them, according to which stories our product owner wanted finished first. We would then assign the stories to team members, after which we would working on them. The sprint planning meetings were also used to take a look at our status, see how many stories we had completed in the previous sprint, and how many that were left. The ones left would be prioritised higher in the next sprint, so we could have them finished and begin working on new ones.

4.3 SMU cooperation

We were to work in teams on the ITU side, but that wasn't the only teamwork that was to be done in this project. It was planned that we should collaborate with a group from Singapore Management University during the project. This meant that another "dimension" was added to the project, as we suddenly were to communicate with people whom we had never met, and whose skills we knew nothing about. It turned out to be more of a challenge than expected, as we learned during the course.

We were introduced to the SMU team the 6th of March, where we agreed on using Google+ Hangout (video conference tool) as our method for communicating during meetings. For communication that did not relate to the meetings, we agreed on using email, as it's an efficient tool for communication. It also has the advantage

that everything that is sent back and forth is documented, and thus can be looked at at a later date, if need be.

4.3.1 Meetings

The abovementioned meetings were scheduled to be every Thursday around 13.00-15.00 depending on the day.We had a total of 5 meetings with the SMU team ¡reference to meetings in appendix¿, in which would update each other on how we were doing, after which we would begin talking about what our plans were until the next meeting. The meetings were also used for sharing ideas about what both groups wanted the service to be able to do.

4.3.2 Conflicts

Working with the SMU team was quite a new experience for us, as no one from our team had worked with a team from that far away before. The only expetations we had, were from what we had been told during the lectures. Therefore, we had hoped that working with the SMU team would be relativly painless, and it looked like that was the case at the beginning. But as time went by, we ran into different problems with regards to working with the SMU team, which will be discussed here.

Mood Changes

The use of Google+ Hangout as a video conference tool improved our communication during the meetings. Unfortunately, it did not mean that we were able to predict some of the "mood changes" that the Singaporeans had. During our meetings we'd agreed with them on something, and the next day we'd receive a mail saying "can we do it this way instead? ", with their suggestion usually being something completely opposite of what we had agreed on. Whenever this happened, we'd end up having an email conversation with the Singaporeans, and in the end we would come to a solution that both sides would agree on.

Wrong API

This wasn't the only problem we encountered with regards to communication. They had spent some time looking an API over before they sent us an email with questions about it, and when we received the questions we had no idea what they were talking about. Somehow, they had managed to find an API that wasn't ours, and they had been looking at that one instead of ours. This lead to confusion and a mail conversation, but in the end we managed make them look at the correct API.

Misconception

On our end there was also some misconception with regards to what they were capable of, with regards to programming. We thought that they were about our level when it came to programming, but it turned out that they weren't. This meant that some things were not done correctly on their end, which resulted in extra work for both ends, along with a lot of extra emailing between the two groups.

Behind schedule

This is not to say that they created all the problems, as we had some problems on our side as well, problems which impacted their side. For example we had some database issues at the beginning of the project, which

pretty much prevented us for developing anything for a week. The database issue put us behind schedule, and because of that, the Singaporeans were put behind schedule.

Error reporting

The fact that the Singaporeans were put behind schedule turned out to be a major problem, as they were rather slow to report when they encountered problems. Towards the end of their schedule, they were unable to make our service work for them, and they didn't tell us until they were approaching their own deadline. We managed to solve the issue, though it could have been handled a lot better and faster if we had received their report earlier.

4.3.3 What we could have done differently

All of the abovementioned problems can be said to have happened because of one problem: Bad communication. Not only in the sense that there wasn't much communication between the groups in general, but also in the sense that we talked "past" each other. As stated at the beginning of this chapter, we use SCRUM on the ITU side, and from our second meeting with the SMU team, we thought they were using SCRUM as well. However, during the process, it felt like they were using another form for SCRUM than we were, if even that, as they wanted to implement the full service at once, instead of working on it over time.

This problem, along with a lot of the abovementioned problems, arose because of bad communication, and could easily have been prevented. If we had spent an hour talking with the SMU team, we would have been able to figure out how exactly they were running their project, and we could have told them how we ran our project. This way we'd know what to expect from each other, and we would have an easier time figuring out how to help each other when needed.

Another thing we could have done as well, was simply to communicate more. We didn't have that much communication with the SMU team overall. The largest part of our communication happened during the meetings, and over the email conversations after each meeting, where questions were asked and answered. Between the meetings, the amount of conversation was fairly small, which of course meant that we didn't know how the SMU team was doing.

The last thing we could have done differently, was to be more insistent when it came to getting updates from them. We sent some mails to them once in a while, asking for status on their end, without recieving an answer. This lead us to believe that everything was fine on their end, which it turned out that it wasn't.

In short, more, and better, communication from the beginning would have done wonders for the project as a whole.

5 Design

5.1 Database

5.1.1 Analysis

Decisions

ER-model

SMU involvement

5.1.2 Tables

5.1.3 Entity Framework

5.2 Service

5.2.1 Analysis

5.2.2 Architecture

5.2.3 Interface

SMU involvement

5.3 Client

5.3.1 Analysis

5.3.2 Architecture

5.3.3 Graphical User Interface

Usability

When we sat down to design our GUI, we felt that it was important that we had a user friendly interface, we therefore choosed to make some usability tests (see 8.1.3) since they will allways grant some degree of usability if performed correctly. From these test we got the feedback that we didn't have enough user confirmation, so we therefore incorporated alot of dialogboxses into our design.

(..... billed af dialogbox)

we first made some paper mockups[1], which focused on the aspect of a very clean environment, with the least amount of redundent data, and buttons. When we then asked the test subjects how they felt about the

[1] see appendix for mockups

interface, we then got the feedback that there was to little confirmation in the interface they felt uncertain that there changes was being saved or whatever they had rented a certain movie. When then took that feedback into account and went about making a digital alpha version[2] of interface which had alot more dialogboxses than the paper mockups.

which we used to make the first round of usability tests. From the first test we got the feed back that our system didn't give the user enough feedback in form of dialogboxses and save changes buttons, when then took that into account and started working on a alpha client which we could use for the second test.

mangler at lave Usability test på clienten.

[2]see appendix for screenshots

6 Implementation

6.1 Service

6.1.1 Architecture implementation

Issues, workarounds and fixes

6.1.2 Error handling

6.2 Client

6.2.1 Architecture implementation

Model-View-ViewModel

Windows Presentation Foundation

Issues, workarounds and fixes

6.2.2 Error handling

7 Manual

7.1 Client

7.1.1 Navigating the client

The following section will be a short manual on how to use the client when trying to preform two standart tasks: to rent a movie, and to upload a moive to the service.

Renting a movie

The first thing you see when you start the client is a login page that requires your username and password, if your not allready a user you can press the signup button which will navigate you to registration page where you can signup for the service, when this is done you will be returned to the login page. If you login as a user image **??** will be shown (.....) and a diffrent if you are an content provider but more on that later, when logined as a user you want to rent a movie lets say "The Avengers". You now have two options either to press the most downloaded button to check if it is in the top most downloaded movies or you can search after it by typing in the name in the search field and hitting the search button.
If you use the search option you will then be navigated to a new page **??**(.....) which contains the result of the search which in the case of "The Avengers" would look like this, to rent the movie you then need to select from the list and press "view movie" which in turn then will open this page **??**(.....) where you will be able to see information about the movie, and also be able to choose which edition to rent. You then choose a edition you want to rent in this case we will choose the Blue-Ray edition of "The Avengers" we then select it in the list and click select edition, this will promt us to are page similar to the view movie page, the only exception being that we can now press "Rent edition" which will then add the movie to your account for seven days and then navigate to the download page where you can press the "Download movie" which will let you download the movie and choose a filepath to save it in. **??**(.....) When done you can then press the logout button to logout and return to the login page or you can press the close button (X) which will then logout you out and close down the application completly.

Uploading a movie

In the case where you login as a contentprovider, you get a startscreen where you can see a list of your uploaded movies, you also have the ability to register new movies. To register a new movie you press the " Register movie" button this will then op this image **??**(.....) here you can give the movie a title, release date, genres and a short description you can then register the information by pressing "Register movie", when doing so a dialogbox will apear **??**(.....) prompting you if you want to upload editions right away aswell.
If you press yes the window will then navigate to this page **??**(.....) which allows you to upload an edition of the movie with a name and a file, when you then press the "Upload edition" button, the edition will be uploaded to the service and you will be returned to your start screen.

7.2 Service

8 Testing

8.1 Strategy

8.1.1 Test types

Scenario-level tests

Service-level tests

Graphical user interface tests

Manual GUI test

8.1.2 Regression tests

8.1.3 Usability tests

When designing a user interface you have to take into account that not all users is equally proficient in navigating IT systems, therefore we have to design a interface which is easy to use. To accomplish this we did a couple of usability tests. Usability tests, is a testing technique which focuses on the usability of a user interface, this is measured in non-functional requirements. For usability testing you need a mockup to test against, you then make a list of usability goals[1] if these goals is fulfilled then you have the user interface that you wanted. For the usability test itself you make a list of scenarios that your user shall go through[2], while performing the scenarios, the user is asked to think aloud, such that the overseer of the test can take notes on how to improve the system.

The way we went about doing our usability test, was to first set down as a team and create some paper mockups, which we found user friendly and had high ease-of-use. We then made some usability goals which if fulfilled, would ensure us that our interfaced was indeed user friendly and had a high ease-of-use. With these we made our first usability test on the paper mockups, we then got the feedback that even though our interface was easy to navigate through, we lacked user conformation all our test users felt uncertain that their actions was saved in the database.

[1] see appendix for our usability goals
[2] see appendix for our usability scenarios

8.2 Test results

8.2.1 Code coverage

8.2.2 Usability results

8.2.3 Results of GUI workflows

8.3 Reflection on test strategy

8.3.1 Ideas for improvement

9 Conclusion and reflection

9.1 Issues and potential fixes

9.1.1 Issues we would prioritise

Bibliography

[1] Blog on "The Way of Testivus": http://www.artima.com/weblogs/viewpost.jsp?thread=203994 (Checked on 22nd of April, 2012)

[2] Wikipedia entry on Blockbuster LLC: http://en.wikipedia.org/wiki/Blockbuster_LLC (Checked on 9th of May, 2012)

[3] Blockbuster revenue loss: http://www.thewrap.com/movies/column-post/blockbuster-announces-q2-results-gets-debt-extension-20105 (Checked on 9th of May, 2012)

[4] Wikipedia entry on Netflix: http://en.wikipedia.org/wiki/Netflix (Checked on 9th of May, 2012)

[5] Interview with Gabe Newell (co-founder and managing director of Valve Cooperation): http://www.tcs.cam.ac.uk/story_type/site_trail_story/interview-gabe-newell/ (Checked on 9th of May, 2012)

[6] Roskilde library website: http://www.roskildebib.dk/ (Checked on 9th of May, 2012)

[7] Wikipedia entry on ITunes store: http://en.wikipedia.org/wiki/ITunes_Store (Checked on 9th of May, 2012)

Appendices

A Who did what?

A.1 Code

Frederik Lysgaard	
	GUI design
Jacob Grooss	
	GUI tests
Jakob Melnyk	
	MVVM implementation
Niklas Hansen	
	Database
	Editions
Ulrik Damm	
	Search

A.2 Report

Frederik Lysgaard	
Jacob Grooss	
Jakob Melnyk	
Niklas Hansen	
Ulrik Damm	

B Written Review

B.1 Layout

Even though the first draft is only ten pages, it feels odd not to have a table of contents (ToC). A ToC would have made it easier to navigate the document.
Section/subsection titles were not very distinct from the standard text. Either some form of numbers or letters could be used to show that a new section begins. Italics could also be considered.
The description of the data model and the web-service felt quite clustered and had no real distinction between when one ended and the other began. Again sectioning could alleviate this issue.
Bullet points (or some other "fancy" representation) of the methods in the web-service description would have been good. It was not very clear what was a method and what was not.
The communication section could have been improved by splitting it into subsections. There are three or four subjects discussed in the section and each of them could have had their own subsection.

B.2 Content

The ER-model notation is mentioned (and Søren Lauesens book referred to) in the communication section, but not in the data model section. It probably should be, as the data model section lacked an explanation of how to read the ER-models.
The test report section could have been improved by showing (and explaining) a template, then giving an example of an actual report log entry. Another option could be to just explain the report log snippet you have actually included, instead of just leaving it in there with no explanation.
The text has a lot of poorly argued for decisions. One of particular note is "Revised ER_model. If your smart you will notice that there is no distinction between admins and normal users, but just use your imagination and trust us n this one."
Another example of poor argumentation: "The way we have been testing out system is far from the most optimal way, but it is the only way to do it." A tool such as Pex is mentioned, but it would be nice if there were a more detailed explanation as to why your system is designed in such a way that only manual testing is possible. It seems weird because it is possible to make tests that do exactly what you are doing in your system. They can just do it automatically. Spamming could be avoided by just being sensible about how often/when you test.
Grammar and spelling errors: We assume this is because of no proofreading being done, but this should be a priority come the final hand-in. One example is the label for the revised data model as quoted previously. Just that label has two spelling errors.
The use cases could use some elaboration (in terms of describing each use case), but the illustration works quite well, and you could possibly just explain the more complicated ones (what does it mean to update a movie?).
You have a good problem introduction (besides the spelling and grammar problems).
Description is a bit lacking on the data models, but the way you present them is quite good.

C SMU meeting logs

C.1 Meeting 1

Date: 6th March 2012
Venue: Video conference
Denmark Time: 9:30 – 9:45
Present: Niklas, Frederik, Jacob, Ulrik, Amritpal, Leonard, Satoshi

Topics covered:

- Introduction
- Communication
  - Skype
  - GoogleHangout+
- Sharing of Information
  - Wiki
  - Facebook group
- Next meeting time
  - Thursday, 3pm Denmark Time
- Expectations
  - As good as we can
  - Good communication

Task to be completed before next meeting:

- Read project scope

Immediate Goals

- Formulate requirements

Next meeting
Date: 8th March 2012
Venue: VoIP
Denmark Time: 15:00 – 17:00

Agenda:

- Roles and Responsibilities
- Project requirements
- Confirmation of from of contact

C.2 Meeting 2

Date: 8th March 2012
Venue: VoIP
Denmark Time: 11:00 – 11:45
Present: Niklas, Frederik, Jacob, Ulrik, Jakob, Amritpal, Leonard, Satoshi

Topics covered:

- Requirements gathering
- Use Case development
- Scope of Project
  - Movie: By ITU
  - Music: By SMU

Task to be completed before next meeting:

- Scope of Movie Rental by ITU side
  - Requirements
  - Use Case
- Scope of Movie Rental by ITU side
  - Requirements
  - Use Case

Next meeting
Date: 15th March 2012
Venue: VoIP
Denmark Time: 13:00 - 14:00

Agenda:

- Refining of Requirements and Use Cases
- Web services
- UI design
- Coding

C.3 Meeting 3

Date: 15th March 2012
Venue: VoIP
Denmark Time: 13:00 - 14:00
Present: Niklas, Frederik, Jacob, Ulrik, Jakob, Amritpal, Leonard, Satoshi

Topics covered:

- Sprint 1
  - Started on Tuesday 13 March 2012
  - End 27 March 2012
  - Sprint planning in progress
- Sprint 2
  - Start 28 March 2012
  - End somewhere between 5th-8th March 2012
- Project Specific
  - Database to be used – MS SQL Server
  - ITU will create backend web services and application with basic UI
  - SMU will modify UI and implement some client-side functionalities
  - No shopping cart will be implemented
    - Instead each user will have a profile that has a list of rented items
    - Each user has to create an account and renting of videos will be done in single "transactions" updating user profiles automatically with a list of rented items
    - This will be done client-side by SMU
  - Users: 3 types of users
    - Admin
      - Has admin rights on other 2 users
    - Content Provider
      - Can upload cannot rent
    - Public User
      - Can rent cannot upload
  - Testing
    - Will be done during each sprint together with functionality development
    - SMU will also conduct testing upon receipt of solution from ITU and provide more documentation for report
  - Sharing on facebook functionality – SMU will implement Client Side
  - Facebook login – not a priority
- Project Backlog
  - ITU to provide a project backlog (without ranking) to get both SMU and ITU updated on functionalities to be implemented
- Metric
  - Project Burndown Chart
    - Weekly updates

Task to be completed before next meeting:

- Stories from ITU
- ITU to provide a basic solution with some functionalities and a simple interface for SMU
  - Deployment instructions
  - Acess to database

Next meeting
Date: 22nd March 2012
Venue: VoIP
Denmark Time: 13:00 - 14:00

Agenda:

- Review solutions and stories for remainder of sprint
- De-conflict any issues
- Discuss advanced features
- Review overall schedule

C.4 Meeting 4

Date: 27th March 2012
Venue: VoIP
Denmark Time: 11:15 - 11:20
Present: Niklas, Frederik, Jacob, Ulrik, Jakob, Amritpal, Leonard, Satoshi

Topics covered:

- Sprint 1 Recap
  - Web services are somewhat done
  - SMU team will send list of questions regarding various methods and variables of the web services to ITU for clarification
  - Web service reference address needed to be communicated
- Sprint 2
  - Started
  - Stories of ITU needs to be uploaded to wiki
  - Will end early late next week
- Project Specific
  - Database just re-established on servers
- Project Backlog
  - Needs to be uploaded onto wiki by ITU
- Metric
  - Project Burndown Chart
    - Needs to be uploaded onto wiki by ITU

Task to be completed before next meeting:

- Stories from ITU
- Wed reference address
- ITU to reply to SMU's email

Next meeting
Date: 29th March 2012
Venue: VoIP
Denmark Time: 13:00 - 14:00

Agenda:

- Review solutions and stories for remainder of sprint
- De-conflict any issues
- Review overall schedule

C.5 Meeting 5

Date: 29th March 2012
Venue: VoIP
Denmark Time: 15:45 - 16:45
Present: Niklas, Frederik, Jacob, Ulrik, Jakob, Amritpal, Leonard, Satoshi

Topics covered:

- Web Services
  - Token
    - Checks if these guys is an admin or publisher
  - Register new user
    - Class instances
  - Upload and download of movies
    - Name of file
    - Length of stream
    - 2gb space limit
- Sprint 2
  - Movie ownership
  - Retrieve entire list of movie

Task to be completed before next meeting:

- Web Services API

Next meeting
Date: 5th April 2012
Venue: VoIP
Denmark Time: 13:00 - 14:00

Agenda:

- Review solutions and stories for remainder of sprint
- De-conflict any issues
- Review overall schedule

D System Diagrams

D.1 Class diagrams

D.2 Service API

E Test results

This appendix describes the results of all our tests. Our handwritten, automated tests are arranged in tables in section E.1. Our usability tests are described in section E.3.

E.1 Automated test results

This section shows the result of running our automated tests the day following the code freeze.

E.1.1 Scenario tests

Test title	Test Purpose	Result
AddGenreTest	Verify that you can add a genre to a movie	Pass
AddOne	Verify that if I add a new movie, the first movie to be returned is the new one	Pass
AddOneInFuture	Verify that movies with a release date in the future, will not appear in the newest movies	Pass
AddOneWithoutReleaseDate	Verify that movies without a release date, will not appear in the newest movies	Pass
AdminRentalHistory	Verify that a admin has no rental history	Pass
BrowseKnownGenreTest	verify that when the user browses for a specific known genre, all movies with that genre gets returned	Pass
ContentproviderRentalHistory	Verify that a contentProvider has no rental history	Pass
DeleteMovieEdition	Verify that it is possible to delete a movie edition	Pass
DeleteMovieEditionFromOther Provider	Verify that it is not possible to delete a movie edition, from another content provider	Pass
DeleteMovieFromOtherProvider	Verify that it is not possible to delete a movie that belongs to another content provider	Pass
DeleteMovieTest	Verify that deletion of a movie is possible	Pass
EditMovieFromOtherProvider	Verify that it is not possible to edit a movie uploaded by another content publisher	Pass
EditMovieInformationInvalid MovieIdTypeTest	Verify that the method throws the correct exception when called by an account with an insufficient user type	Pass
EditMovieInformationInvalid UserTypeTest	Verify that the method throws the correct exception when called by an account with an insufficient user type	Pass
EditMovieInformationValidTest	Verify that the method changes the values of the movie	Pass
EditMoviePartOfInfo	Verify that it is possible to only update part of the information about a movie	Pass
EditMoviePartOfInfoInvalid Values	Verify that a field is only updated, if the new value is valid	Pass
EditMoviePartOfInfoMixed ValidInvalid	Verify that fields with new valid values will be updated, even when other fields will not be updated, because of invalid values	Pass
EditPartOfProfileInfoTest	Verify that it is possible to edit only part of a user's profile	Pass
EditPartOfProfileOnlyPassword Test	Verify that it is possible to edit a user's password, and nothing else	Pass

EditProfileTest	Verify that it is possible to edit a user profile	Pass
GetAllGenresTest	verify that all genres in the database gets returned by GetAllGenres	Pass
GetAllMovies	Verify that All work as intended	Pass
GetCurrentRentalsTest	Verify that the user only gets current rentals and that they all belong to him	Fail
GetMovieInformationInvalid MovieIdTest	Verify that the method returns null when called with a movie ID that doesn't corrospond to a movie in the database	Pass
GetMovieInformationValidTest	Verify that the method returns the correct data	Pass
GetUnreleasedMovieInfoFuture Release	Verify that even though editions have been added to a movie, they will not appear / be passed to the clients, if the movie is not released	Pass
GetUnreleasedMovieInfoNo ReleaseDate	Verify that even though editions have been added to a movie, they will not appear / be passed to the clients, if the movie is not released	Pass
InsufficientAccessDeleteMovieTest	Verify that only Content Providers can delete movies	Pass
InvalidValuesRegisterMovieTest	Verify that it is not possible to use invalid values in the method	Pass
Limiting	Verify that it is possible to limit the amount of movies returned	Pass
LoginWithDifferentUsernameAnd Password	Verify that even though a user with a given username exists, and a user with a given password exists, login will fail if those two users are not the same	Pass
LoginWithExistingUser	Verify that login is possible when using the right username and password	Pass
LoginWithNonExistingUser	Verify it is not possible to login, when no users with the given username and password exists	Pass
LoginWithWrongPassword	Verify that even though a user with a given username exists, login will fail if the password is wrong	Pass
LoginWithWrongUsername	Verify that even though a user with a given password exists, login will fail if the username is wrong	Pass
LogoutValidToken	Verify that it is possible to logout, when specifying a valid token	Pass
MostDownloadedMultipleEditions	Verify that even though the rentals are split between multiple editions, the right movies in the right order is still returned	Pass
MostDownloadedWithRentals	Verify that when trying to get the most downloaded movies, the right movies are returned in the right order	Pass
MultipleRentalHistory	Verify that a user wtih serveral movies in rental history and with multiple instance of the same movie will return the correct list	Fail
NotAUserRentMovieTest	Verify that only users can rent movies	Pass
NullTokenRegisterMovieTest	Verify that it is not possible to call the method with a null token	Pass
RegisterMovieTest	Verify that a content provider is able to register movies in the database	Pass
RemoveGenreTest	verify that you can remove a genre from a movie	Pass

RentalHistoryNoRentals	Verify that you will get a empty list from a user with no rental history	Pass
RentalHistoryTest	Verify that it is possible to retreive list of the user rental history	Pass
RentalOfMovieWithFutureRelease	Verify it is not possible to rent a movie with a release date in the future	Pass
RentalOfMovieWithoutRelease Date	Verify it is not possible to rent a movie without a release date	Pass
RentMovieTest	Verify that it is possible to rent a movie	Pass
SearchBadSpelling	verify that search results includes movies with spelling errors in the title	Pass
SearchDifferentCase	verify that a movie is returned when a user searches for the title but with incorrect case	Pass
SearchExcactTitle	Verify that a movie gets returned when the user searches for its exact title	Pass
SearchLimit	verify that putting a limit on the search results actually limits the number of returned movies	Pass
SearchMoreTokens	verify that a movie will be returned, even if tokens not in the name is part of the search string	Pass
SearchOrder	verify that the exact movie title match will be ordered before a partly match	Pass
SearchPartlyTitle	verify that a movie is returned when a user searches for a part of the title	Pass
SearchTokenMatchCountOrder	verify that the search results is ordered by the number of token matches	Pass
SearchVeryBadSpelling	verify that the search result does not include words that are too badly spelled	Pass
SearchWithoutResult	verify that an empty collection is returned when the user searches for a title not in the database	Pass
SignUpWithEmptyEmail	Verify that email has to be set	Pass
SignUpWithEmptyPassword	Verify that password has to be set	Pass
SignUpWithEmptyUsername	Verify that username has to be set	Pass
SignUpWithExistingUsername	Verify that it is not possible to signup with a username that is already in use	Pass
SignUpWithInvalidInfo	Verify that type is automatically set to user, token is reset and ID is auto-generated when trying to set those settings to invalid values	Pass
SignUpWithValidInfo	Verify that it is possible to sign up	Pass
WrongUserTypeRegisterMovie Test	Verify that it is not possible to use the method as a user of type User	Pass

E.1.2 Service tests

Test title	Test Purpose	Result
AllGenresValidServiceTest	Verify that GetGenres return some genres	Pass
AllGenresWihtoutTokenService Test	Verify that without a token, GetGenres doesn't return anything	Fail
DeleteEditionValidServiceTest	Verify that you can delete a movie edition	Fail

DeleteMovieValidServiceTest	Verify that you can delete a movie	Fail
EditMovieInsufficientRightsServiceTest	Verify that normal users cannot edit movies	Pass
EditMovieValidServiceTest	Verify that you can edit movies	Fail
EditUserNullServiceTest	Verify that EditUser fails on invalid input	Pass
EditUserValidServiceTest	Verfiy that you can edit a user	Fail
GetAllMoviesGenreServiceTest	Verify that movies can be limited to a specific genre	Fail
GetAllMoviesLimitServiceTest	Verify that you can limit the number of movies returned	Fail
GetAllMoviesMostDownloadedServiceTest	Verify that movies can be sorted by number of rentals	Fail
GetAllMoviesNewestServiceTest	Verify that movies can be sorted by newest	Fail
GetAllMoviesValidServiceTest	Verify that movies are returned on valid input to GetMovies	Fail
GetAllMoviesWithoutTokenServiceTest	Verify that a token is needed to browse movies	Pass
GetMovieInformationUnknownMovieServiceTest	Verify that GetMovieInformation doesn't return anything when an unknown movie is referred to	Fail
GetMovieInformationValidServiceTest	Verify that GetMovieInformation gets information about a movie	Pass
GetMovieInformationWithoutToken ServiceTest	Vecrify that without a token, GetMovieInformation doesn't return anything	Pass
GetRentalsNullServiceTest	Verify that GetRentals fail without a user toke	Pass
GetRentalsValidServiceTest	Verify that you can get a users rentals	Fail
LoginValidServiceTest	Verify that you can log in	Pass
LoginWrongPasswordServiceTest	Verify that the user won't get logged in, if using a wrong password	Fail
LogoutValidServiceTest	Verify that you can log out	Pass
RegisterMovieInsufficientRightsServiceTest	Verify that normal users cannot register movies	Pass
RegisterMovieValidServiceTest	Verify that you can register a movie	Fail
RentMovieContentProviderServiceTest	Verify that content providers cannot rent movies	Fail
RentMovieValidServiceTest	Verify that users can rent movies	Fail
SearchValidServiceTest	Verifies that movies are returned from a search	Fail
SearchWithoutQueryServiceTest	Verify that nothing is returned for a null query	Pass
SearchWithoutTokenServiceTest	Verifies that nothing is returned without a user token	Pass
SignupMissingInfoServiceTest	Verify that you cannot create a user without basic information	Pass
SignupValidServiceTest	Verify that you can create a new user	Pass

E.1.3 Graphical User Interface tests

Test title	Test Purpose	Result
GuiTest01CreateNewUser	See if it's possible to create a new user account	Pass
GuiTest02UserLogin	See if it's possible for the user to log in	Pass
GuiTest03UserEditUser Information	See if it's possible for the user to edit his/her information	Fail
GuiTest04UserLogout	See if it's possible for the user to log out	Fail
GuiTest05UserViewAllMovies	See if it's possible for the user to view all the offered movies	Pass

GuiTest06UserViewMoviesBy Newest	See if it's possible for the user to sort all movies with the newest movie first	Fail
GuiTest07UserSearchForMovie	See if it'sp ossible for the user to search for a movie	Pass
GuiTest08UserViewMoviesBy Genre	See if it's possible for the user to see all movies from a specific genre	Fail
GuiTest09UserRentSpecific MovieEdition	See if it's possible for the user to rent a specific movie edition	Pass
GuiTest10UserRentAndView Rentals	See if it's possible for the user to rent a movie and find it in his/her rental list	Fail
GuiTest12CPLoginLogout	See if the content provider can log in and log out	Fail
GuiTest13CPRegisterMovie	See if the content provider can register a movie	Pass

E.1.4 Failed automatic tests

This section describes which of our automatic tests that fail and what error they fail with.

Test title	Fail reason
GetCurrentRentalsTest	Threw exception: "ArgumentNullException: Value cannot be null. Parameter name: String"
MultipleRentalHistory	Threw exception: "System.InvalidOperationException: There is already an open DataReader associated with this Command which must be closed first.
DeleteEditionValidServiceTest	Threw exception: "System.ServiceModel.Fault-Exception"
DeleteMovieValidServiceTest	"AssertIsTrue failed. DeleteMovie failed"
EditMovieValidServiceTest	Assert.AreEqual failed. Expected: <New title>. Actual:<Ocean's Eleven>. Movie title wasn't changed
EditUserValidServiceTest	System.ServiceModel.Dispatcher.NetDispatcherFault-Exception: The formatter threw an exception while trying to deserialize the message: There was an error while trying to deserialize parameter
GetMovieInformationUnknownMovie-ServiceTest	Assert.IsFalse failed. GetMovieInformation didn't fail
GetAllMoviesGenreServiceTest	System.ServiceModel.Dispatcher.NetDispatcherFault-Exception: The formatter threw an exception while trying to deserialize the message: There was an error while trying to deserialize parameter
GetAllMoviesLimitServiceTest	Assert.IsTrue failed. Result is false
GetAllMoviesMostDownloadedService-Test	Assert.IsTrue failed. Result is false
GetAllMoviesNewestServiceTest	System.ServiceModel.Dispatcher.NetDispatcherFault-Exception: The formatter threw an exception while trying to deserialize the message: There was an error while trying to deserialize parameter
GetAllMoviesValidServiceTest	System.ServiceModel.Dispatcher.NetDispatcherFault-Exception: The formatter threw an exception while trying to deserialize the message: There was an error while trying to deserialize parameter
GetRentalsValidServiceTest	Assert.IsTrue failed. GetRentals failed
LoginWrongPasswordServiceTest	Assert.IsFalse failed. Login didn't fail.

RegisterMovieValidServiceTest	System.ServiceModel.Dispatcher.NetDispatcherFault-Exception: The formatter threw an exception while trying to deserialize the message: There was an error while trying to deserialize parameter
RentMovieContentProviderServiceTest	System.ServiceModel.Dispatcher.NetDispatcherFault-Exception: The formatter threw an exception while trying to deserialize the message: There was an error while trying to deserialize parameter
RentMovieValidServiceTest	Assert.AreNotEqual failed. Expected any value except:<0>. Actual: <0>. rentals is null
SearchValidServiceTest	Assert.IsTrue failed. Result is false
GuiTest03UserEditUserInformation	Could not find 'Close application' control
GuiTest04UserLogout	Could not find 'Close application' control
GuiTest06UserViewMoviesByNewest	Could not find 'MovieListBox' control
GuiTest08UserViewMoviesByGenre	Could not find 'Close application' control
GuiTest10UserRentAndViewRentals	Could not find 'Close application' control
GuiTest12CPLoginLogout	Could not find 'Close application' control

E.2 Manual test

E.3 Usability tests

This section describes the results of our usability tests. What influence the usability tests had on our graphical user interface and how we performed them is described in the report[1].

[1]The influence is described in section 5.3.3 on page 14 and how we performed the tests is described in 8.1.3 on page 18

F GUI images

F.1 Hand-drawn sketches

F.2 GUI prototype

G Original Use Cases

The use cases here are the first list of use cases we agreed on with SMU. These later changed into what can be found in our Requirements chapter in section 3.4 on page 8.

G.1 User account management

- A user is signing up for the service.
- A user log in to the service.
- A user edits his/her personal information.

G.2 Media browsing

- A user is browsing the newest added movies.
- A user is browsing the most downloaded movies.
- A user is browsing movies by genre.
- A user is browsing all movies and sorts them by their name or genre.
- A user is browsing all movies he/her previously rented.
- A user is searching for at movie by its name.

G.3 Media rental

- A user is renting a movie.
- A user is viewing information about a movie.

G.4 Content management

- A content manager uploads a new movie, and enters information about that movie.
- A content manager edits information for a movie.
- A content manager deletes a movie.

G.5 System management (?)

- A system manager browses all the content managers.
- A system manager creates a new content manager.
- A system manager deletes a content manager.
- A system manager browses all the users.
- A system manager deletes a user.

H F# Handins

This chapter of the appendix contains our handins for the F# assignments. To make sure they could fit on the pages, we had to break up some of the lines in the code. We do feel that we have been able to make it look decent, however. The original .fs files have also been included on the DVD including with the report.

H.1 F# Handins - Frederik Lysgaard

H.1.1 HandIn 1

```
(* Student name: Frederik Roden Lysgaard
   Mail: Frly@itu.dk *)
module Module1

//Opgave 1
let sqr x = x*x

//Opgave 2
let pow x n = System.Math.Pow(x, n)

//Opgave 3
let dup x : string = x+x

//Opgave 4
let rec dupn (s : string)  (x : int) = if x = 1 then s else s + dupn s (x-1)

//Opgave 5
let timediff (hh1, mm1) (hh2, mm2) = (hh2*60 + mm2) - (hh1*60 + mm1)

//Opgave 6
let minutes (hh, mm) = timediff (00, 00) (hh, mm)
```

H.1.2 HandIn 2

```
(* Student name: Frederik Roden Lysgaard
   Mail: Frly@itu.dk *)
module Handin2

//Opgave 7
let downTo (n : int) = if n < 1 then [] else [n .. -1 .. 1]

let downTo2 (n :int) =
    match n with
    | n when n < 1 -> []
    | _ -> [n .. -1 .. 1]

//Opgave 8
```

```
let rec removeEven (list :int list) =
    match list with
    | [] -> []
    | [x] -> [x]
    | a :: b :: rest -> a :: removeEven rest


//Opgave 9
let rec combinePair (list :int list) : (int*int) list =
    match list with
    | [] -> []
    | [x] -> []
    | a :: b :: rest -> (a, b) :: combinePair rest

//Opgave 10
let explode (s :string) : char list = List.ofArray (s.ToCharArray())

let rec explode2 (s :string) : char list =
    match s with
    | s when s.Length < 1 -> []
    | _ -> [s.[0]] @ explode2 (s.Substring 1)

//Opgave 11
let implode (s :char list) : string =
        List.foldBack (fun elem acc -> string (elem) + string(acc)) s ""

let implodeRev (s :char list) : string =
        List.fold (fun elem acc -> string (acc) + string(elem)) "" s

//Opgave 12
let toUpper s =
        implode (List.map System.Char.ToUpper (explode s))

let toUpper1 =
        explode >> List.map System.Char.ToUpper >> implode

let toUpper2 s :string =
        explode s |> (implode << List.map System.Char.ToUpper)

//Opgave 13
let palindrome (s :string) =
        (explode s |> List.map System.Char.ToUpper |> implodeRev) = toUpper s

//Opgave 14
let rec ack (m, n) =
    match (m, n) with
    | (m, n) when m < 0 || n < 0 -> failwith "The Ackermann
                        function is defined for non-negativ numbers only"
    | (m, n) when m = 0 -> n + 1
    | (m, n) when n = 0 -> ack (m - 1, 1)
    | _ -> ack (m - 1, ack (m, n - 1))
```

```
//Opgave 15
let time f =
  let start = System.DateTime.Now in
  let res = f () in
  let finish = System.DateTime.Now in
  (res, finish - start)

let timeArg1 f a = time (fun () -> f a)
```

H.1.3 HandIn 3

H.1.4 HandIn 4 & 5

H.2 F# Handins - Jacob Claudius Grooss

H.2.1 HandIn 1

```
module HandIn1
//Exercise 1
let sqr x = x * x

//Exercise 2
let pow x n = System.Math.Pow(x, n)

//Exercise 3
let dup (s:string) = s + s

//Exercise 4
let rec dupn (s:string, x) = if x = 0 then s else s + dupn(s, x - 1)

//Exercise 5
let timediff (hh1, mm1)(hh2, mm2) = (hh2 * 60 + mm2) - (hh1 * 60 + mm1)

//Exercise 6
let minutes (hh, mm) = timediff (00,00)(hh,mm)
```

H.2.2 HandIn 2

```
//Exercise 7
let rec downTo n = if n < 1 then [] else n :: downTo(n - 1)

let rec downTo2 n =
    match n with
                | n when n < 1 -> []
        | 1 -> [1]
        | _ -> n :: downTo(n-1)
```

```
//Exercise 8
let rec removeEven (xs: int list) =
    match xs with
        | [] -> []
        | [xs] -> [xs]
        | xs :: xy :: rs -> xs :: removeEven(rs)

//Exercise 9
let rec combinePair (xs: int list) =
    match xs with
        | [] -> []
        | [xs] -> []
        | xs :: xy :: rs -> xs :: combinePair(rs)

//Exercise 10
let explode (s:string) =
    s.ToCharArray() |> List.ofArray

let rec explode2 (s:string) =
    match s with
        | s when s.Length < 1 -> []
        | _ -> s.[0] :: explode2 (s.Substring 1)

//Exercise 11
let implode (s:char list) =
    List.foldBack (fun str ch -> string(str) + string(ch)) s ""

let implodeRev (s:char list) =
    List.fold (fun str ch -> string(ch) + string(str)) "" s

//Exercise 12
let toUpper (s:string) =
    implode (List.map (fun x -> System.Char.ToUpper x) (explode s))

let toUpper1 (s:string) =
    explode >> (List.map (System.Char.ToUpper)) >> implode

let toUpper2 (s:string) =
    explode s |> (implode << List.map System.Char.ToUpper)

//Exercise 13
let palindrome (s:string) =
    (explode s |> implodeRev |> toUpper) = toUpper s

//Exercise 14
let rec ack (m, n) =
    match (m, n) with
        | (m, n) when m < 0 || n < 0 -> failwith "The Ackermann function
                                    is defined for non negative numbers only."
        | (m, n) when m = 0 -> n + 1
        | (m, n) when n = 0 -> ack (m - 1, 1)
```

```
        | (m, n) -> ack (m-1, ack(m, n-1))

//Exercise 15
let time f =
    let start = System.DateTime.Now in
    let res = f () in
    let finish = System.DateTime.Now in
    (res, finish - start);

let timeArg1 f a = time(fun () -> f(a))
```

H.2.3 HandIn 3

H.2.4 HandIn 4 & 5

H.3 F# Handins - Jakob Melnyk

H.3.1 HandIn 1

```
module Module1

// Exercise 1
let sqr x = x*x

// Exercise 2
let pow x n = System.Math.Pow(x, n)

// Exercise 3
let dup s : string = s + s

// Exercise 4
let rec dupn (s:string) x =
        if x>=1 then (if x = 1 then s else s + dupn s (x-1)) else ""

// Exercise 5
let timediff (hh1, mm1)(hh2, mm2) = (hh2*60 + mm2)-(hh1*60 + mm1)

// Exercise 6
let minutes (hh, mm) = timediff(00, 00)(hh, mm)
```

H.3.2 HandIn 2

```
module Module2

// Exercise 7
let rec downTo x =
        if x < 1 then [] else (if x = 1 then [x] else x :: downTo (x - 1))
```

```
let rec downTo2 x =
    match x with
    | x when x < 1 -> []
    | 1 -> [1]
    | _ -> x :: downTo2 (x - 1)

// Exercise 8
let rec removeEven (x:int list) =
    match x with
    | [] -> []
    | [xs] -> [xs]
    | xs :: ys :: zs -> xs :: removeEven zs

// Exercise 9
let rec combinePair (x:int list) : (int * int) list =
    match x with
    | [] -> []
    | [xs] -> []
    | xs :: ys :: zs -> (xs, ys) :: combinePair zs

// Exercise 10
let explode (s:string) = List.ofArray (s.ToCharArray())

let rec explode2 (s:string) : char list =
    match s with
    | s when s.Length < 1  -> []
    | _ -> s.[0] :: explode2 (s.Substring 1)

// Exercise 11
let implode (cl:char list) : string =
                List.foldBack (fun elem acc -> string(elem) + string(acc) ) cl ""

let implodeRev (cl:char list) : string =
                List.fold (fun elem acc -> string(acc) + string(elem) ) "" cl

// Exercise 12
let toUpper (s:string) = implode (List.map System.Char.ToUpper (explode s))

let toUpper1 = explode >> List.map System.Char.ToUpper >> implode

let toUpper2 (s:string) = explode s |> (implode << List.map System.Char.ToUpper)

// Exercise 13
let palindrome (s:string) = (explode s |> implodeRev |> toUpper) = toUpper s

// Exercise 14
let rec ack (m, n) =
    match (m, n) with
    | (m, n) when m < 0 || n < 0 -> failwith "The Ackermann function
                                  is defined for non negative numbers only."
    | (m, n) when m = 0 -> n + 1
```

```
    | (m, n) when n = 0 -> ack (m - 1, 1)
    | (m, n) -> ack(m - 1, ack (m, n - 1))

// Exercise 15
let time f =
    let start = System.DateTime.Now in
    let res = f () in
    let finish = System.DateTime.Now in
    (res, finish - start)

let timeArg1 f a = time(fun () -> f(a))
```

H.3.3 HandIn 3

```
odule FSharpHandIn3

type 'a BinTree =
    Leaf
    |   Node of 'a * 'a BinTree * 'a BinTree

let intBinTree =
    Node(
        43,
        Node(25, Node(56,Leaf, Leaf), Leaf),
        Node(562, Leaf, Node(78, Leaf, Leaf))
        )

// Exercise 16
let rec inOrder tree =
    match tree with
    Leaf -> []
    |   Node(n, treeL, treeR) -> inOrder treeL @ [n] @ inOrder treeR

// Exercise 17
let rec mapInOrder (f:'a -> 'b) (tree:'a BinTree) : 'b BinTree =
    match tree with
    Leaf -> Leaf
    |   Node(n, treeL, treeR) ->
            let left = mapInOrder f treeL
            let root = f(n)
            let right = mapInOrder f treeR
            Node(root, left, right)
(*Example:
The result tree should always be the same, as the function should
access all the elements no matter what.
The reason the individual nodes may not contain the same information
could be that the function depends on the order in which
the elements are accessed.*)

// Exercise 18
let rec foldInOrder f a t =
```

```
    match t with
    | Leaf -> a
    | Node(x, leftTree, rightTree) ->
    let left = foldInOrder f a leftTree
    foldInOrder f (f x left) rightTree

// Exercise 19 & 21 & 22
type expr =
    | Const of int
    | If of expr * expr * expr
    | Bind of string * expr * expr
    | Var of string
    | Prim of string * expr * expr

let rec evalN expr (d:System.Collections.Generic.Dictionary<string, expr>) =
    match expr with
    | Const i -> i
    | Prim("-", expr1, expr2) ->
        evalN expr1 d - evalN expr2 d
    | Prim("+", expr1, expr2) ->
        evalN expr1 d + evalN expr2 d
    | Prim("max", expr1, expr2) ->
        List.max [evalN expr1 d; evalN expr2 d]
    | Prim("min", expr1, expr2) ->
        List.min [evalN expr1 d; evalN expr2 d]
    | Prim("=", expr1, expr2) ->
        if evalN expr1 d = evalN expr2 d then 1 else 0
    | If(expr1, expr2, expr3) ->
        if evalN expr1 d <> 0 then evalN expr2 d else evalN expr3 d
    | Bind(var, value, expr1) ->
        d.Add(var, value)
        evalN expr1 d
    | Var(name) when d.ContainsKey(name) ->
        evalN (d.[name]) d
    | Var(name) ->
        failwithf "Unknown variable '%s'" name
    | Prim(opr, _, _) ->
        (printfn "Operation %s not supported" opr; 0)

let eval expr =
    evalN expr (new System.Collections.Generic.Dictionary<string, expr>())

// Exercise 20
let testMinus =
    eval (Prim("-",Const(20),Const(30))) // Expected result = -10
let testPlus =
    eval (Prim("+",Const(20),Const(30))) // Expected result = 50
let testMax =
    eval (Prim("max",Const(20),Const(30))) // Expected result = 30
let testMin =
    eval (Prim("min",Const(20),Const(30))) // Expected result = 20
```

```
let testEqualFalse =
    eval (Prim("=",Const(20),Const(30))) // Expected result = 0
let testEqualTrue =
    eval (Prim("=",Const(20),Const(20))) // Expected result = 1

// Exercise 23
let testBindOne = // Expected result = 57
    eval (Bind("p", Prim("+", Const(13), Const(29)), Prim("+", Var("p"), Const(15))))
let testBindTwo = // Expected result = -16
    eval (Bind("x", Prim("-", Const(13), Const(29)), Prim("+", Var("x"), Const(15))))
let testBindThree = // Expected result = 97
    eval (Bind("x", Const(97), Bind("y", Const(3), Prim("max", Var("x"), Var("y")))))
let testBindFour = // Expected result = 0
    eval (Bind("x", Const(97), Bind("y", Const(3), Prim("=", Var("x"), Var("y")))))
let testBindFive =  // Fail case
    eval (Bind("x", Prim("+", Const(13), Const(29)), Prim("+", Var("y"), Const(15))))
```

H.3.4 HandIn 4 & 5

H.4 F# Handins - Niklas Hansen

H.4.1 HandIn 1

```
// Author: Niklas Hansen <nikl@itu.dk>
module Handin1

// Exercise 1
let sqr (x:int) =
    x * x

// Exercise 2
let pow (x:float) (y:float) =
    x ** y

// Exercise 3
let dup (s:string) =
    s + s

// Exercise 4 - v1
let rec dupn (s:string) (n:int) =
    match n with
    | 0 -> ""
    | _ -> s + dupn s (n-1)

// Exercise 4 - v2
//let rec dupn (s:string) = function
//    | 0 -> ""
//    | n -> s + dupn s (n-1)
```

```
// Exercise 5
let timediff (h1:int, m1:int) (h2:int, m2:int) =
    ((h2 * 60) + m2) - ((h1 * 60) + m1)

// Exercise 6
let minutes (hh:int, mm:int) =
    timediff (00, 00) (hh, mm)


printfn "1. Sqr 3: %i" (sqr 3)
printfn "2. pow 3 2: %f" (pow 3.0 2.0)
printfn "3: dup \"Hi \": %s" (dup "Hi ")
printfn "4. dupn \"Hi \" 3: %s" (dupn "Hi " 3)

printfn "5a. timediff (12, 34) (11, 35): %i" (timediff (12, 34) (11, 35))
printfn "5b. timediff (12, 34) (13, 35): %i" (timediff (12, 34) (13, 35))

printfn "6a. minutes (14, 24): %i" (minutes (14, 24))
printfn "6b. minutes (23, 1):  %i" (minutes (23, 1))
```

H.4.2 HandIn 2

```
// Author: Niklas Hansen <nikl@itu.dk>
module Handin2

// Exercise 7a
let rec downTo (n:int) =
    if n > 0
        then n :: downTo (n-1)
        else []

// Exercise 7b
let rec downTo2 (n:int) =
    match n with
    | n when n <= 0 -> []
    | _ -> n :: downTo2 (n-1)

// Exercise 7b v2
//let rec downTo2 = function
//    | n when n <= 0 -> []
//    | n -> n :: downTo2 (n-1)

// Exercise 8
let rec removeEven = function
    | [] -> []
    | [n] -> [n]
    | n :: m :: tl -> n :: removeEven tl

// Exercise 9
let rec combinePair = function
    | []-> []
```

```
    | [n] -> []
    | n :: m :: tl -> (n, m) :: combinePair tl

// Exercise 10a
let explode (s:string) =
    let chars = s.ToCharArray()
    List.ofArray(chars)

// Exercise 10b
let rec explode2 (s:string) =
    match s with
    | "" -> []
    | _ -> s.Chars 0 :: explode2 (s.Remove(0, 1))

// Exercise 11a
let implode (c:char list) =
    List.foldBack (fun x y -> sprintf "%c%s" x y) c ""

// Exercise 11b
let implodeRev (c:char list) =
    List.fold (fun x y -> sprintf "%c%s" y x) "" c

// Exercise 12a
let toUpper (s:string) =
    implode (List.map (fun x -> System.Char.ToUpper(x)) (explode s))

// Exercise 12b
let toUpper1 (s:string) =
    (explode >> List.map (fun x -> System.Char.ToUpper(x)) >> implode) s

// Exercise 12c
let toUpper2 (s:string) =
    s |> (implode << List.map (fun x -> System.Char.ToUpper(x)) << explode)

// Exercise 13
let palindrome (s:string) =
    let org = s.ToLower().Replace(" ", "")
    let rev = new string (Array.rev (org.ToCharArray()))
    org = rev

// Exercise 14
let rec ack (m:int, n:int) =
    match m, n with
    | (0, n) -> n + 1
    | (m, 0) when m > 0 -> ack(m - 1, 1)
    | (m, n) when m > 0 && n > 0 -> ack(m - 1, ack(m, n - 1))
    | (m, n) -> failwith "Invalid input!"

// Addon for Exercise 15
let time f =
    let start = System.DateTime.Now
```

```
    let res = f ()
    let finish = System.DateTime.Now
    (res, finish - start)

// Exercise 15
let timeArg1 f a =
    time (fun () -> f a)


printfn "7a. downTo 5: %s" ((downTo 5).ToString())
printfn "7a. downTo -3: %s" ((downTo -3).ToString())
printfn "7b. downTo2 5: %s" ((downTo2 5).ToString())

printfn "8. removeEven [1; 2; 3; 4; 5]: %s" ((removeEven [1; 2; 3; 4; 5]).ToString())
printfn "8. removeEven []: %s" ((removeEven []).ToString())
printfn "8. removeEven [1]: %s" ((removeEven [1]).ToString())

printfn "9. combinePair [1; 2; 3; 4]: %s" ((combinePair [1; 2; 3; 4]).ToString())
printfn "9. combinePair [1; 2; 3]: %s" ((combinePair [1; 2; 3]).ToString())
printfn "9. combinePair [1; 2]: %s" ((combinePair [1; 2]).ToString())
printfn "9. combinePair []: %s" ((combinePair []).ToString())
printfn "9. combinePair [1]: %s" ((combinePair [1]).ToString())

printfn "10a. explode \"star\": %s" ((explode "star").ToString())
printfn "10b. explode2 \"star\": %s" ((explode2 "star").ToString())

printfn "11a. implode ['a';'b';'c']: %s" (implode ['a';'b';'c'])
printfn "11b. implodeRev ['a';'b';'c']: %s" (implodeRev ['a';'b';'c'])

printfn "12a. toUpper \"Hej\": %s " (toUpper "Hej")
printfn "12b. toUpper1 \"Hej\": %s " (toUpper1 "Hej")
printfn "12c. toUpper2 \"Hej\": %s " (toUpper2 "Hej")

printfn "13. palindrome \"Anna\": %s" ((palindrome "Anna").ToString())
printfn "13. palindrome \"Ann\": %s" ((palindrome "Ann").ToString())

printfn "14. ack(3, 11): %i" (ack(3, 11))

printfn "Extra. time: %s" ((time (fun () -> ack (3,11))).ToString())
printfn "15. timeArg1 ack (3, 11): %s" ((timeArg1 ack (3, 11)).ToString())

System.Console.ReadKey(true)
```

H.4.3 HandIn 3

H.4.4 HandIn 4 & 5

H.5 F# Handins - Ulrik Flænø Damm

H.5.1 HandIn 1

```
KF 02
Handin 1
Ulrik Damm (ulfd@itu.dk)

let sqr x = x * x;;

let pow x n = System.Math.Pow (x, n);;

let dup s = s + s;;

let rec dupn s n =
        match n with
        | 0 -> ""
        | 1 -> s
        | n -> s + dupn s (n-1)
        ;;

let timediff (time11, time12) (time21, time22) = (time21 - time11) * 60 +
(time22 - time12);

let minutes (time1, time2) = timediff (0, 0) (time1, time2);

let Main =
        printfn "%i" (sqr 2);
        printfn "%f" (pow 2.0 3.0);
        printfn "%s" (dup "Hi ");
        printfn "%s" (dupn "Hi " 3);
        printfn "%i" (timediff (12,34) (11,35));
        printfn "%i" (timediff (12,34) (13,35));
        printfn "%i" (minutes (14,24));
        printfn "%i" (minutes (23,1));
```

H.5.2 HandIn 2

```
let rec downTo n =
    if n < 2
    then raise (new System.Exception("Invalid value"))
    else if n = 1
    then [1]
    else n :: downTo (n-1);;

let rec downTo2 = function
    | n when n < 1 -> raise (new System.Exception("Invalid value"))
    | 1 -> [1]
    | n -> n :: downTo2 (n-1);;

let rec removeEven = function
```

```
    | [] -> []
    | [n] -> [n]
    | [n; _] -> [n]
    | n :: m :: tl -> removeEven [n; m] @ removeEven tl;;

let rec combinePair = function
    | [] -> []
    | [n] -> []
    | [n; m] -> [n, m]
    | n :: m :: tl -> combinePair [n; m] @ combinePair tl;;

let explode (str : string) = List.ofArray(str.ToCharArray());;

let rec explode2 (str : string) =
        if str.Length = 0 then []
        else if str.Length = 1 then [str.Chars(0)]
        else str.Chars(0) :: explode2 (str.Remove(0, 1));;

let implode str = List.foldBack (fun x y -> sprintf "%c%s" x y) str "";;

let implodeRev str = List.fold (fun x y -> sprintf "%c%s" y x) "" str;;

let toUpper str =
     implode (List.map (fun x -> System.Char.ToUpper x) (explode str));;

let toUpper1 str =
     (explode >> (List.map (fun x -> System.Char.ToUpper x)) >> implode) str;;

let toUpper2 str =
     (implode << ((fun x -> System.Char.ToUpper x) |> List.map) << explode) str;;

let rec palindrome (str : string) =
    if str.Length <= 1 then true
    else if str.Chars(0) = str.Chars(str.Length - 1)
         then palindrome (str.Substring(1, str.Length - 2))
    else false;;

let rec ack = function
    | (0, n) -> n + 1
    | (m, 0) -> ack (m - 1, 1)
    | (m, n) -> ack (m - 1, ack (m, n - 1));;

let time f =
    let start = System.DateTime.Now in
    let res = f () in
    let finish = System.DateTime.Now in
    (res, finish - start);

let timeArg1 f a = time (fun x -> f a);;

let Main = printfn "lol";;
```

H.5.3 HandIn 3

H.5.4 HandIn 4 & 5