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 15), 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 10).

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 13.

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**.
- 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.

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.

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

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

  - Use a version control system.
  - Document design decisions.
  - Use an iterative development strategy.
- Quality Assurance (QA)
  - Use cases must be covered by tests[4].
  - Test code coverage must be thorough[5].
    - Minimum overall coverage of the service: 50%
    - Goal for overall coverage of service: 85%
    - Critical sections of service: 80%
  - All code must be documented[6].
  - User interface must be usability tested.

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.

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]Functionality must be tested on both the service and the client if possible.
[5]See chapter 8 Testing for reasoning behind numbers.
[6]XML headers for classes, fields, constructors and methods. Additional comments if deemed necessary.

4 Cooperation

4.1 Cooperation

4.2 ITU group structure

4.2.1 Meetings

4.3 SMU cooperation

4.3.1 Meetings

4.3.2 Conflicts

4.3.3 What we could have done differently

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

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

Issues, workarounds and fixes

6.2.2 Error handling

7 Manual

7.1 Client

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

7.1.1 Navigating the client

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 diffrent if you are a 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 search field and hitting the search button.
By doing so you will 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 page the only exception that we can now press "Rent edition which will then add the movie to your account for seven days and navigation to the download page where you can then press the "Download movie" which will let download the movie and choose a filepath to save it in. **??**(.....) When done you can the 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.

In the case where you login as a contentprovider, you get a startscreen where you can see a list of your uploaded movies and 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 a 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 interface tests

8.1.2 Regression tests

8.1.3 Usability tests

8.2 Test results

8.2.1 Code coverage

8.3 Reflection on test strategy

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 Service API handed to SMU

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.2. Our usability tests are described in section **??**.

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

E.1.2 Service tests

Test title	Test Purpose	Result

E.1.3 Graphical User Interface tests

Test title	Test Purpose	Result

E.2 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 10 and how we performed the tests is described in 8.1.3 on page 13

F GUI images

F.1 Hand-drawn sketches

F.2 GUI prototype

G Original Use Cases

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

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