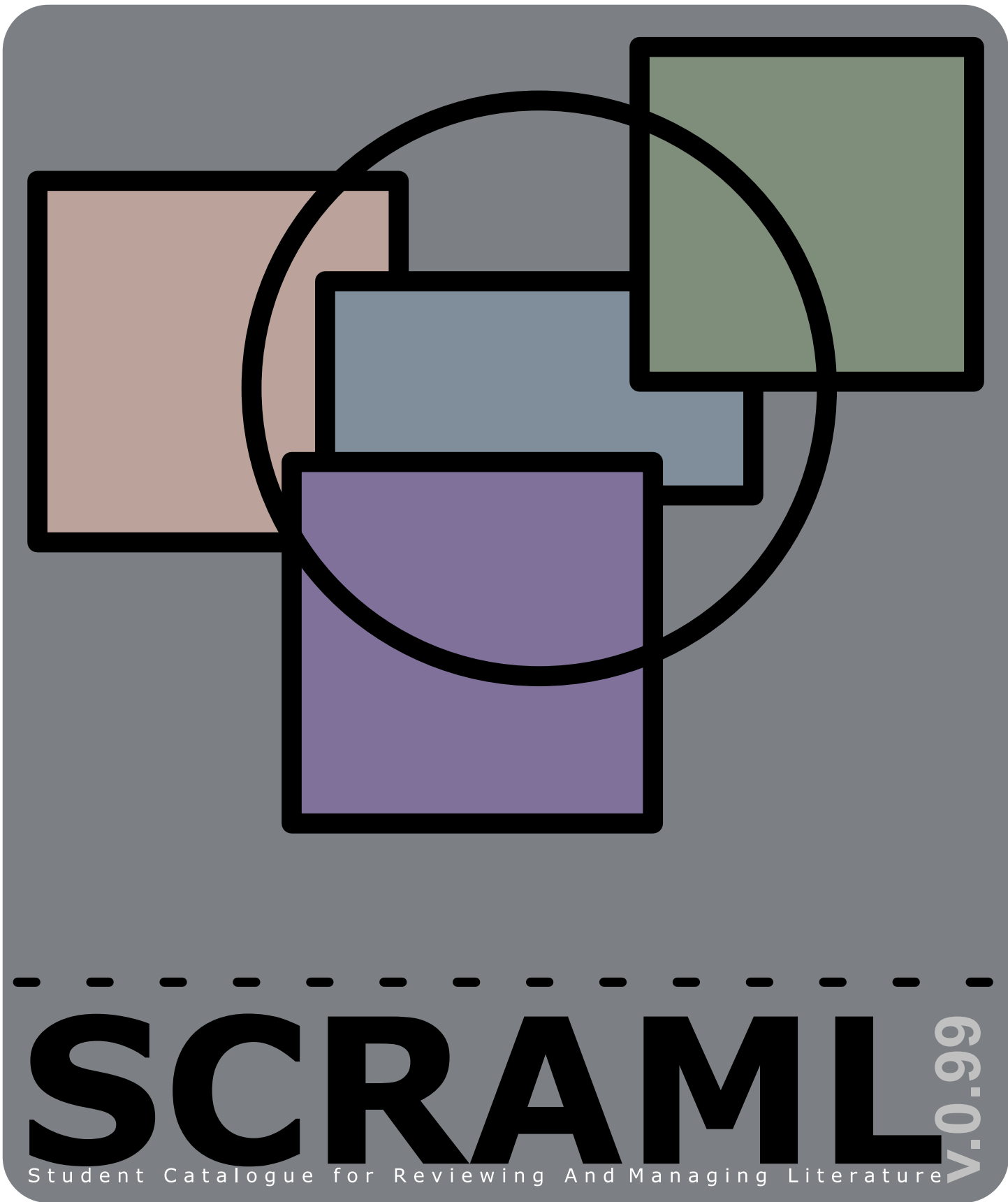


Group d103a - Inf1/Dat1 - 2006



SCRAML v.0.99
Student Catalogue for Reviewing And Managing Literature

Jonatan Riber Andersen
Kasper Stæhr Hornbek
Dan Leinir Jensen
Jacob Korsgaard
Andreas Ibsen
Niels Husted
Jan Jensen

Project report
Final version

Title:

SCRAML
Student Catalogue for Reviewing And Managing Literature

Theme:

Article Catalogue System

Project period:

Inf1/Dat1, fall semester 2006
September 04th to December 21st

Project group:

d103a

Participants:

Jonatan Riber Andersen
Kasper Stæhr Hornbek
Dan Leinir Jensen
Jacob Korsgaard
Andreas Ibsen
Niels Husted
Jan Jensen

Supervisor:

Dalia Tiesyte

Print run:

9

Pages:

123

Appendices (with type):

Usability test tools (7, text)
Analysis and design standards (3, text)
Analysis of del.icio.us (1, text)
Technical Memos (6, text)
iTunes Screenshot (1, image)
Product (1, CD-ROM)

Abstract:

The report describes the process of analysis, design, implementation, testing and documentation of the SCRAML (Student Catalogue for Reviewing And Managing Literature) system. SCRAML is a tool designed for managing and discussing literature used in university projects and the relevance of this literature to the project.

The system has been developed using Object-Oriented Analysis & Design and implemented in C# using Object-Oriented Programming.

The outcome was a fully functional client/server-based system with multi-user support. The system has been exposed to a usability test. Furthermore parts of the system were tested using black and whitebox testing. The tests showed that the system has a high level of usability and a low rate of errors.

The purpose of this project is to acquire experience in using Object Oriented Analysis and Design (OOA&D), and Object Oriented Programming (OOP) by developing an application using these disciplines. The OOA&D model (as described in the book Object Oriented Analysis & Design [6]) is followed strictly, but a waterfall development model was used as opposed to the suggested iterative approach. This was done at the request of the lecturers of this semester. For programming the language C# is used in conjunction with the Microsoft .NET framework. Figures are designed according to Unified Modeling Language (UML).

I Analysis	7
1. The Task	8
1.1. Purpose	8
1.2. System Definition	8
1.2.1. FACTOR	8
1.2.2. Stakeholder analysis	9
1.3. Context	10
1.3.1. The current situation	10
1.3.2. Rich pictures	11
1.4. Problem Domain	12
1.5. Application domain	14
2. Problem domain	15
2.1. Classes	15
2.1.1. Description of classes	15
2.1.2. Events	19
3. Application domain	21
3.1. Usage	21
3.1.1. Overview	21
3.1.2. Actors	21
3.1.3. Scenarios	22
3.1.4. Use cases	23
3.2. Functions	29
3.2.1. Complete list of functions	29
3.2.2. Specification of complex functions	29
3.3. User interface	31
3.3.1. Usage aims	31
3.3.2. Conceptual model and forms of interaction	32
3.3.3. General interaction model	33
3.3.4. Technical platform	33
4. Strategy for Future Development	36
4.1. Economy	36
II Design	37
5. The Task	38
5.1. Purpose	38
5.2. Corrections to the Analysis	38
5.3. Quality Goals	38

6. Technical Platform	40
6.1. Equipment	40
6.2. System Software	40
6.3. System Interfaces	40
6.4. Design Language	41
7. Architecture	42
7.1. Design criteria and requirements crucial for the architecture	42
7.2. Generic design decisions	43
7.3. Component Architecture	45
7.4. Exemplary Design	47
7.4.1. Register user sequence diagram	47
7.4.2. Login sequence diagram	47
7.4.3. Create project sequence diagram	48
8. Components	49
8.1. General Overview	49
8.2. Descriptions of Components	49
8.2.1. PersistentData Component	49
8.2.2. Catalogue Component	50
8.2.3. AccessHandler Component	51
8.2.4. Server-API Component	51
8.2.5. Client-API Component	52
8.2.6. The Graphical User Interface Component	52
8.3. Interaction spaces	52
8.3.1. Create user dialogue	52
8.3.2. The Literature entry window	53
8.3.3. Presentation model	53
9. Programming	57
9.1. Programming	57
9.1.1. Program execution	57
9.1.2. GUI: CreateNewUserForm	58
9.1.3. The AccessHandler	58
9.1.4. Catalogue	59
9.1.5. PersistentData: XMLHandler	60
III Implementation	62
10. Implementation	63
10.1 Environment	63
10.2 Implementation vs. Design	63
10.3 The GUI	67
10.3.1 The Main Window:	68
10.3.2 Project View:	68
10.3.3 Literature View:	68
10.3.4 Custom Controls	69
IV Test	73

11. Unit tests	74
11.1 Blackbox testing	74
11.2 Whitebox testing	76
12. Usability test	80
12.1 Plan for the usability test	80
12.2 Analysis of results	83
12.2.1 Discussion	88
V Study Report	92
13. Academic reflection	93
13.1 Project and team management	93
13.2 System and the domain	95
13.3 Analysis	95
13.4 Design	98
13.5 Implementation	99
13.5.1 Data storage	101
13.6 Unit testing	102
13.7 Usability testing	102
13.8 Conclusion	104
VI Appendix	106
A. Usability test	107
A.1. Test documents	107
A.2. Test introduction	107
A.3. Tasks for the usability test	108
A.4. Exercises for the usability test	111
A.5. Expected solutions to the exercises	112
A.6. Debriefing questions for the usability test	113
B. Analysis and design standards	114
C. Technical Memos	118
C.1. Passing data from server to client	118
C.2. Using a DataTable structure for storing Tags	118
C.3. Saving DateTime using XML	118
C.4. Error Handling in Persistent Data	119
C.5. UniquelyIdentifiable	119
C.6. XML	119
D. Role model analysis of del.icio.us	120
D.1. Folksonomy	120
D.2. Item relevance	121
D.3. Relevance	121
D.4. iTunes	121

Part I

Analysis

The analysis document describes the process of defining a system to solve the problem at hand. First off a preliminary system definition is described, then the problem domain is clarified and key objects involved, and their behavior. Finally the application domain is described, usage and users of the system defined, as well as key functions of the program.

The Task

1.1. Purpose

This document describes the development of a literature management system also referred to as article cataloging system. The main purpose of this system, is to ease the administration of literature sources in university projects. The motivation and starting point for this project is the following project-proposal:

The goal of this project is to develop a catalogue system for text material used in an academic project. During an academic project such as a semester project or a PhD project a vast amount of material is read. Some of this material is cited in the project, some is forgotten, and some is never used. It is often the case that not all members of the project read all the material. To allow the participants of the project to easily share information about the read material a catalogue system is proposed. The catalogue system should allow the participants of a given project to enter information about the read material such as summaries, notes, and applicability in the current project. The system should be able to store information about different kinds of content such as books, articles, technical reports etc. The material might have cross-references and it should be possible to specify this. The content differs in some areas but has a wide range of similar characteristics. The system should be able to store and retrieve the catalogue from disk. It should be possible for different users to register material and comments in the system, but it can be assumed that only a single user is using the system at once (on a single machine).

1.2. System Definition

The system definition is the first platform for an agreement between the customer and the developer. In this project we take on the role as developers, while the customer is imaginary. The system definition is a short text that in general terms states fundamental decisions about the system.

1.2.1. FACTOR

FACTOR [6] is an acronym for criteria that a system definition must conform to to be a robust reference for the development process. The criteria for the article cataloging system are:

Functionality The main functionality of the system is administration of semester project groups' literature resources (books, journals, reports, articles, web content etc.), i.e. cataloguing information about literature entries and attaching summaries, reviews and comments.

Application domain The system is to be used internally in a student project group and across student project groups and their supervisors at a university.

Conditions In order for the system to have any right to exist it is important that it has high utility and that the system is easy and satisfying to use for the people in the application domain.

Technology The system is implemented in the programming language C#, and is supposed to run on students' computers. This is why the technology demand is a standard PC supporting the .NET framework¹.

Objects The objects that the system has to handle are: Projects, users, literature entries, literature suggestions, reviews and comments.

Responsibility Our assumption is that a group during a semester project period studies large amounts of literature, and that it can be difficult to achieve an overview and basic understanding of it. This is why the system is meant to be a tool for sharing of knowledge of literature resources in a semester project group (including its supervisors) and across other project groups.

The system definition

An IT-system intended to be used as a tool for sharing of knowledge of literature resources in and across student project groups at a university. The purpose of the system is to catalogue information about literature entries such as titles, authors and year of publication of books, articles, journals, reports, other papers and web content. The system is intended to be used in a semester project by both the project group members and the group's supervisor(s) and it should be possible to retrieve literature information from finished projects and other on-going projects. The assumption is that a group during a semester project period studies large amounts of literature, and that it can be difficult to achieve an overview and basic understanding of it. Not all encountered literature is applicable in the current project, however it might be in a later project and it can be hard to remember the content and scope of a specific book. Furthermore not every group member is familiar with every piece of literature. On this background the system should assist group members in affiliating a project with catalogued information about their literature including attached summaries, comments and reviews. Afterwards it should be possible for a group member to search for and retrieve this information. In terms of system properties it should be easy and satisfying to use and it should be able to run on a standard PC supporting the .NET framework.

1.2.2. Stakeholder analysis

The Oxford Online Dictionary [1] defines a stakeholder as

a person with an interest or concern in something.

It is common practice to divide stakeholders into several categories, depending on how directly they influence the project/entity. This is why we have primary, secondary, tertiary, and facilitating stakeholders. Our focus in the development process is centered on the former two most influential categories.

¹According to [7].

Primary stakeholders:

University students in a project group: The students are the main participants in the application domain, and thus the primary intended users of our system. It is our goal to develop a system that aids managing their project literature.

Project group supervisors: The supervisors might make literature suggestions to the group, or be interested in whether the applied literature is sufficient and academically approved.

Secondary stakeholders:

Researchers: If a researcher is working on a similar topic as the project group, he/she might be interested in exchanging literature references. Furthermore even though our system is aimed at helping the students, it might be useful amongst professional researchers as well. However one must assume that such researchers do not have the exact same demands as students. This is why we delimit our application domain to student project groups.

Tertiary stakeholders:

Librarians: In some cases librarians help students obtain literature for a project. If our system provides a list of previously encountered material, it can ease the librarians' search process by letting them ignore this material, or inspire them to find related material.

The faculty: If we assume that an article cataloguing system helps improving the quality of studies, the faculty might have an interest in applying the system in all groups.

Facilitating stakeholders:

IT-staff: The IT-staff at the university could take the role as the system-administrators of the system if implemented at the entire faculty. They would probably demand that the system has low system and maintenance requirements.

Developers: Obviously the developers have an interest in the succes of the system as well as the users, but because developers are not part of the application domain, they are only categorized as facilitating stakeholders.

1.3. Context

In the following section we address the relevant circumstances in the context of our system. The main goal of this section is to explicitate the situation, and create an overview of the work tasks in our article cataloguing system.

1.3.1. The current situation

In this first section we provide by example a description of how the actual student project work might take place at our own Aalborg University. The purpose is to gain an understanding of the flow of literature in a real world project group. Obviously we cannot take all aspects into account because each project group will do things differently. In

other words the following are activities that all groups go through (though perhaps in a different order than the one given below).

At the beginning of a semester the project groups are formed. Afterwards a group room and one or more supervisors are allocated to each group. Large parts of the project work is carried out in the group room. Each group decides on a project topic based on a catalogue of project proposals written by the supervisors. After project plan and report content outline are devised, the next step is gathering of information. Resources are retrieved by the group members by visiting libraries and searching the Internet and different databases. Furthermore a group receives literature suggestions by attending the various courses offered.

When the group members have obtained some literature they either utilize them right away or store them for future use. In case a piece of literature is in the shape of web content or in another electronic format it is either stored on a hard drive as a link or as an electronic copy. In case it is a book, journal, a printout (of for instance electronically available material) or another physical piece of literature, the group members have to store it in a physical space – typically the group room or at group members' private address.

The process of searching for new or previously encountered literature is often ongoing during most of a project period, for which reason the need for managing the literature is present at all times.

After the gathering of information it is time for a project group to write the project report. This is a prolonged task that to a considerable extent is based on the obtained literature. For that reason it is important that the group members have swift access to both information about and to the literature itself. However many issues can obstruct this accessibility; for instance when several members searches for literature it may be hard for the individual members and the group as a whole to keep track of who have come across which material, and whether specific material is applicable in the project.

Furthermore a group member, who holds a specific book that is needed in a certain part of the report, may be absent because of illness, and thus the remaining group members have a problem. Both with regards to the book itself and information about it – such as title and author names.

Another possible issue can arise when two group members have studied the same piece of literature and disagree on how relevant it is. They advance their arguments but settle on extending the decision whether to apply it or not. When they at a later time return to the same material they might have a hard time remembering the course of their discussion, and thus have to start from scratch once again.

It is most likely that a project group's supervisor(s) have some knowledge about the project topic, and consequently is able to recommend some specific material. However some difficulties concerning distributing this information to an entire group may occur.

As the time comes when the project groups have to hand in their reports they need to incorporate a bibliography. If this has not been devised continuously during the project period, it may be difficult to collect a list of applied literature at this late stage.

1.3.2. Rich pictures

We need to analyze the problem domain, and to highlight the problematic parts of the project, so it will be possible to eliminate these problems at an early stage in the process.

Our tool to do this is rich pictures. A rich picture is a drawing which expresses an individual's personal understanding of a given situation.

A lot of information can be included into a rich picture, so in order to ensure utility it is important that the main focus remains on the important aspects of the situation. In the

development of the rich pictures our group made various pictures and then combined the drawings into a common set of pictures.

Another reason to develop rich pictures is that they provoke the system developers to develop new ideas. Furthermore the pictures can prove helpful in a situation where the developers have to communicate with the future users.

It is practical to differentiate between rich pictures that focus on stability (The current situation), and pictures that focus on change (The future situation) [6].

Focus

Figure 1.1 on the facing page shows a representation of the actual situation – with focus on how students are categorizing their literature today:

Text books: Literature that the project group studies.

(1) Happy Student: The guy who has read material that he finds relevant to the project.

(2) Neutral Student: The guy who does not know, if the material he has read is relevant to/usable in the project.

(3) Unhappy Student: The guy who has read a lot of irrelevant material.

(4) Fainted/Passed out Student: He was drunk so he forgot which parts of the literature that was relevant and interesting. He could also represent a black hole, where some good sources go without any reason, and maybe he will read material that is already read by other members of the group.

Change

In figure 1.2 on the next page we have illustrated how we expect our cataloguing system to aid the project groups in organizing their literature.

Text books: In the beginning of the project a lot of material is read by the members of a project group.

The light bulbs: When the material is read, the group members decide which parts are relevant in the current project.

The single computer: The material is categorized in our cataloguing system, where a group member can input sources, comments, keywords, and a rating based on how applicable the material is in the project.

The row of four computers: In the last step, it is possible for other group members to access the cataloguing system on their own computer.

1.4. Problem Domain

The final system should register comments, ratings of relevancy, cross-references, summaries, references to related projects, used and non-used sources, the topic of the current project, and participants in the project. The participants can be divided into two categories: students and supervisors.

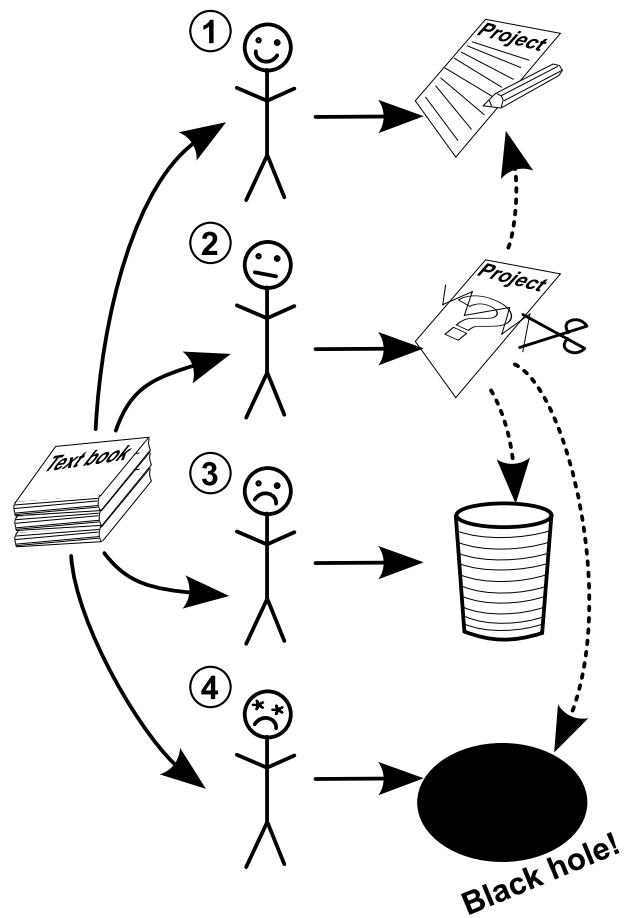


Figure 1.1.: This rich picture illustrates how students are categorizing their literature today.

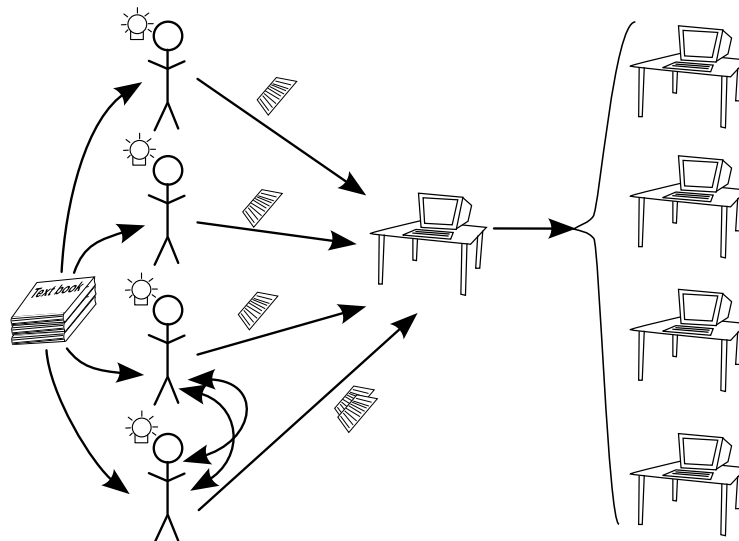


Figure 1.2.: This rich picture illustrates how students take advantage by using our cataloging system.

The students are the ones who normally submit data about sources to the system.

The supervisors benefit from knowing which sources the group is using, the ability to suggest sources and to halt the students if they are getting sidetracked with their literature.

The focus of the system is to ease the work in the project group, and to ensure that important sources are not lost.

1.5. Application domain

The system must support insertion of various information about sources for university projects. The students in a project group input pieces of literature relevant to their current project into the system. This can be articles, books, other projects, websites, lectures or other types of material. When a student submits material to the system, the material must be viewable not only to other members of his group and their supervisor, but also to other users of the system.

Essential tasks of the system

- Register participants
- Manage participants
- Register projects
- Register literature entries
- Manage literature entries
- Distribute information to group-members and supervisors

Actors in the application domain

- The members of any project group
- The supervisors

The items above should be seen as a temporary list of the tasks and actors, and will be further specified in the following chapters.

Problem domain

2.1. Classes

This section describes the classes in the problem domain. These classes are selected using an iterative approach. The classes are part of the model that contains the information the system has to manage. Classes are related in the model as pictured on figure 2.1. We do not use any clusters.

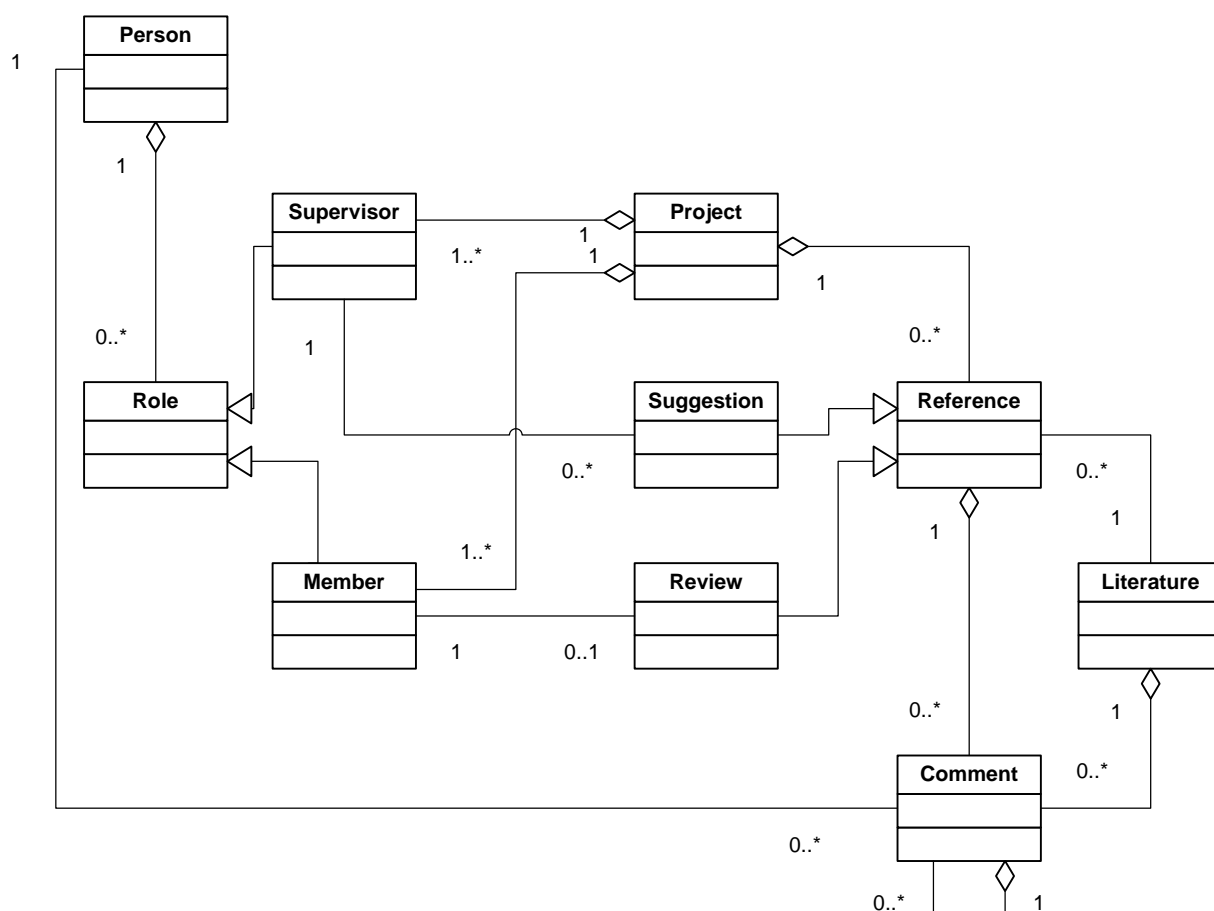


Figure 2.1.: Class diagram of the problem domain

2.1.1. Description of classes

In the description of each class, all references to the class and other classes are marked using *italics>. Each class is described through a general definition, a list of attributes and a description of the class behavior. The model contains some specializations, only a single one of these has additional relevant information attached, beyond the relations to other classes in the model. The additional attributes of *review* is described in the description of the *reference* class.*

Person class

See figure 2.2

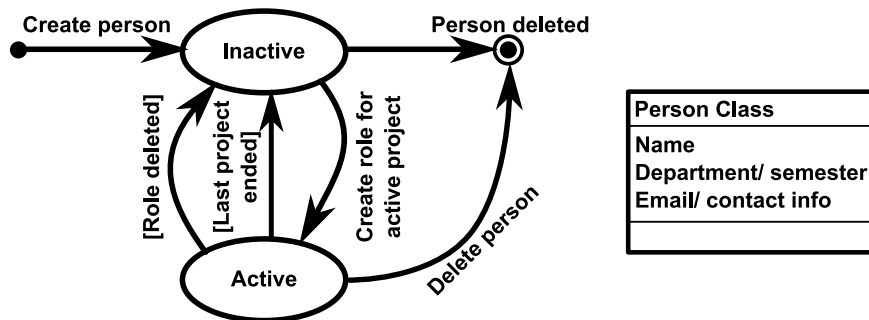


Figure 2.2.: Statechart diagram and attribute table for the *person* class

Definition:

The *person* class describes a *person*; a *person* has *roles* which connect him to *projects*, as either a *supervisor*, or a *member* of a *project* group.

Attributes:

Name - Name of the *person*.

Department - semester - The semester and department *person* attends. An example could be Computer Science 3rd Semester.

Email / contact info - Contact information.

Behavior:

When a *person* is created in the system he has no roles. A *person* is affiliated with a *project* by creating a *role* for him in the *project*. That *person* is active, if he has a *role* in an active *project*. Likewise he is inactive, if he has no *roles*, or all the *projects* he has a *role* in are submitted and thereby inactive.

Role class

See person class figure 2.2

Definition:

A *role* is what affiliates a *person* with a *project*. There are two specializations of the *role* class, which are *member* and *supervisor*. A *role* describes what kind of relationship a *person* has with a specific *project*.

Attributes:

None.

Behavior:

A *role* is created when a *person* is affiliated with a *project*. It is deleted if the *project* or the *person* is deleted. It can also be explicitly deleted, which effectively disconnects a *person* from a *project*.

Literature class

See figure 2.3 on the next page

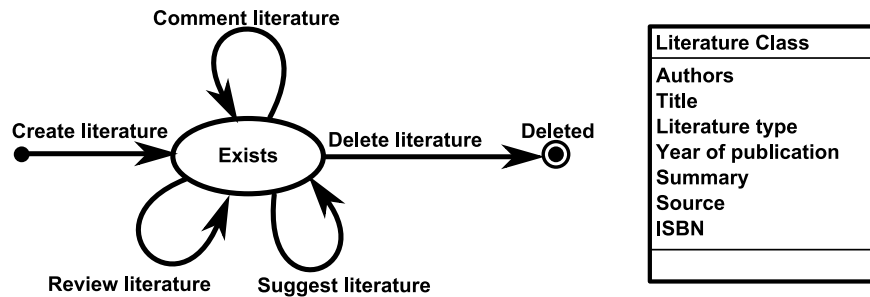


Figure 2.3.: Statechart diagram and attribute table for the *literature* class

Definition:

The *literature* class describes a piece of *literature*. It contains information about a specific piece of *literature*. Examples of *literature* could be: A book, an article, a journal, an Internet address, a figure etc.

Attributes:

Authors - Authors of the *literature*.

Title - The title of the *literature*.

Literature type - For instance book, report, article or Internet resource.

Year of publication - When was the *literature* entry published?

Summary - A summary of the *literature*.

Source - The source of the *literature*.

ISBN - An ISBN number if available.

Behavior: A piece of *literature* is entered into the system when a *reference* to it is created. It is only deleted if explicitly deleted.

Reference class

See figure 2.4

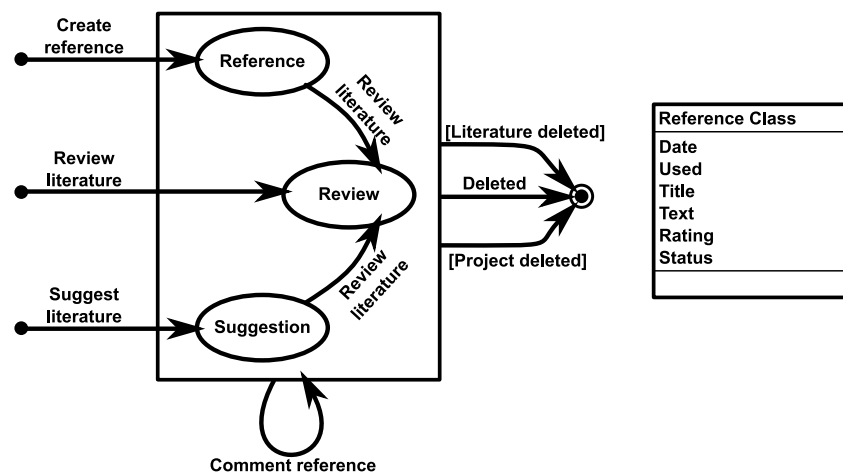


Figure 2.4.: Statechart diagram and attribute table for the *reference* class

Definition:

A *reference* connects a piece of *literature* to a *project*. It has two specializations,

review and *suggestion*. A *review* is created by a *group member* and contains a *review* of the relevance of the *literature* to the *project* that it is referenced to. A *suggestion* is made by a *supervisor* and is a *reference* to a piece of *literature* that the *supervisor* suggests that the *group* reads.

Attributes:

- Date - The date the *reference* was made.
- Used - A true/false statement that defines whether or not the *literature* is used in the *project* report.
- Title - (*Review* only) The title of the *review*.
- Text - (*Review* only) The *review*.
- Rating - (*Review* only) The reviewers rating of the *literatures* relevance to the related.
- Status - Whether the *literature* is available, in possession of a *group member* or not available.

Behavior:

A *reference* will be deleted if the referenced *literature* or *project* is deleted, furthermore a *review* is deleted upon deletion of the *member* who created the *review* and a *suggestion* is deleted upon deletion of the *supervisor* that added the *suggestion*.

Project class

See figure 2.5

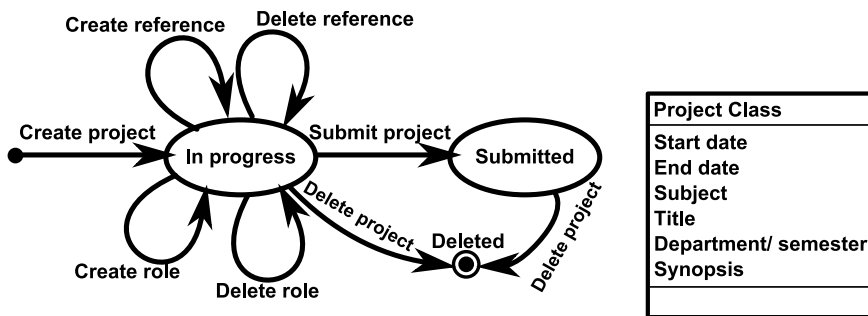


Figure 2.5.: Statechart diagram and attribute table for the *project* class

Definition:

Describes a *project*, it contains one or more *members*, and one or more *supervisors*. *References* to *literature* are added to the *project*.

Attributes:

- Start / end date - Start date is the date on which the *project* is begun. The end date is when the *project* is submitted.
- Subject - The subject of the *project*.
- Title - The title of the *project*.
- Department / semester - The semester and department of the *project*. An example could be Computer Science 3rd Semester.
- Synopsis - Synopsis of the final report.

Behavior:

A *project* is created with one or more *supervisors* and *members*. After the creation these *roles* can be assigned to the *project*. Assigned *roles* can also be removed, for

example in case of dropouts or reassignment of *supervisors*. A *project* is active when it has group *members* assigned and it is finished when the final report has been submitted. If the *project* is not expected to be finished, it can be deleted, causing cascading deletions of *references* and *roles*, since we assume that a project which is not finished does not have at sufficient level of quality.

Comment class

See figure 2.6

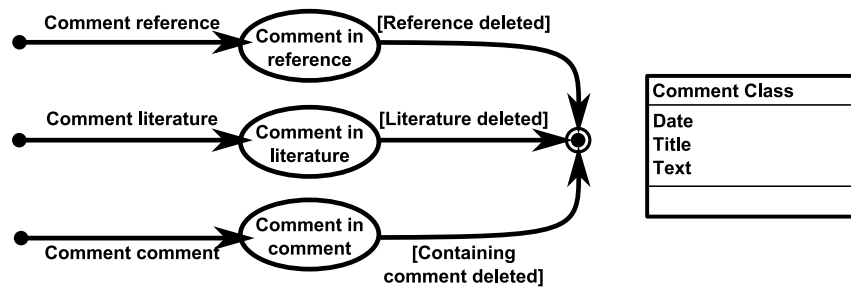


Figure 2.6.: Statechart diagram and attribute table for the *comment* class

Definition:

A piece of text that comments one *reference*, one piece of *literature* or another *comment*. It is created by one *person*.

Attributes:

- Date - The date the comment was made.
- Title - The title of the comment.
- Text - The text of the comment.

Behavior:

A comment is created by one *person* and continues to exist until the object, that the *comment* belongs to, is deleted. The comment will not be deleted if the creator is deleted because the comment can have other comments attached.

2.1.2. Events

An event is an abstraction of a process or activity in the problem domain. Identifying events is crucial as events provide the basis for defining the behaviour of the system. Table 2.1 on the next page contains the final events of the system.

Event	Project	Person	Role	Reference	Literature	Comment
Project created	+	*	+			
Project deleted	+	*	+	+		+
Project submitted	+	*				
Role created		*	+			
Role deleted		*	+	+		
Person created		+				
Person deleted		+	+			+
Reference created				+	+	
Reference deleted				+		+
Literature reviewed				+	+	
Literature suggested				+	+	
Literature deleted				+	+	+
Literature commented						+
Reference commented						+
Comment commented						+
Comment deleted						+

Table 2.1.: Final Event table: To the left the events and to right the objects they affect. Events marked by + can only occur once, events marked by * can occur more than once.

Application domain

3.1. Usage

The main goal of this section is to describe the usage of the system within the application domain.

3.1.1. Overview

When a student joins the university he is invited to create a user account in the article system. This account enables that student to create and be member of projects stored in the system. At the start of each semester new project groups are formed. After the groups have been formed one of the members from each group creates a new project in the system. When the project has been created, the rest of the group members can then be affiliated with the project in the system by the project creator.

In the beginning of the project a large amount of literature is read by the group. After a member of the group has found a piece of literature she considers relevant to the project, he or she adds an entry to the project containing various information about that piece of literature. Now other members of the group are able to comment on that entry.

In case a piece of literature read by the group is not directly relevant to the project it can be added to the system itself, instead of the project. It is possible for the project group to view the list of used literature of former and active project groups.

The supervisor of the group can add an entry to the project with suggested literature.

Finally the group can generate a bibliography for the report by exporting the list of literature applied in the project.

3.1.2. Actors

This section describes the actors who interact with the system. An actor is an abstraction of different types of users or systems.

Based on the overview in section 3.1.1 we can identify the possible actors.

Definition of actors

The Group Member (student): A member of a project group is a user of the system. The user is typically a university student with fair computer qualifications.

The Supervisor: A user can be assigned to a project as supervisor. A supervisor could typically be a member of the teaching staff. In both cases it would be reasonable to presume they have fairly good computer qualifications.

The Moderator: There is a need for someone to delete failed projects. If a project never gets submitted its use of literature might not be valid and therefore not relevant to other groups. A moderator could typically be a secretary of the department or a member of the IT-staff (as mentioned in the Stakeholder analysis in section 1.2.2 on page 9.)

Personas



Name	Andy Gibson
Age	35
Occupation	PhD-student
Nationality	Danish

Andy Gibson might be a typical user of the system. Andy Gibson is a 35 year old PhD-student at Aalborg University. He is currently working on his own PhD project "How to use computers to generate artificial life forms" a project he has almost finished. Before he started on his PhD he was working for a local software company for 5 years. Apart from his PhD he is supervising a number of project groups and master students at the department of computer science. As a supervisor he often uses an enormous amount of paper to keep track of how the project groups are doing. He often sends emails to his project groups with literature suggestions for them to read.

He lives in a small flat with his girlfriend and their one year old boy. As a person he is very tolerant and honest. He is also a member of the elite part of the Danish National Guard, but after he became a father, he does not have much spare time.

Andy's computer skills are good as long that he do not need to work with computer hardware. He has worked with a lot different operating systems, including Windows (which he prefers), MacOS and different types of Linux.



Name	Brian Jensen
Age	23
Occupation	Student
Nationality	Danish

Brian Jensen is a 23 year old student at the DAT1/INF1 semester at Aalborg University. His studies aside he works as a consultant in a small computer company, where his area of expertise is Linux servers and virtual servers. He has been working for the company for 5 years. When Brian is not studying or working he likes to drink beer with his friends, work on his own projects and listen to heavy metal. Brian lives in a very small and very unorganized apartment in the center of Aalborg.

Brian is the type who likes to take the initiative in his project groups and have the overview of the projects progress. But he is also a bit lazy, and he dislikes writing or saying the same thing twice.

Brian has a large amount of knowledge about computers; he has been working with computers since he was 5 years old. But he is impatient; if there is a piece of software he can not get working within the first 5 minutes he finds something else. Brian really hates Java and anything programmed in Java and he is chairman of the local "I hate Java" society.

3.1.3. Scenarios

Describing the application domain in detail would generate a lot of various information. This information might not all be relevant to the development process of the system. Therefore scenarios are a possible way to achieve an appropriate abstraction from technical detail. During this activity aspects which we would not discover during other analysis activities become unveiled. The section below is a collection of possible scenarios.

When we mention "the system" we always refer to our article cataloging system.

The scenario

After the project launch the 5th semester Computer Science student John from the project group x214d creates a user account in the system. At the library he has found a book called "AI – a question of perception" which he thinks would come to good use in x214ds project named "AI". Since it is the first time John uses the system, he has to create a new project called "AI". Likewise he must create a new literature entry, in order to be able to write information (such as authors) about the book into the system. Furthermore he reviews the book, in the sense that he in textual form explains why he thinks the book might be relevant to the "AI" project. At any given time after the project creation he is able to add as many pieces of literature to the project "AI" as he wishes.

John then wants to let his groupmates be part of the "AI" project. To do this he encourages his group mates to create their own user accounts. When this is done John associates these user accounts with the project "AI".

Afterwards his group mate Allan wants to see which literature related to "AI" his group mates have encountered so far. Allan encounters the book "AI – a question of perception", and finds out John has made a comment on which parts of the book is relevant to their project. Allan already knows the book and he disagrees on John's point of view and therefore he makes a new comment to Johns review. He does this by selecting Johns review, entering his comment and submitting it.

At a certain time during the project period John and Allan's group mate Ann is writing a part about artificial perception in the project report, she needs to underpin her assertions with literature references. Ann searches in the system for literature concerning this topic, finds the information about "AI – a question of perception", and realizes that it could be useful. Ann decides to check whether other project groups with similar topics have found sources about artificial perception. By doing this she realizes that the neighbor group x216d has been using the book "Artificial Perception for dummies", which she then borrows from the library.

The PhD student Mary is supervisor for the group x214d. After receiving a temporary edition of the groups report she finds the part about artificial perception insufficient. Mary has a book that she knows would be suitable in that context. She therefore logs into the system and adds a suggestion of that particular book to the group.

At a subsequent group meeting Mary tells the group that the part about artificial perception is not good enough and that she has added a suggestion to the group in the system. Ann, who wrote the section about artificial perception, locates the suggested book which enables her to correct the problematic part of the report.

At the end of the semester John, Allan and Ann want to hand in their project with a full list of literature. To do this they ask the system to generate a BibTex-file¹ which they can apply in their project report. When the "AI" project has finished it becomes inactive, but still available in the system, resulting that other future project groups with similar topics are still able to find literature used in the "AI" project.

3.1.4. Use cases

Figure 3.1 on the following page depicts the relations between actors and use cases. In the following the use cases are described in further detail.

¹**BibTex**: A literature handling system for the typesetting system \LaTeX

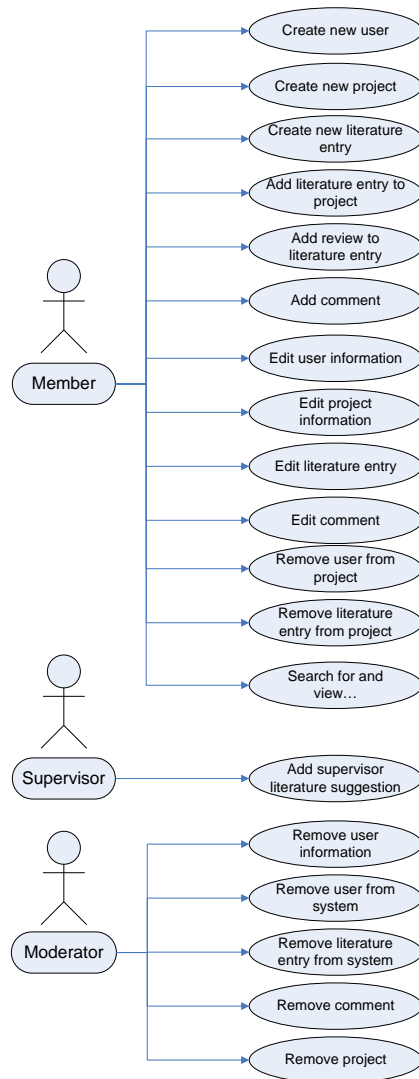


Figure 3.1.: An overview of which actors can be involved in which use cases.

Create new user

The first thing a person needs to do when he wants to use the system is to establish himself as a user (i.e. make a user account). When opening the system it first asks for a username (and password). If the user is not represented in the system yet, the system demands that he creates a new user. He enters his name followed by a department/semester specification and his contact info (email address). Finally he has to enter the desired username and password. This use case is illustrated in figure 3.2 on the next page.

Objects involved Person

Create new project

As soon as a user has created a user account, he can create a new project. To do this the user has to enter a project title and a project subject followed by information about which department and semester the project is related to. Afterwards the system suggests the

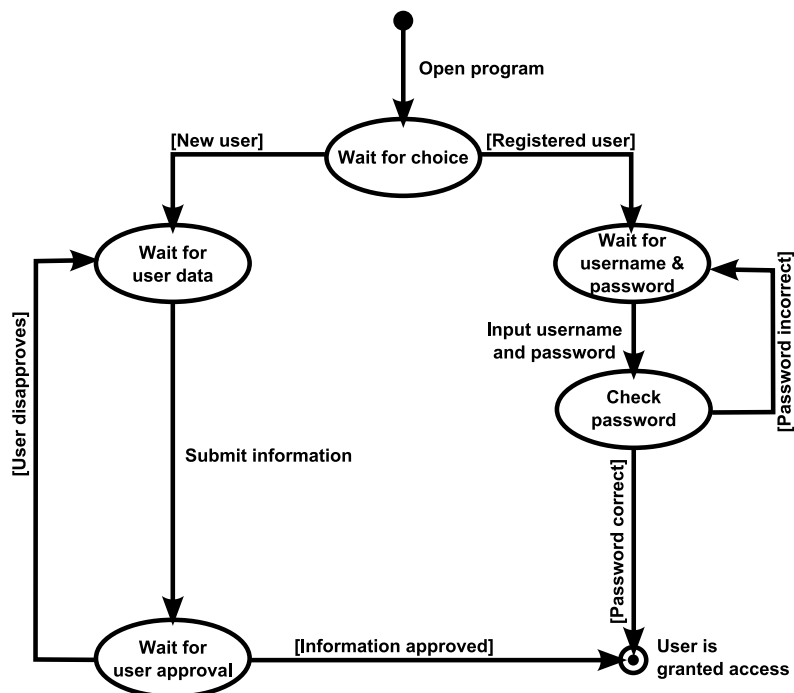


Figure 3.2.: Statechart diagram of the use case: Create new user.

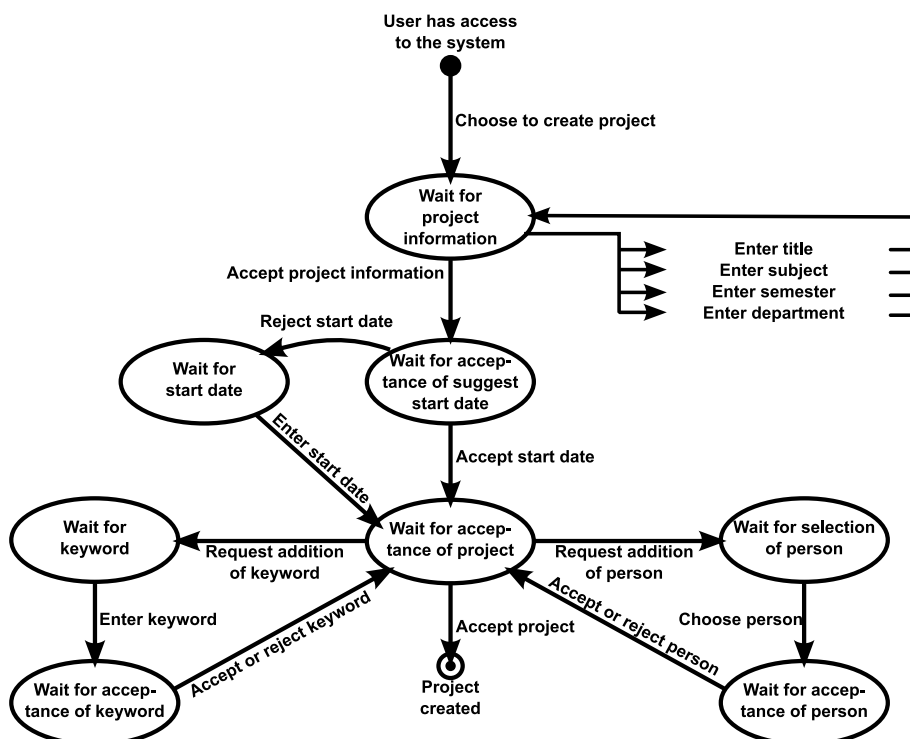


Figure 3.3.: Statechart diagram of the use case: Create new project.

current date as the project start time, and offers the user with the following two options: He can choose to accept the suggestion, or he can alter the date. Finally the user can optionally choose to specify some project related keywords, and to add members and supervisors before he creates the project. This use case is illustrated in figure 3.3.

Objects involved Person, project, role, member, supervisor

Create new literature entry

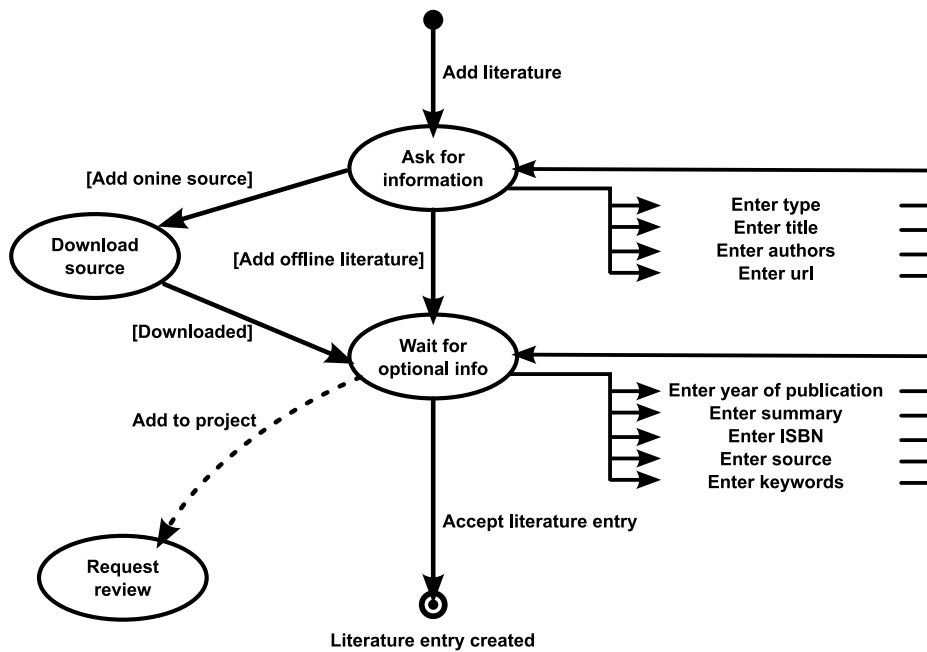


Figure 3.4.: Statechart diagram of the use case: Create new literature entry.

A user of the system is always able to create a new literature entry. To begin this process he has to specify whether he is adding an Internet source, or an offline source. After this he has to specify the type of literature (i.e. book, article, journal etc.) followed by the title and the name of the authors. If the user chooses to add an Internet source, the system requests for an Internet address, and suggest the user to manually create a backup of this Internet source. After this there is an amount of optional information (depending on the literature type) he is able to enter in arbitrary order:

- Objective information: summary, ISBN and keywords.
- Subjective information: review and reference to a project.

The next step is the creation of the literature entry. This use case is illustrated in figure 3.4.

Objects involved Person, literature, reference, review

Add literature reference to project

As a literature entry is created it is not necessarily affiliated with a project. To do this a user first browses for the literature entry, and selects which project he wants to affiliate it with. This creates a reference between the objects. He is only allowed to add references to projects he is either the creator or member of. When adding a reference it is possible to mark it as used in the project report.

Objects involved Member, literature, reference, project

Add review to literature entry

To review a piece of literature, a user first has to locate the literature and choose to create a review of it. A user can only review literature in connection with a project. When the user has written his review he has the opportunity to rate the entry as well. A user can also select a suggestion and review the suggested literature, turning the suggestion into a review.

Objects involved Member, supervisor, literature, review, project, reference

Add comment

A user has the opportunity to add a written comment at any time. First the user browses for the literature entry, review, suggestion or comment he wishes to comment on. Then he enters his comment and submits it.

Objects involved Person, member, supervisor, literature, comment, reference

Add supervisor literature suggestion

When a user has the role of project supervisor he can suggest literature by choosing a project followed by literature entry. If the literature entry does not exist in the system the supervisor is enabled to add it.

Objects involved Supervisor, suggestion, literature, project

Edit user information

A user can alter his user account by choosing to manage his own account. By doing this he can edit information such as e-mail address, password and department information.

Objects involved Person

Edit project information

The creator of a project is enabled to change project information like title, subject, keywords, department, and semester. The creator does this by selecting the project and informing the system that he wants to alter the project's information.

Objects involved Member, project

Edit literature entry

A user can add, delete and alter information about literature entries that he has created in the system by selecting the particular literature entry and informing the system that he wants to edit details about it.

Objects involved Person, member, supervisor, literature

Edit reference/review

A user can edit references or reviews. The creator of a review can make changes to the review. Any user affiliated with the project can change information about a reference and whether or not the referenced literature is used in the project.

Objects involved member, reference

Edit comment

When a user has added a comment he can choose to alter the content of it by clicking the particular comment and then change the comment, but the user is not allowed to remove the comment himself.

Objects involved Person, member, supervisor, comment

Remove user from system

The moderator can remove a user by selecting him and informing the system that the user should be removed. When deleting a user the entries created by that particular user still remains in the system, but the user cannot access to the system.

Objects involved Person

Remove literature entry from the system

Literature can be removed from the system by the moderator. The moderator locates the literature in the system and then instructs the system to remove it.

Objects involved Person, literature, reference, comment, review, suggestion

Remove comment

Comments can be removed by the moderator, but when a comment is deleted all of the subcomments will be removed as well. When the moderator finds a comment that he wants to delete he can remove it by instructing the system to do it.

Objects involved Person, comment, literature

Remove project

When the moderator wants to remove a project he selects the project and then instructs the system to remove it. To ensure that this action does not happen by an accident the system requires that the moderator confirms the action.

Objects involved Project, role, reference, comment

Remove user from project

The project creator can remove a user from the project, by selecting the user and then instructing the system to remove him from the project.

Objects involved Person, member, project

Remove reference from project

References to literature used in a project can be removed by any member of a project. This also deletes all comments to the reference.

Objects involved Reference, comment

Search for and view literature entry

All the users of the system are able to search for and view all literature in the system. To do this a user instructs the system to start a search for literature, followed by information about what he is searching for. The result is based on how it is rated in different projects with similar topics and keywords.

Objects involved Person, literature

Search for and View project

Any user of the system can search for other projects. To do this the user instructs the system to start a search for another project, and specifies what he is looking for.

Objects involved Person, project

Search for and View user

Any user can search the system for other users in order to view contact info and project affiliation.

Objects involved Person

Search for and View review

Any user can search the system for and view reviews.

Objects involved Person, reference, review, comment, literature

3.2. Functions

In this section we will list all the functions of our future system. We will make a more in-depth description of some of the functions assessed to be non-trivial.

3.2.1. Complete list of functions

The complete list of functions can be seen in table 3.1 on the following page. The functions which are simple are considered trivial, i.g. DeleteComment simply removes an object. The name of the function is sufficient to tell what the function does.

3.2.2. Specification of complex functions

We have categorized the functions that are more than simple to be non-trivial and therefore suitable for a more detailed description. The complexity is based on a simple assessment of the function; if the function only is updating the database it is simple. Else if the function is searching or doing updates several places in the database it is regarded as more complex.

Function	Function Type	Complexity
Search()	Compute	Very Complex
AssignPersonToProject()	Update	Medium
ExportBibliography()	Compute	Medium
ShowRelated()	Compute	Complex
DeleteProject()	Update	Medium
CreateProject()	Update	Simple
SubmitProject()	Update	Simple
CreateRole()	Update	Simple
DeleteRole()	Update	Simple
CreatePerson()	Update	Simple
DeletePerson()	Update	Simple
CreateReference()	Update	Simple
DeleteReference()	Update	Simple
ReviewLiterature()	Update	Simple
SuggestLiterature()	Update	Simple
DeleteLiterature()	Update	Simple
CommentLiterature()	Update	Simple
CommentReference()	Update	Simple
CommentComment()	Update	Simple
DeleteComment()	Update	Simple

Table 3.1.: List of functions

Specification of Search(): Search() is our main search function in the system. This is the most complex function in the system primarily because of the "sorted by relevance" output.

Input:

A search query

Do:

Search through keywords for related projects and literature
 Search through projects for title, subject and synopsis
 Search through literature for author, title and summary

Output:

All found objects sorted by relevance to the search query.

Specification of AssignPersonToProject(): AssignPersonToProject() is one of the basic functions in the system. The reason for the complexity being higher than simple is that the function requires interaction between at least 3 different classes.

Input:

A person, a project and the person's role in the project

Do:

Create a new role of the specified type and assign it to the person and to the project.

Output:

None because it is an update function.

Specification of ExportBibliography(): ExportBibliography() is the function for exporting the actual bibliography into various formats to be used in the report.

Input:

A project and a type of output (e.g. BibTeX)

Do:

For each reference in project check if reference is used. If true then add to bibliography. Save bibliography and export to the given type of output.

Output:

A bibliography of the given type.

Specification of ShowRelated(): The ShowRelated() function is a searching function that finds relations between projects and literature etc. based on keywords. It has a rather high complexity.

Input:

A project or a piece of literature

Do:

Search through projects or literature and return related objects

Output:

Show related objects

Specification of DeleteProject(): DeleteProject() involves at least 3 different classes and has therefore higher complexity than simple.

Input:

A project

Do:

Remove all references from project

Remove all roles

Remove project

Output:

None because it is an update function.

3.3. User interface

3.3.1. Usage aims

There are two properties in focus when designing a user interface, namely, *usability* and *user experience* [5]. How important these goals are depends on the system. In other words a system has to be tailormade in order for it to suit the application domain. Table 3.2 on the next page displays a number of goals [5, p. 14 and 18] and the X's show how we have prioritized them in our article cataloguing system. The overall pattern in table 3.2 is that usability is of higher priority compared to user experience.

		Crucial	Important	Less important	Irrelevant
Usability	Effectiveness	X			
	Efficiency		X		
	Safety		X		
	Utility	X			
	Learnability		X		
	Memorability		X		
Experience	Satisfying	X			
	Enjoyable			X	
	Fun			X	
	Entertaining				X
	Helpful		X		
	Motivating			X	
	Aesthetically pleasing			X	
	Creativity supportive			X	
	Rewarding	X			
	Emotionally fulfilling			X	

Table 3.2.: The usage aims of our system.

3.3.2. Conceptual model and forms of interaction

A conceptual model is an overall take on how the interaction between human and system should take place. We distinguish between the following forms of interaction[13]: Menu, Dialogue, Form-filling, Browsing, Manipulation and Instruction.

Forms of interaction in carrying out specific activities

In order to be able to determine the conceptual model of our system we have listed (see table 3.3) a number of typical interaction activities that users would perform alongside some possible ways these interaction activities could be realized in the system.

Activity	Possible forms of interaction
Create new user	Menu, Dialogue, Form-filling
Create new project	Menu, Dialogue, Form-filling
Add literature entry	Menu, Dialogue, Form-filling
Review/rate literature	Menu, Browsing, Form-filling, Manipulation
Add comment to literature review	Menu, Browsing, Form-filling
Add literature to project bibliography	Menu, Browsing, Manipulation
View other bibliography	Menu, Browsing

Table 3.3.: Some possible forms of interaction in carrying out specific activities.

From table 3.3 it is clear that all of the activities could be accessed from some menu arrangement. Moreover, form-filling (which in essence can be regarded as a form of dialogue) is rather dominant in situations where the user inputs data such as project name, keywords, literature entries and literature reviews. After this "input phase" (which probably is ongoing during most of a project period) the dominant activity must be *browsing*, since the users need to search for and eventually extract information from the system.

On this background and the fact that one of the key purposes of the system is to provide

Use-case	Form of interaction	Activity
Create new user	Dialogue	Adding
Create new project	Dialogue	Adding
Create new literature entry	Dialogue	Adding
Add literature entry to project	Dialogue	Adding
Add review to literature entry	Dialogue	Adding
Add comment	Browsing/Dialogue	Adding
Add supervisor literature suggestion	Dialogue	Adding
Edit user information	Manipulation	Editing
Edit project information	Manipulation	Editing
Edit literature entry	Manipulation	Editing
Edit comment	Manipulation	Editing
Remove user from system	Instruction	Updating
Remove literature entry from system	Instruction	Updating
Remove comment	Instruction	Updating
Remove project	Instruction	Updating
Remove user from project	Instruction	Updating
Remove literature entry from project	Instruction	Updating
Search for and view literature entry	Browsing	Searching
Search for and view project	Browsing	Searching
Search for and view user	Browsing	Searching
Search for and view review	Browsing	Searching

Table 3.4.: List of forms of interactions in our system

a project group with an overview of its literature resources, we determine the main conceptual model to be **browsing** with pronounced elements of **dialogue**. In some occasions manipulation would be an obvious form of interaction as well. For instance the system could enable the user to add a piece of literature to a project bibliography by applying a drag-and-drop mouse action.

3.3.3. General interaction model

The list shown in table 3.4 depicts forms of interaction based on the use cases.

The general interaction model of our system is shown in figure 3.5 on the next page. The purpose of the literature entry view is to show the selected literature entry with its associated attributes, possible review, comments and project references. A solution could be to include review and comment views directly into the literature entry view. At this point in the development process it is our intention to make a designated moderator view through which all system tasks can be carried out. This is why we have left out the moderator specific use cases in the figure 3.5 on the following page.

3.3.4. Technical platform

As defined in the System Definition in section 1.2 on page 8, the system should be able to run on a standard PC supporting the .NET framework and having a network connection. The minimum system requirements for the .NET framework is a Windows 98 computer [9].

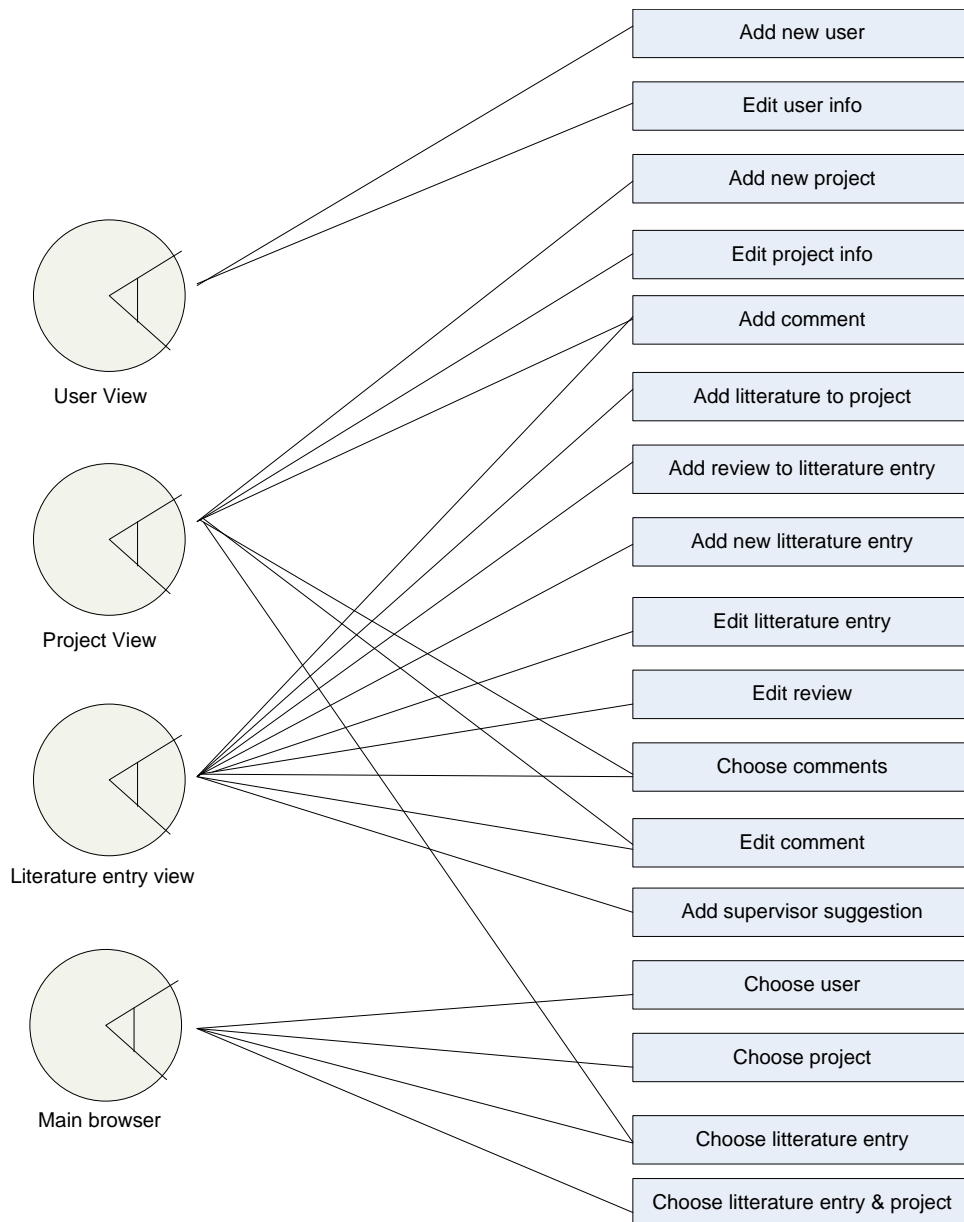


Figure 3.5.: General interaction model of our system.

The reason for the requirement of the .NET Framework is that we are going to develop the system in the C# programming language, which needs .NET framework to run.

Strategy for Future Development

This chapter contains information about the future development of our article system.

The next step in the process is to make a design document, which describes the system design in detail. The design document is the link between the analysis and the actual implementation of the system. There are several activities to be done in the design phase:

- Detailed design of the functionalities of the system
- The design of a more detailed class diagram
- Consider the design of the Graphical User Interface (GUI)

After making the design document we will implement our system, meaning the actual programming of the system. There are also activities like testing the system with future users and documentation of the system. General evaluation of the system is also part of the implementation process.

4.1. Economy

We believe that it is possible to make an implementation of our article system within the time we have available for this project. There should be no need to buy new hardware for us to develop or run our application.

Part II

Design

This document describes the design of a literature management system called SCRAML. It follows the guidelines for design set by the Object Oriented Analysis and Design[6] method. First it defines the criteria of the design, then the details of how the system should be implemented.

The Task

5.1. Purpose

The main purpose of the literature management system is to aid university student project groups in managing their literature sources and to find relevant literature by studying what literature is used in other similar projects. In addition to this the system will help project supervisors by letting them view what literature a project group is using in order to gain insight with the project, make suggestions and criticize a particular piece of literature.

5.2. Corrections to the Analysis

Important corrections done in the analysis document since first project review:

- Mapped FACTOR criteria onto the system definition. See section 1.2 on page 8.
- Added description of the current situation. See section 1.3.1 on page 10.
- Moved description of the two rich pictures. See section 1.3.2 on page 12.
- Changed the class diagram of the problem domain. See section 2.1 on page 15.
- Changed the use case diagrams. See figure 3.1 on page 24, 3.2 on page 25, 3.3 on page 25 and 3.4 on page 26.

The new version of the analysis document has been included.

5.3. Quality Goals

The quality of the system greatly influences how successful it will be. In order to ensure this quality we have decided which criteria are most important as seen in table 5.1 on the facing page. In the following is a short justification for the rating of each goal.

- **Usable:** This is an essential goal. The purpose of introducing a system is to make a positive change in the way information of the problem domain is handled, and to aid the actors of the application domain in carrying out their work tasks. In other words, a system does not make any sense if it is not usable.
- **Secure:** Since our system is neither a system to control a high risk application domain, nor a system to handle very sensitive data, security is not that significant. However the login facility must be secure, in order to insure that the right users have the correct rights within the system.
- **Efficient:** Efficiency is assessed to be very important, since users quickly lose patience with inefficient systems.

Criteria	Very important	Important	Less important	Irrelevant
Usable	X			
Secure			X	
Efficient	X			
Correct		X		
Reliable		X		
Maintainable			X	
Testable		X		
Flexible			X	
Comprehensible	X			
Reusable			X	
Portable			X	
Interoperable			X	

Table 5.1.: Table of quality goals for system properties.

- **Correct:** It is important that the system is correct – i.e. behaves as expected, because the system shall support the students projects. If the system is incorrect there will be no reason to use the system at all.
- **Reliable:** An unreliable system is undesirable in most situations. Reliability is closely connected to system robustness – the notion of the system not easily breaking down. Our aim is to make the system rather robust, and has accordingly categorized reliability as important.
- **Maintainable:** This goal has low priority because the system is part of a study project and thus is not expected to undergo further development and adjustments after project report submission.
- **Testable:** We are expected to perform and document systematic tests of our system, and for that reason testability is rated as important.
- **Flexible:** See "Maintainable".
- **Comprehensible:** Comprehensibility is a measure of the effort needed for other developers than those who actually developed the system to gain an understanding of the system and how it works. The system is – as stated above – part of a study project, and for that reason comprehensibility is rated very important. This to insure that the system is comprehensible both to ourselves and to the people who are going to review our work.
- **Reusable:** See "Maintainable".
- **Portable:** Portability is less important in this case since we restrain our design focus to the Windows platform. We are aware that other platforms such as Unix and Linux are used at universities, but we assume that Windows is the most widely used operating system.
- **Interoperable:** The system is stand alone and need not be linked to other systems. For that reason interoperability is considered less important.

Technical Platform

In this chapter we describe the technical platform of the article cataloguing system. The subjects covered are the equipment to run the software on, the software interfaces our system needs to work and the design language applied.

6.1. Equipment

In overall terms the system consists of two parts – a system server and one or more PC clients from which the users through network (LAN¹ or Internet) can access the data which will be stored on the system server. Most of the computations will be done on the server side which also implies that the system requirements for the server will be rather high and the client requirements rather low.

6.2. System Software

The client part of the system will be developed for computers running the operating system Microsoft Windows. The only further requirement is that the client computer has the .NET 2.0 framework installed. The minimum recommended system requirements for the .NET framework [8] are a Pentium 90 MHz CPU and 96 MB RAM. The specifications of modern PCs and laptops exceed these requirements by far.

In the light of the fact that we here at the Department of Computer Science have Unix servers, we assume that the typical university servers are running Unix. However due to the lack of developer experience, we focus on developing the server software for the Windows platform. Only the client software and not the server software will contain Windows Forms and it should for that reason be rather easy to adjust the server software to the Unix platform. But since portability is rated less important (according to the table of quality goals table 5.1 on the previous page), we will not spend resources trying to achieve this.

6.3. System Interfaces

RMI² will be used as the interface between the client and the article cataloguing system server. The RMI client invokes remote methods on the server and data is sent back to the client from the server.

The systems network interface is interchangeable therefore any type of external interface could be used on the server-side of the program. For example a web-service interface or a HTTP server could be implemented.

¹Local Area Network

²Remote Method Invocation

6.4. Design Language

The applied design language is the subset of UML³ which is used in Object Oriented Analysis & Design [6] to create design class diagrams to depict classes, their attributes, methods and relations and sequence diagrams to illustrate how objects interact during a certain period of time.

³Unified Modeling Language

Architecture

7.1. Design criteria and requirements crucial for the architecture

Table 7.1 elaborates on the quality goals¹ rated "Very important" of table 5.1 on page 39. Furthermore the two important factors persistency and error handling are covered.

	Factor	Measures and quality scenarios	Variability (current flexibility / future evolution)	Impact of factor (and its variability on stakeholders, architecture and other factors)	Priority for success	Difficulty or risk
Usability	The system is intended to be used by many different users at a university	First time users should be able to gain an overall understanding of how the system works within a time frame of 10 minutes.	Current flexibility – If this requirement is not acceptably maintained the system will not be successful.	High impact on the largescale architecture and the detailed UI-design. To ensure that primary stakeholders utilize the system it is important with a high level of usability. If the system fails to comply with the usability demands the primary stakeholders will seek out alternate solutions.	H	M
Efficiency	It is crucial to the success of the system that the users easily can input or get the information they want.	Access to user inquired information should be quick (including user interface navigation), and relevant search results should be displayed within 3 seconds.	Current flexibility – So far slow response is acceptable because we are not using a database system but a XML-based server. – Future flexibility – It is the intention that the system in the future will be based on a database server instead of XML files - This will enhance the efficiency of the system.	High impact on the largescale architecture, data access, algorithms and UI design.	H	H
Comprehensibility	It is crucial that it is possible to get an overview of the entire system in order to fix errors, and to help other developers understanding the system.	The source code should be well documented and commented. The use of well-known design patterns can help increase comprehensibility.	Current flexibility – We do not allow any flexibility in this step because there are different developers working with the system that needs to understand how specific functionality works.	High impact on the largescale and component architecture.	H	H
Persistency	Persistency is essential. The system would not make any sense without the opportunity to add, search, edit and retrieve data.	It should be possible to store data entries in the system, and search for and edit these entries at a later system run.	Current flexibility – Crucial for the success of the system hence no flexibility.	Very high impact on the success of the system, and is for that reason a crucial part of the architectural design.	H	H
Error handling	The system must be robust and forgiving concerning the most common errors.	Error handling should be implemented and error messages should be kind, informative and provide suggestions on how to solve the problem.	Current flexibility – Most errors must be well handled – especially errors caused by the user. – Future flexibility – Error-handling that is not yet implemented should be easy to implement.	Errors can arise from both internal system implications and the user interaction. The latter is an important aspect in order to honor the demands given in the table describing the usage aims for the user experience (satisfying, rewarding and helpful).	H	M

Table 7.1.: Factors crucial for the architecture (H = High, M = Medium and L = Low).

All the factors in the table are based on things that we believe is most important in order to ensure that the system will be successful. This is why we rated the priority for

¹In this context the three goals are considered "factors" that influence the architecture.

success high for all the factors in the table.

The difficulty or risk ratings are based on how complex an eventual change to the system software would be if the specific criteria have not been successfully filled.

7.2. Generic design decisions

The following is a description of the generic design decisions. In other words our solutions to recurring problems.

Architecture patterns

We apply a Closed-Strict layered architecture pattern which means that a given layer only can use operations from the layer immediately below. Furthermore we follow a Client-Server pattern where most of the work is carried out on the server-side, and only the GUI component is handled on the client-side.

The GUI

The GUI is designed to fit in the windows environment because we are using Visual Studio 2005 we will not put very much effort in this. The designer tools in this IDE do most of the design for us, so in most of the program we will use the standard controls provided.

- *Affordance*: It has to be as clear as possible how to interact with the program.
- *Mapping*: The mapping between the controls in the GUI and the objects they affect has to be as direct as possible.
- *Consistency*: There are two types of consistency: Internal in the system and External related to other systems. Internal refers to for instance consistent use of terms throughout the system, while External refers to resemblance with the way existing systems are dealing with similar issues.
- *Feedback*: When the users performs an action the system has to respond explicitly to inform the user that it is working on his request. This is done for instance by changing the cursor icon to an hour-glass when the request takes a little time to carry out.

Persistency handling

Data about users, projects, literature entries, etc. is stored on the server side as files. The persistent layer is the lowest layer in the component architecture diagram as illustrated in figure 7.3 on page 46. On every program run the entire data collection is loaded into the model component for efficient, random access – for instance for searching purposes. When the editing is finished the data collection is saved onto the persistent layer.

Documentation style

Standard Visual Studio 2005 XML documentation style is applied, with additional comments describing namespaces.

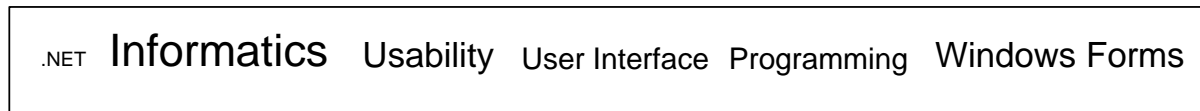


Figure 7.1.: An example of a tag cloud

Coding style

Standard Visual Studio 2005 coding style (for instance the use of curly brackets and indentation) is applied.

Naming conventions

All names used in the system design must be descriptive of what they represent. Furthermore the following rules apply:

- *Variables*: Must be written in lowercase. In case of concatenated words the first letter in every word must be in uppercase e.g. "dateOfBirth".
- *Instances of classes*: Must be written in lowercase.
- *Methods*: Is capitalized.
- *Properties*: Named the same as the corresponding variable if they exist, except that it is capitalized.
- *Classes*: Is capitalized.

Broad folksonomy

The first of the following pictures illustrate how selecting terms for queries for projects and literature should be visualized in our system. The second demonstrates how to define those terms when viewing the item's properties. The system presented here is based around the concept of broad folksonomies[4], a term first coined by Thomas Vander Wal[14] to describe a method for social bookmarking. See also Appendix D on page 120.

An alternative way to select search terms in the system (as opposed to typing them out manually) is the tag cloud. This should be a part of the final system as illustrated on Figure 7.1. The tag cloud illustrates the frequency of a keyword in the system by varying the text size. In this example Informatics is the most used keyword.

The sketch in figure 7.2 on the facing page illustrates how the relevance of a keyword is represented for the user when viewing a piece of information, such as a project or a piece of literature. The keywords of each item must be ordered by relevance, as shown by way of a progressbar. The small plus and minus labels beside each item's text and progress bar indicate buttons where the top items are keywords the user has already assigned to this item. The lower items (with the plus) indicate keywords that other users have assigned to the item.

Clicking the minus and plus buttons will show the user's disagreement or agreement with the keyword. If he clicks the minus button will remove the user's association of the keyword with the item (that is: the user untags the item). If on the other hand he clicks the plus button outside one of the keywords he has not yet assigned himself, he assigns this keyword with the piece of information (that is: he tags the item). Thus respectively decreasing or increasing the keyword's relevance in the system to this item of information.

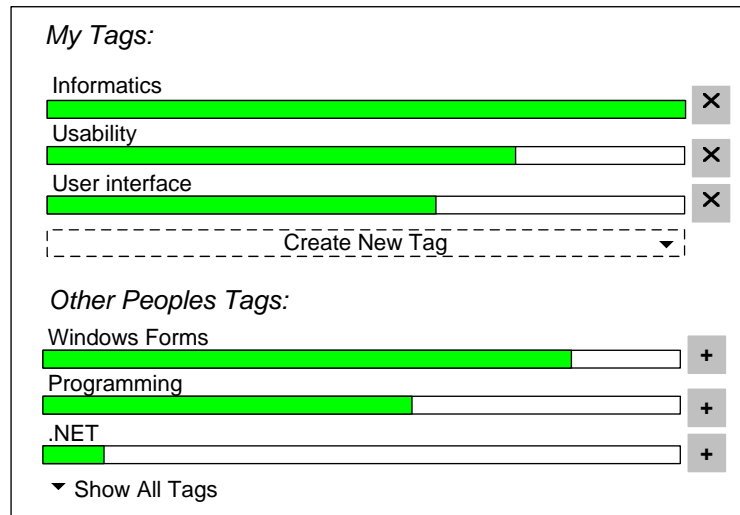


Figure 7.2.: An example of the keyword list view

7.3. Component Architecture

The diagram in figure 7.3 on the next page depicts the overall layered client-server component architecture of our system. The top layer consists of the clients 1 to n, where a client is situated on a users PC/laptop. Each client is an aggregation of a GUI component and a Client-API² which provides connectivity to a server component. The architecture is according to section 7.2 on page 43 Closed-Strict. The Client-API and Server-API are transparent. Further description is detailed in figure 8.1 on page 49. The system implements a 6-layered architecture instead of the OOAD proposed 3-layered architecture (Model, Functionality and Interface). In order from top to bottom the layers are:

GUI The user-interface presented to a user of the system

Client-API The client side of the network architecture.

Server-API Server side of the network architecture.

Accesshandler Handles authentication of users and their permissions in the system.

Catalogue Contains the model of the system and all functions to modify that model.

Persistent Data Handles saving the model to a persistent data source (eg. harddrive).

Using the 6-layered architecture allows for interchangeable implementations of each layer. The network architecture (Server-API and Client-API) is completely interchangeable, allowing the use of several different protocols at the same time. This is achieved by ensuring that the interface provided to the GUI from Client-API is identical to the interface provided to Server-API from the Accesshandler. Persistent Data is also completely interchangeable, which allows developers to use custom implementations, saving the model data to a variety of persistent data types (e.g. different filetypes like XML and proprietary binary files or saving data directly via network to a server).

²Application Program Interface.

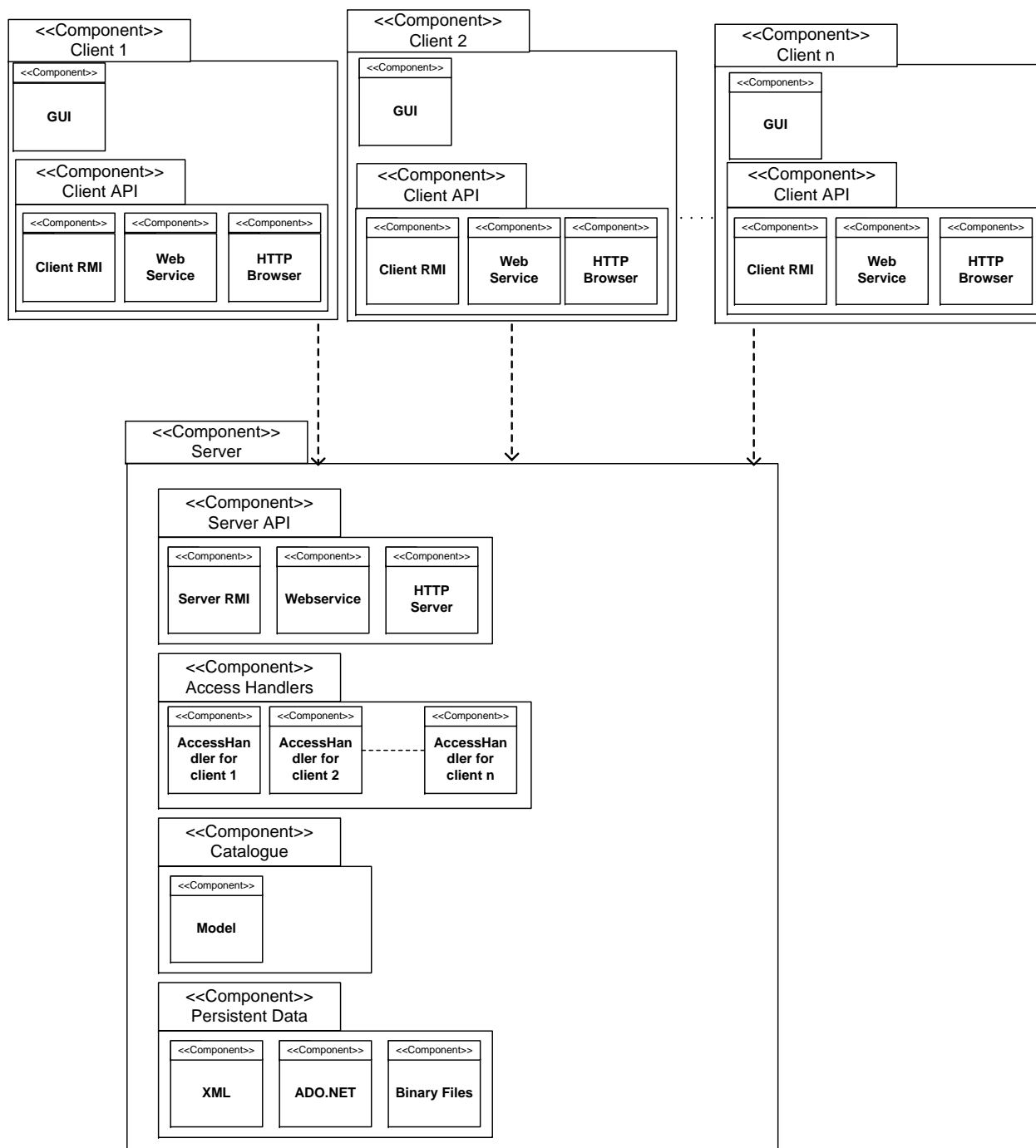


Figure 7.3.: Diagram depicting the component architecture of the system.

7.4. Exemplary Design

In the following we describe some use case scenarios by using sequence diagrams. This is done to show how some different use cases will be handled by the system. The method invocation arguments are referred to as "arg". We have chosen the use-cases, that were realistic to implement at time of writing. Furthermore these use cases illustrates important parts of the system. We consider the "create_project" -diagram as a nontrivial part of the system, and "register_user" and "login" are considered trivial.

7.4.1. Register user sequence diagram

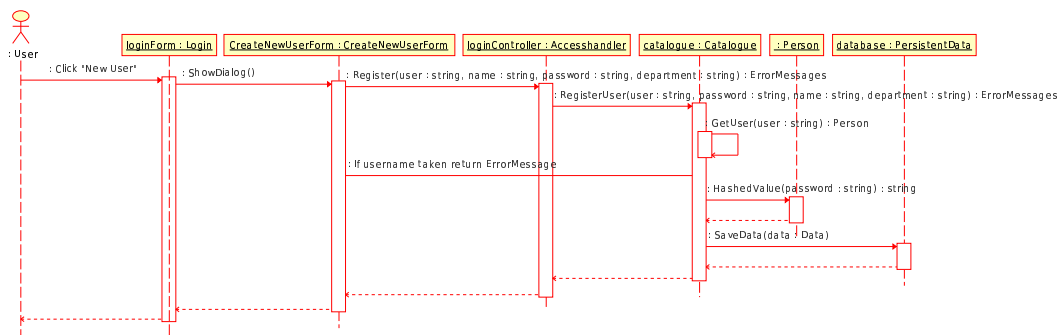


Figure 7.4.: Create new user sequence diagram

Figure 7.4 depicts the states in the process of creating a new user. When the user has entered the required fields in the instantiation of CreateNewUserForm the form invokes the loginController instantiation of AccessHandler with the register(arg) statement. The Accesshandler protects the system by only allowing the new user to make the call to the RegisterUser(arg) in the Catalogue instantiation. The Catalogue first checks if the username already has been taken by using the GetUser(arg) statement. If this is the case the Catalogue returns an ErrorMessage to the CreateNewUserForm. If the user does not already exist the Catalogue accesses the Person class' HashedValue(arg) operation in order to generate a hash of the user's password. The hash is used as encryption. Finally the Catalogue makes a call to SaveData(arg) which is a part of the PersistentData instantiation. This will save the new user information in the persistent data source and hereby finished the user creation.

7.4.2. Login sequence diagram

Figure 7.5 on the next page depicts the states in the login process. When the user has entered the required information in the instantiation of the Login (form) the form calls the Login(arg) statement in the loginController instantiation of the Accesshandler class. This forces the loginController to make a call for the GetUser(arg) statement in the Catalogue instantiation. The GetUser(arg) returns the result to the loginController. If the user does not exist the loginController returns an error to the Login (form) and terminates. If the user exists the loginController invokes the HashedValue(arg) function of the Person class in order to check if the entered password is correct. HashedValue(arg) returns the result of the password check to the loginController. If the password is invalid the loginController returns an error message to the Login (form) and terminates. If the password was correct the loginController returns a message to the Login (form) informing

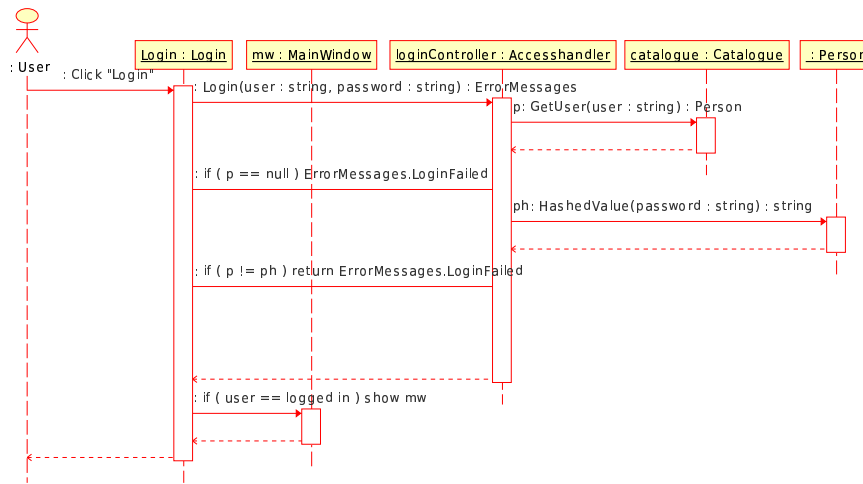


Figure 7.5.: Login sequence diagram

that the login was successful. Finally the Login (form) invokes an instantiation of the MainWindow.

7.4.3. Create project sequence diagram

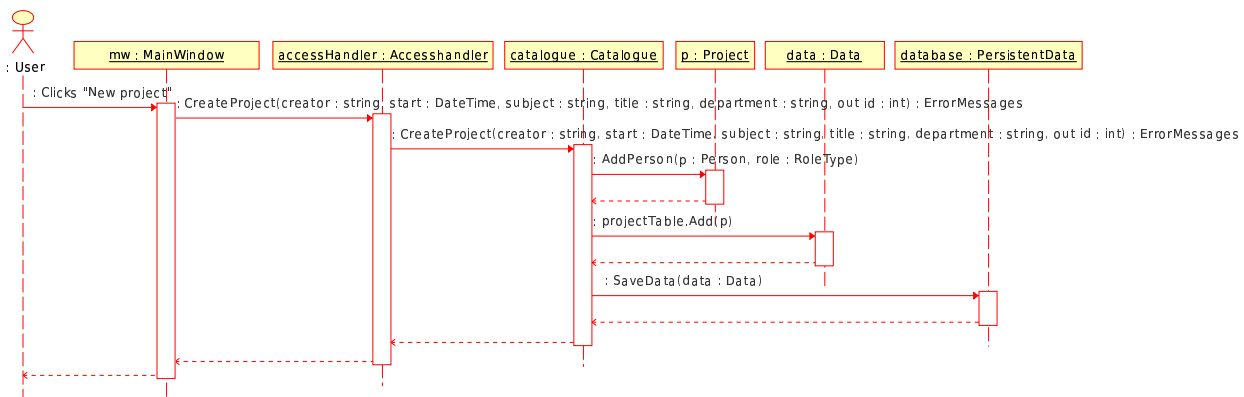


Figure 7.6.: Create new project sequence diagram

Figure 7.6 depicts the states in the create new project process. When the user has entered the required information about the project that he wants to add in the MainWindow instantiation, it will make a call to the CreateProject(arg) function in the AccessHandler instantiation. The AccessHandler will then redirect the request to the CreateProject(arg) function in the Catalogue instantiation. The Catalogue instantiation then makes a call to the AddPerson(arg) statement in the Project instantiation in order to connect the user with the project being created. When this is done the Catalogue instantiation invokes the projectTable.Add function in the Data instantiation. This function will add the project ind the list of projects in the system. When this is done the Catalogue makes a call for the SaveData(arg) statement in the database instantiation of the PersistenData class. This function will save the added data to the persistent data source. Finally the MainWindow is told that the project creation was successful.

Components

8.1. General Overview

The diagram 7.3 on page 46 depicts the general architecture of the system. A fully detailed class diagram of the system is shown in figure 8.1 on page 54. The following section details the system further.

8.2. Descriptions of Components

The component descriptions consists of a general description of the component structure and if relevant a description of the involved classes. This section details the classes' purposes and responsibilities.

8.2.1. PersistentData Component

The PersistentData component (in the center of figure 8.1 on page 54) handles saving the system data to a persistent source, for example on a harddisk. An abstract class named after the component PersistentData makes it possible for developers to use any type of persistent storage. As an example, an XMLHandler class was implemented, which provides load and save functions for the models data.

Data is encapsulated in a Data object which contains 3 dictionaries. One for people (Person class), one for Literature and one for Projects. A Data class is passed to the Save function and the Load function returns a new instance of the Data class.

If a database backend was used, it would have been advantageous for the PersistentData class to provide functions for saving single records of system objects, but such an implementation was not prioritized.

PersistentData Interface

PersistentData is an interface that provides the functions needed to save and load the system model. It allows for very low coupling between the actual implementation of the PersistentData and the Catalogue object.

XMLHandler Class

Loads and saves a data object to XML-files. The XMLHandler class inherits and implements load and save functions from the PersistentData interface.

BinaryData Class

Because the PersistentData interface allows for different implementations, changing the way the program saves data is very simple. Implementing a BinaryData class would allow for saving the program data into proprietary binary files, potentially faster than the XMLHandler approach.

ADO.NET Class

Another possibility is using the .NET Framework's data abstraction model ADO.NET to save and load data. Implementing an ADO.NET handler class allows for easy communication with a database or other ADO.NET compatible persistent data implementations.

8.2.2. Catalogue Component

The Catalogue component contains the actual model of the system. At runtime the Catalogue component loads all available data from the PersistentData component and keeps the data in a data structure consisting of dictionaries.

Catalogue Class

See the uppermost centre of figure 8.1 on page 54. This class is the core of the system. It has responsibility over the model. It contains all Projects, Literature, People, Reference, Review, Suggestion, Role and Comment objects. Catalogue has to ensure that the model is consistent, therefore all changes to the model are done through this class. This class also contains all tags and their relations to objects in the model. Catalogue also has the responsibility of saving the model to via the persistent data component. This class also contains synchronization functions for the multiuser environment. This class is a facade controller of the model and thus is very extensive, which unfortunately lowers its cohesion. It is also the Creator of all objects in the model as well as an expert on the information in the model.

The Catalogue object is singular, only one exists for the server. It contains a reference to an object that implements PersistentData and a reference to the Data that it has loaded from the persistent data.

Search Function This function takes one argument, a search string, and returns a list of literature and projects. It returns a list of ResultInfo that contains an ID from the UniquelyIdentifiable class, a piece of text that describes a piece of literature or project and a type to distinguish between project and literature. The returned list is sorted after how the match was done, if the search string matches the title of a result, then it goes in the front of the list, if it matches a tag then all related results are put in the center of the list, if the match was made on the synopsis, summary or title then the results are put in the end of the list.

```
1 public List<ResultInfo> Search(string query)
2 {
3     query = query.ToLower().Trim();
4     List<ResultInfo> infos = new List<ResultInfo>(),
5         titleMatches = new List<ResultInfo>();
6     Dictionary<int, ResultInfo> addedInfos = new Dictionary<int, ResultInfo>();
7     if (query.Length < 4)
8         return infos;
9
10    foreach (Project p in data.ProjectTable.Values)
11    {
12        if (p.Title.ToLower().Contains(query))
13        {
14            ResultInfo ri = new ResultInfo(p.ID.ToString(), p.Title, typeof(Project));
15            titleMatches.Add(ri);
16            addedInfos.Add(p.ID, ri);
17            continue;
18        }
19        if (p.Subject.ToLower().Contains(query) || p.Synopsis.ToLower().Contains(query))
20        {
21            ResultInfo ri = new ResultInfo(p.ID.ToString(), p.Title, typeof(Project));
22            infos.Add(ri);
23            addedInfos.Add(p.ID, ri);
24        }
25    }
26
27    foreach (Literature literature in data.LiteratureTable.Values)
28    {
```

```

29     if (literature.Title.ToLower().Contains(query))
30     {
31         ResultInfo ri = new ResultInfo(literature.ID.ToString(), literature.Title, typeof(Literature));
32         titleMatches.Add(ri);
33         addedInfos.Add(literature.ID, ri);
34         continue;
35     }
36     if (literature.Author.ToLower().Contains(query) || literature.Summary.ToLower().Contains(query))
37     {
38         ResultInfo ri = new ResultInfo(literature.ID.ToString(), literature.Title, typeof(Literature));
39         infos.Add(ri);
40         addedInfos.Add(literature.ID, ri);
41     }
42 }
43
44 DataRow[] row = data.Tags.Select("Tag LIKE '" + query.Replace("'", "'") + "'", "Tag");
45 foreach (DataRow r in row)
46 {
47     int id = (int)r["ID"];
48     if (addedInfos.ContainsKey(id))
49         continue;
50
51     if (data.ProjectTable.ContainsKey(id))
52     {
53         ResultInfo ri = new ResultInfo(id.ToString(), data.ProjectTable[id].Title, typeof(Project));
54         addedInfos.Add(id, ri);
55         infos.Insert(0, ri);
56     }
57     if (data.LiteratureTable.ContainsKey(id))
58     {
59         ResultInfo ri = new ResultInfo(id.ToString(), data.LiteratureTable[id].Title, typeof(Literature));
60         addedInfos.Add(id, ri);
61         infos.Insert(0, ri);
62     }
63 }
64 titleMatches.AddRange(infos);
65 return titleMatches;
66 }

```

8.2.3. AccessHandler Component

AccessHandler provides login function and ensures that a logged-in user only has the privileges he is entitled to. It is the link between the Server-API and the Catalogue. It mirrors most of the Catalogues public functions but adds additional user specific checks and extra functions. A new AccessHandler class instance is created for each logged-in user.

AccessHandler Interface

The AccessHandler interface describes all an AccessHandler's functions. An AccessHandler mirrors most of the functions on the Catalogue component but adds additional user authentication and permission checks. A Client creates a proxy object that implements the AccessHandler interface, when methods are called on the proxy object, the messages are marshalled and sent over the network by the Client-API component. The Server-API Component receives the marshalled method calls, demarshalls them and calls the users AccessHandler object on the server. The AccessHandler defines a Controller of the Catalogue.

8.2.4. Server-API Component

The Server-API is the server program's external interface. It provides access to an AccessHandler object. It is implemented as an abstract class called Server-API. By abstracting the communication between server and client, we open up for the possibility of implementing different solutions. Possible solutions for the Server-API are Web-Service, HTTP-Server and Remote Method Invocation(RMI), the last of which we intend to use. The Server-API lowers the coupling between the AccessHandler and the implementation of the server-side network code.

RMI-Server Class

The RMI-Server class inherits the Server-API abstract class. It is a Remote Method Invocation implementation, and is implemented in C# using .NET's Remoting library. The RMI-Server initializes the Remoting interface by opening a local port for connection and registering an RMI-ServerObject.

RMI-ServerObject Class

Is a remote object. This object inherits MarshalAsObjRef to enable its publication on an RMI-Server. When initializing the RMI-Server this object type is registered as a SingleCall remote object, meaning that a new instance is created for every user. The RMI-ServerObject Class contains only one function, a way to create a new AccessHandler for the user connecting and passing the client a reference to this new AccessHandler. This class is used in conjunction with the .NET Remoting library. An instance of this object is volatile in .NET remoting and thus allowing it to serve non-volatile AccessHandler remote objects, resulting in less disconnection errors and less error handling code.

8.2.5. Client-API Component

Client-API provides transparency for the client implementation. It creates instances of proxy objects that implements the AccessHandler interface. Using a proxy object allows for abstraction and very low coupling between the GUI and the Client-API.

Client-API Interface

Provides the necessary functions to require a proxy object of the server AccessHandler.

RMI-Client class

Connects to an RMI-Server. It has a function to get a reference to or proxy of the AccessHandler. It implements the Client-API interface.

8.2.6. The Graphical User Interface Component

The GUI component consists of a collection of different windows forms. They are implemented using C# and .NET's System.Windows.Forms library. They provide the interface for working with the AccessHandler. They implement all functions of the AccessHandler in different windows and dialogs as described in the following Interaction Spaces section.

8.3. Interaction spaces

This section describes those interaction-elements, that is not contained in another window.

8.3.1. Create user dialogue

In the window on figure 8.2 on page 55 the forms of interaction used are *conversation* and *browsing*. In this window the user can enter various data about themselves. In the listbox the user can choose which semester he is following. The button "Create User"

saves the current data, and the "Cancel" -button quits the current window and presents the Login window.

8.3.2. The Literature entry window

In the window on figure 8.3 on page 55 the only form of interaction used is *conversation*. In this window it is possible for the user to enter the relevant information about their literature. The buttons in the bottom of the window, allows the user to navigate to relevant parts of the program. When the user clicks "Review and comments" the user will be able to access these, and the user will be presented with a new window. The next button in the window is "Add this entry to project bibliography" and when this button is clicked, the current literature entry can be associated with a relevant project. When the "OK" -button is clicked the literature entry is just added to the system as a piece of literature, but not associated with any project. The "Cancel"-button closes the current window, and presents the user with the main-window.

8.3.3. Presentation model

The presentation model as seen on figure 8.4 on page 56 is an abstract version of our user interface. The purpose of the presentation model is to show how our interaction spaces are structured and linked together, which helps us design the GUI. Most of the interaction spaces in the presentation model have input, output and action elements. A input can for instance be text from a text box. Output is something displayed for the user, for instance a list of literature. Actions are elements the user can interact with, for instance if the user presses a button.

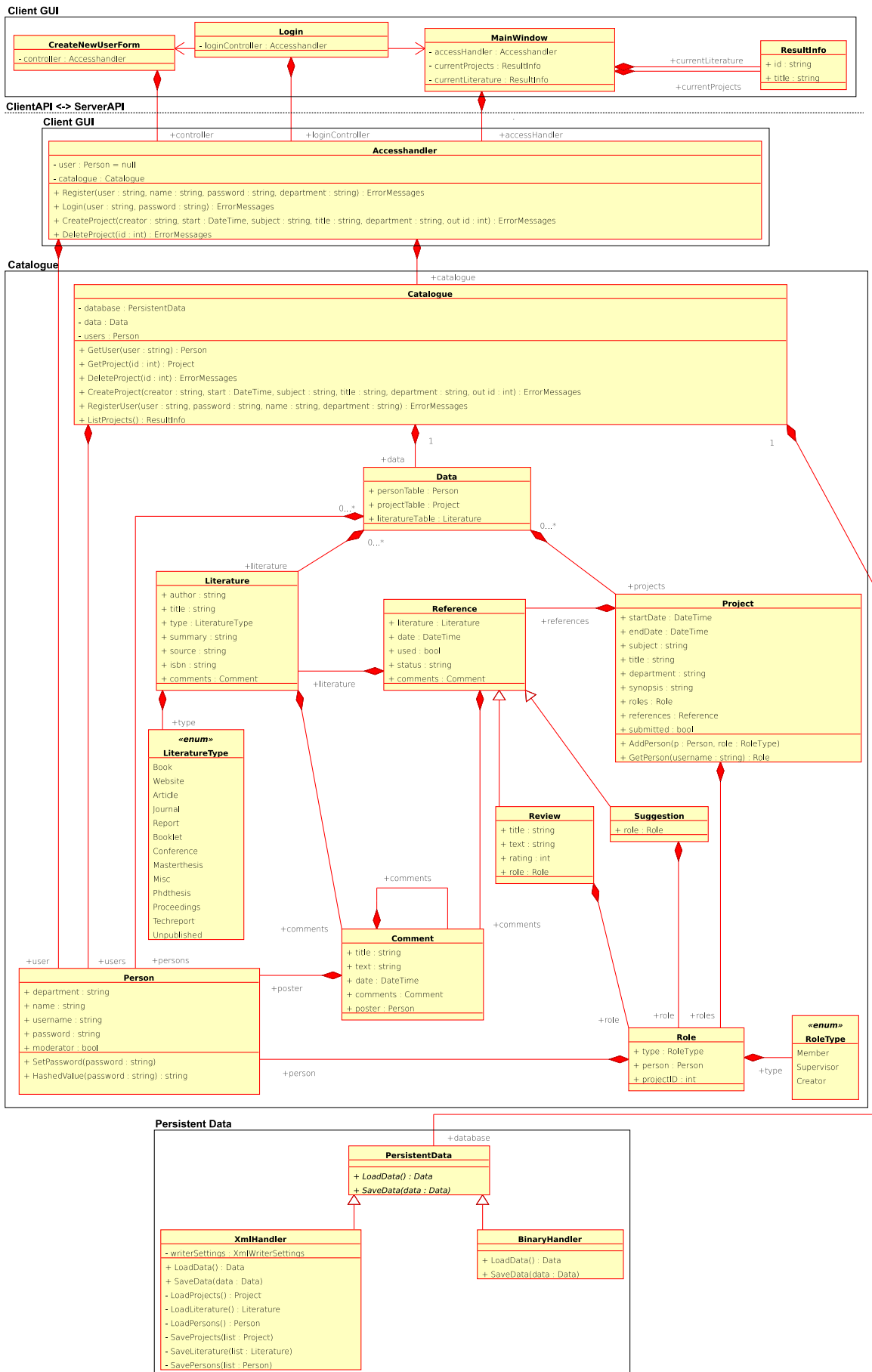
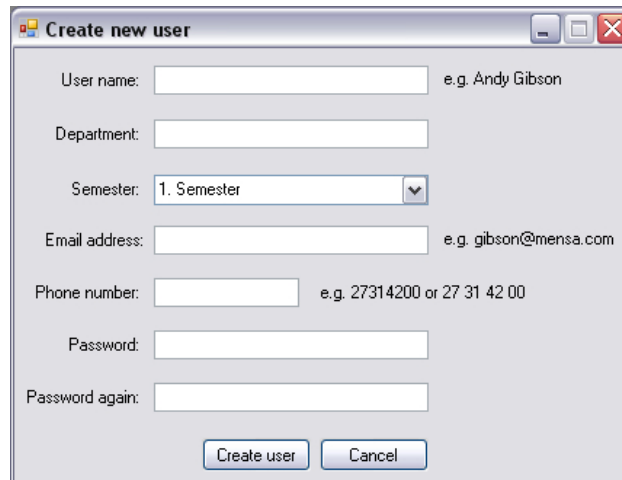


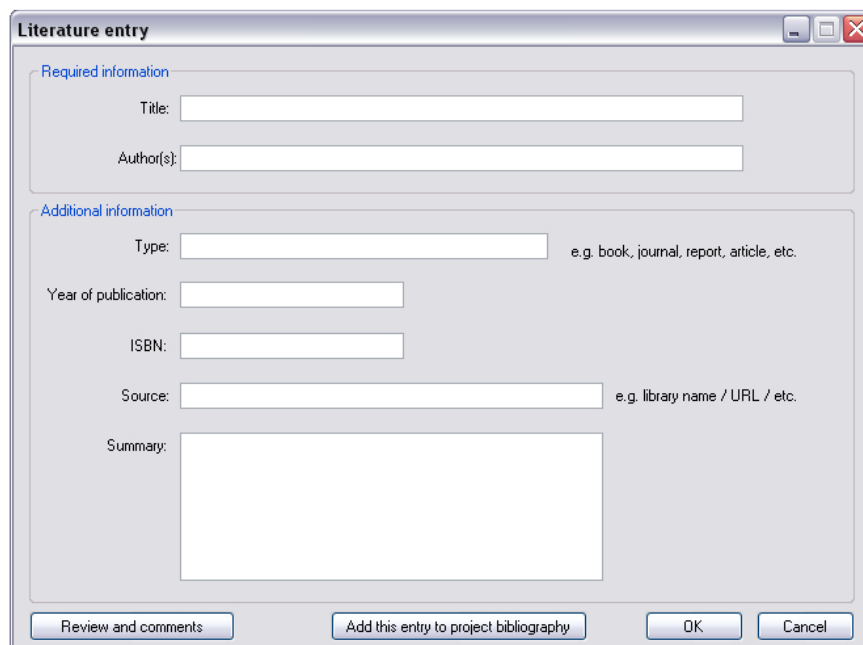
Figure 8.1.: Detailed class diagram for our system



The 'Create new user' dialog box contains the following fields and controls:

- User name: e.g. Andy Gibson
- Department:
- Semester: (dropdown menu)
- Email address: e.g. gibson@mensa.com
- Phone number: e.g. 27314200 or 27 31 42 00
- Password:
- Password again:
- Buttons:

Figure 8.2.: "Create User Window"



The 'Literature entry' dialog box is organized into two sections:

- Required information:**
 - Title:
 - Author(s):
- Additional information:**
 - Type: e.g. book, journal, report, article, etc.
 - Year of publication:
 - ISBN:
 - Source: e.g. library name / URL / etc.
 - Summary:
- Buttons:

Figure 8.3.: "Literature entry window"

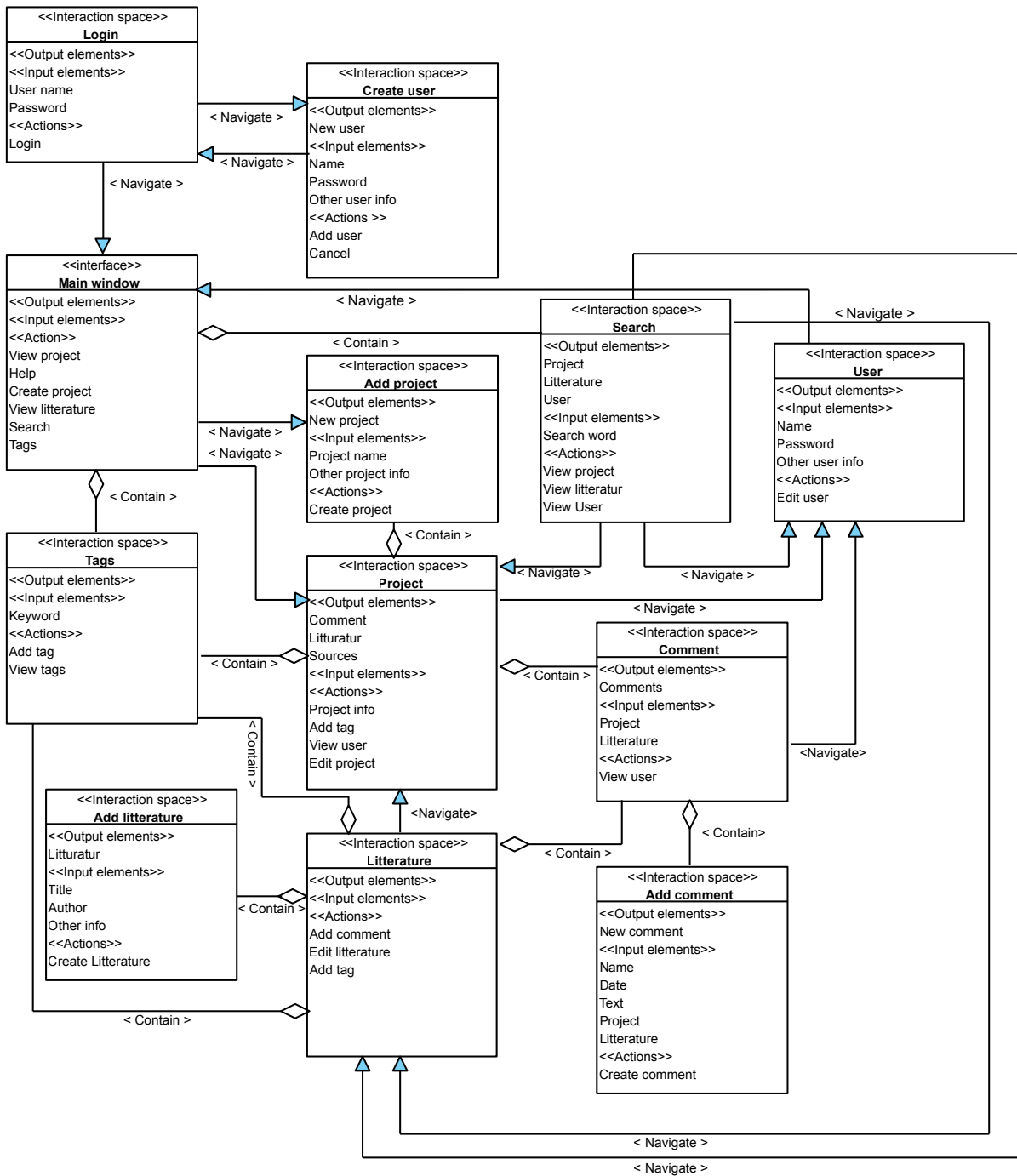


Figure 8.4.: Presentation model. Contain means that the interaction spaces is included in the interaction spaces it is contained in. Navigate means that it is possible to navigate from one of the interaction spaces to another.

Programming

9.1. Programming

This section shows an example of how to implement parts of the system. It serves to prove that implementing the system is feasible. We describe the use case Create new user and the interaction from the GUI down to the Catalogue as shown on figure 7.4 on page 47. We include some selected pieces of code. This example was chosen to highlight an update function, because a large portion of the program consists of these. This implementation uses an RMI communication model and XML for saving data.

9.1.1. Program execution

On server start, an instance of XmlHandler and Catalogue is made and the RMIServer is started.

Main function of the Server class

```
1 public static void Main(string args[])
2 {
3     // XmlHandler is our implementation of the Persistent Data component
4     PersistentData pd = new XmlHandler();
5
6     // Catalogue is instanciated
7     Catalogue catalogue = new Catalogue(pd);
8
9     // RMIServer is started, starts listening on port 8080.
10    RMIServer rmiServer = new RMIServer(catalogue);
11    rmiServer.Listen(8080);
12
13    // Wait for someone to close the server again by pressing a button
14    Console.Read();
15
16    // Gracefully disconnect clients
17    rmiServer.Close();
18 }
```

A client starts up by attempting contact to an RMI-Server. The address and port of this server can be loaded from a configuration file for this implementation, alternatively another dialog could be used to input server information. After connection is made the login window is shown.

Main function of the Client class

```
1 public static void Main(string args[])
2 {
3     try {
4         // Start Client
5         RMIConfiguration conf = LoadConfig();
6         RMIClient client = new RMIClient(conf);
7     } catch (IOException e) {
8         MessageBox.Show("`Could not load File");
9         return;
10    } catch (RemotingException e) {
11        MessageBox.Show("`Could not Connect");
12        return;
13    }
14
15    // Get accesshandler proxy
16    AccessHandler accessHandlerProxy = client.GetAccessHandler();
17
18    // Start Login Window
19    Login login = new Login(accessHandlerProxy);
20
21    Application.Run(login);
22 }
```

9.1.2. GUI: CreateNewUserForm

This form is the beginning, it is opened from the login window and is passed a reference to a proxy to an AccessHandler.

The CreateNewUserForm class

```

1 public partial class CreateNewUserForm : Form
2 {
3     // Reference to the accessHandler
4     private AccessHandler accessHandler;
5
6     // Constructor
7     public CreateNewUserForm(AccessHandler a)
8     {
9         // Set the AccessHandler
10        this.accessHandler = a;
11
12        // Initialize the form
13        InitializeComponent();
14    } // End constructor
15
16    // The function for a "CreateNewUser" button on the GUI
17    private void CreateUserButton(object sender, EventArgs e)
18    {
19        // Make some strings and retrieve data from textboxes on the GUI
20        string name = nameTextBox.Text;
21        string email = emailTextBox.Text;
22        string department = departmentTextBox.Text;
23        string username = usernameTextBox.Text;
24        string password = passwordTextBox.Text;
25        string confirm = confirmTextBox.Text; // Confirm password again
26
27        // This function checks if the confirmed password matches the password.
28        if (password != confirm)
29        {
30            MessageBox.Show("Passwords do not match!", "Try again!", MessageBoxButtons.OK, MessageBoxIcon.Error);
31            return;
32        }
33
34        ... // check more textboxes
35
36        // Finally call the "register" method on the accessHandler object with the input from the GUI.
37        // Return an error if the username is already taken.
38        ErrorMessage lm = accessHandler.Register(username, password, name, department);
39        if (lm != ErrorMessage.OK)
40        {
41            MessageBox.Show("Username already Exists!", "Try again!", MessageBoxButtons.OK, MessageBoxIcon.Error);
42            return;
43        }
44
45        // Tell GUI the user was created
46        MessageBox.Show("User was created!", "Success!", MessageBoxButtons.OK, MessageBoxIcon.Information);
47        this.Close();
48    } // End function
49
50 } // End class
51

```

9.1.3. The AccessHandler

When GetAccessHandler is called on an RMIClient instance it contacts RMIServer and gets a remote reference to a new AccessHandler object. RMIClient then creates the proxy object and returns that to the GUI. On the server, the RMIServer uses its reference to Catalogue to create the AccessHandler. The following is an excerpt from the AccessHandler class.

The AccessHandler class

```

1 // The different types of messages that can be returned as a result of a function call on the accessHandler.
2 public enum Messages
3 {
4     OK,
5     LoginFailed,
6     UserExists,
7     OperationNotAllowed,
8     Fail
9 }
10
11 public class AccessHandler
12 {
13     // When a user logs in a reference to his Person object is stored here.
14     // It is used to check whether or not he is allowed to perform the functions that are called.
15     private Person user = null;
16
17     // A private reference to the catalogue

```

```

18     private Catalogue catalogue;
19
20     // Constructor, called by the RMIServer object.
21     public AccessHandler(Catalogue catalogue)
22     {
23         this.catalogue = catalogue;
24     }
25
26     // The register function which was called with input from the GUI. It returns an message which
27     // could be "OK", "UserExists", etc.
28     // Because any client, logged in or not, can create a new user/person, no check is performed
29     public Messages Register(string user, string password, string name, string department)
30     {
31         return catalogue.RegisterUser(user, password, name, department);
32     }
33
34
35     // This is included to highlight the AccessHandlers main function.
36     // A check is made whether or not the current logged in user is allowed to delete a given project
37     public Messages DeleteProject(int id)
38     {
39         // if user variable is null, the user is not logged in.
40         if (user == null)
41             return Messages.OperationNotAllowed;
42
43         // Get the project from catalogue.
44         Project p = catalogue.GetProject(id);
45
46         // find this users role in the project.
47         Role r = p.GetPerson(user.UserName);
48
49         // A user is only allowed to delete the project if he is the creator of the project or a moderator.
50         if (user.Moderator || (r != null && r.Person == user && r.Type == RoleType.Creator))
51         {
52             return catalogue.DeleteProject(id);
53         }
54         else
55         {
56             return Messages.OperationNotAllowed;
57         }
58     }
59
60     //... continued

```

9.1.4. Catalogue

Catalogue receives calls from the accesshandler, updates and saves the model.

The Catalogue class

```

1 public class Catalogue
2 {
3     // Declare the PersistentData object used for storing data
4     private PersistentData database;
5
6     // Data Object contains the model
7     private Data data;
8
9     // Constructor
10    public Catalogue(PersistentData database)
11    {
12        this.database = database;
13        // Load data into memory
14        data = database.LoadData();
15        // Load users into the table
16        foreach (Person p in this.data.PersonTable.Values)
17        {
18            users.Add(p.UserName, p);
19        }
20    }
21
22    // The RegisterUser function
23    public Messages RegisterUser(string user, string password, string name, string department )
24    {
25        // Check if person exist
26
27        if (data.PersonTable.ContainsKey(user))
28            return Messages.UserExists;
29
30        // Create the new user. The "HashedValue" function hashes the users password into some tokens
31        // incomprehensible to humans.
32        p = new Person(user, Person.HashedValue(password), name, department);
33        data.PersonTable.Add(p.UserName, p);
34
35        // Add the new user to the table
36        users.Add(user, p);
37
38        // Save the data
39        database.SaveData(data);
40
41        return Messages.OK;
42    }

```

43
44 //... Continued ...

9.1.5. PersistentData: XMLHandler

This implementation is a test. It iterates all content of the Data object and writes it to a couple of XML- files. It makes extensive use of the XmlReader and XmlWriter classes in C#. The reasons for not using the XmlSerializer, is that it requires changing the model to accommodate the XmlSerializer by marking up all model classes with attributes that describe how they should be serialized in XML. To allow for lower cohesion a simpler approach is applied, exemplified in the following code:

The XmlHandler class

```

1 public class XmlHandler : PersistantData
2 {
3     XmlWriterSettings writerSettings;
4
5     // Constructor
6     public XmlHandler()
7     {
8         writerSettings = new XmlWriterSettings();
9         writerSettings.Indent = true;
10        writerSettings.IndentChars = ("  ");
11    }
12
13    // Loads all data
14    public override Data LoadData()
15    {
16        Dictionary<int, Project> projectTable = LoadProjects();
17        Dictionary<int, Literature> literatureTable = LoadLiterature();
18        Dictionary<string, Person> personTable = LoadPersons();
19        DataTable tags = new DataTable();
20        tags.Columns.Add("Tag", typeof(string));
21        tags.Columns.Add("ID", typeof(int));
22        tags.Columns.Add("UserName", typeof(string));
23        tags.PrimaryKey = new DataColumn[] { tags.Columns[0], tags.Columns[1], tags.Columns[2] };
24        Data data = new Data(projectTable, literatureTable, personTable, tags);
25        LoadTags(data);
26        LoadRoles(data);
27        return data;
28    }
29
30
31    // Saves a data object
32    public override void SaveData(Data data)
33    {
34        SaveProjects(data);
35        SaveLiterature(data);
36        SavePersons(data);
37        SaveTags(data);
38    }
39
40    // Load projects
41    private Dictionary<int, Project> LoadProjects()
42    {
43        Dictionary<int, Project> projectTable = new Dictionary<int, Project>();
44
45        string projectPath = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) + Path.DirectorySeparatorChar + "
46        Projects.XML";
47        if (!File.Exists(projectPath))
48            return projectTable;
49
50        using (System.Xml.XmlTextReader reader =
51            new System.Xml.XmlTextReader(projectPath))
52        {
53            Project project = null;
54            while (reader.Read())
55            {
56                reader.MoveToContent();
57
58                if (reader.NodeType == System.Xml.XmlNodeType.Element &&
59                    reader.Name == "Project")
60                {
61                    if (project != null)
62                        projectTable.Add(project.ID, project);
63                    project = new Project(0, "", "", "", "");
64                }
65
66                if (reader.NodeType == System.Xml.XmlNodeType.Element &&
67                    reader.Name == "ID")
68                {
69                    reader.Read(); reader.MoveToContent();
70                    project.ID = Convert.ToInt32(reader.Value);
71                }
72
73                //... More reading of variables ...
74            }
75
76            if (reader.NodeType == System.Xml.XmlNodeType.Element &&

```



```

74         reader.Name == "Department")
75     {
76         reader.Read(); reader.MoveToContent();
77         project.Department = reader.Value;
78     }
79
80     }
81     if (project != null)
82         projectTable.Add(project.ID, project);
83     }
84     return projectTable;
85 }
86
87 // Saving projects and roles.
88 private void SaveProjects(Data data)
89 {
90     List<Project> list = new List<Project>();
91     List<Role> roles = new List<Role>();
92     foreach (Project p in data.ProjectTable.Values) list.Add(p);
93
94     using (XmlWriter writer = XmlWriter.Create(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) + Path.
95         DirectorySeparatorChar + "Projects.XML", writerSettings))
96     {
97         writer.WriteStartElement("Projects");
98         for (int i = 0; i < list.Count; i++)
99         {
100             // Write XML data.
101             writer.WriteStartElement("Project");
102             writer.WriteElementString("ID", list[i].ID.ToString());
103             writer.WriteElementString("StartDate", list[i].StartDate.ToString());
104             writer.WriteElementString("Title", list[i].Title);
105             writer.WriteElementString("Department", list[i].Department);
106             writer.WriteElementString("Subject", list[i].Subject);
107             writer.WriteEndElement();
108
109             roles.AddRange(list[i].Roles);
110         }
111         writer.WriteEndElement();
112         writer.Flush();
113     }
114
115     using (XmlWriter writer = XmlWriter.Create(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) + Path.
116         DirectorySeparatorChar + "Roles.XML", writerSettings))
117     {
118         writer.WriteStartElement("Roles");
119         for (int i = 0; i < roles.Count; i++)
120         {
121             // Write XML data.
122             writer.WriteStartElement("Role");
123             writer.WriteElementString("Project", roles[i].Project.ID.ToString());
124             writer.WriteElementString("Person", roles[i].Person.UserName);
125             writer.WriteElementString("Type", roles[i].Type.ToString());
126             writer.WriteEndElement();
127         }
128         writer.WriteEndElement();
129         writer.Flush();
130     }
131 }
132 ... More Save and Load functions ...

```

Part III

Implementation

This section describes the implementation of the program. It also highlights the differences between the developed program and the original design. Some changes are done during implementation, these are often caused by the programming language in use or because of oversights in the original design document. Only the most interesting changes to the program structure and functionality are showcased.

Implementation

10.1. Environment

Programming Language The program is implemented using Microsoft .NET Framework and C#. The integrated developer environment(IDE) Microsoft Visual Studio is used. The most notably used class libraries from the .NET framework in the program are System.Windows.Forms and System.Runtime.Remoting. System.Windows.Forms and related libraries provide the user interface base code and System.Runtime.Remoting provides a readily available RMI implementation. Other classes handling the filesystem, data structures and collections are also used in the program. Alternatively an open-source implementation of the framework and included class libraries like Mono could have been used. This was rejected because the implementation of basic libraries like System.Windows.Forms is still lacking and because the program developers are trained in the use of Microsoft Visual Studio.

Assemblies The program is divided into several projects, each outputting an assembly. .NET creates 3 types of assemblies, a console application, a windows application and a class library. Console application open in the console where the program output is shown and input is made. Windows applications open as GUI windows using the System.Windows.Forms library. Both are EXE files. Class libraries are DLL files, these are shared assemblies that are often reused in different programs. Our program consists of three main assemblies and some extra class libraries.

Client The Client part of the program, implements the GUI component and Client-API. It is a Windows application.

Server The Server part of the program, implements PersistentData, Catalogue, AccessHandler and Server-API. It is a Console application.

Shared Contains the common elements Client and Server share. These are primarily the structures and messages needed in network communication.

Custom Controls Several custom controls were implemented for use in the Client assembly. The most interesting of these is the TagControls project, it contains custom controls for displaying, adding and removing tags. It also contains the TagCloud control. Custom controls are primarily interesting because they are usable within the Visual Studio Designer, easing development of the GUI.

10.2. Implementation vs. Design

Data Representation The program refrains from using the standard and usually preferred data storage method, Relational Databases. This decision was taken because implementing a program using Relational Databases would not strengthen the developers' knowledge of Object Oriented Programming. Instead an in-memory collection of

Dictionaries and lists are used. The Tags in the program however are implemented using an in-memory DataTable object which represents a relational table. This was done purely for performance and simplicity.

Data is saved using .NET's built in binary serialization feature, this choice was made when an attempt to implement it, as an alternative to saving in XML files, resulted in more stable and simpler code than that used for saving XML files. Though it is worth noting that saving in XML is still possible by changing very little source code as shown in the following code:

Snippet of Main() in Startup.cs in the server project

```
31 // Create the persistent data layer
32 // IPersistentData persistentdata = new BinaryData(Path.GetDirectoryName( Assembly.GetExecutingAssembly().Location)); //
   change this to:
33 IPersistentData persistentdata = new XmlData(); // this
```

Searching by Relevance To implement the feature of searching by relevance, that is finding other material in the system which is relevant to a single piece of literature, tags are implemented using the folksonomy theory, see appendix D on page 120. Tags, which are small keywords, can be assigned to any kind of material in the system by any user, if two pieces of material have the same tags they are relevant to each other. The more tags they have in common the more relevant they are to each other.

Client Error Handling The design document specified the usage of a common Error handling class. This feature was not implemented in the program, because all error messages are displayed using the MessageBox function of System.Windows.Forms. Another error handling class was however implemented to handle network connection errors. The ReconnectHandler implements the IAccessHandler interface and encapsulates all calls to the Client-API AccessHandler with a Try-Catch block. If the connection to the server fails, an exception is thrown and ReconnectHandler can attempt to reestablish the connection. For this implementation, if a single reconnection attempt fails the ReconnectHandler shows that reconnection failed and the re-throws the exception, crashing the Client. A class or form to handle multiple reconnection attempts is needed to improve this. The problem with reconnection errors, causing exceptions was discovered relatively late in the process and therefore this proof-of-concept solution was implemented.

Utils class A simple class for static functions used in both Server and Client was implemented in the Shared assembly. The HashedValue function for generating hashes of passwords was moved from Person to the Utils class. This was done because the Person class is located in the Server assembly.

Error Messages Calling methods on the AccessHandler can result in errors as a result of bad input, faulty network connection or missing privileges. These errors are handled in a couple of different ways. The Messages enumeration is the preferred method, it is used for all update commands. The Messages Enum is fairly short and do not return very descriptive error messages. This could be rectified by adding several error messages to the enumeration, or even implementing a function on the AccessHandler that gets last occurred error. Implementation of this was not prioritized, because if an error occurs when for example creating a project, being able to know that it either just failed, often because of faulty input, or the user was not allowed to create the project, are messages enough to describe most if not all functions on the AccessHandler.

Method calls that return actual data have different return values when an operation fails. This is most often implemented by returning NULL instead of the data requested. In these cases a GetLastError function on the AccessHandler might be preferable, but this also is of little importance until other clients, beyond the one implemented by us, are put to use. Implementing other clients might be difficult without very clear error messages on failed method calls on the AccessHandler.

Following is the Messages enumeration and an example of its usage in RMIAccessHandler:

Messages Example

```

1  /// <summary>
2  /// This enumeration contains the possible messages for the server-side methods to return.
3  /// </summary>
4  public enum Messages
5  {
6      OK,
7      LoginFailed,
8      UserExists,
9      OperationNotAllowed,
10     Fail,
11     UserHasAlreadyTagged
12 }
13
14 public partial class RMIAccessHandler
15 {
16     /// <summary>
17     /// First make a check if the password is correct. If yes call the corresponding method on the catalogue.
18     /// </summary>
19     /// <param name="user">The user's username</param>
20     /// <param name="password">The user's password</param>
21     /// <returns>Messages.OK on success and Messages.LoginFailed on fail.</returns>
22     public Messages Login(string user, string password)
23     {
24         Person p = catalogue.GetUser(user);
25         if (p == null)
26             return Messages.LoginFailed;
27
28         if (p.Password == password)
29         {
30             this.user = p;
31             return Messages.OK;
32         }
33
34         return Messages.LoginFailed;
35     }
36 }

```

Use-case Discrepancies Edit comment is currently not possible, neither is delete comment. Delete comment has ramifications because deleting a comment that has replies is not accepted behavior. It was decided during implementation that editing and deleting comments was not to be implemented. If an error was made in a comment, users will have to add a reply to the comment to point out the error. Remove User also was not implemented, this was done because deleting a user can result in cascading deletes. Each user has a Person class which is then used in a multitude of other classes to identify the creators of the object, in order to do permission testing. If a user is deleted, consequently all objects related to that person will have to be deleted. The solution to this problem is simply disabling a user instead, easily done by changing the users password to an unguessable value. As with remove user, this would be the responsibility of the moderator.

Usage of data structures in networking Fairly early on it was discovered that implementing an interchangeable network architecture caused object oriented programming to not be optimal for the task. Thus the solution implemented was creating data structures that could contain all data of an object on the server. These data structures are then passed when information is retrieved from or update on the server. An example of the use of these structures in creating, updating and getting info about a project follows:

Demonstration of usage of ProjectInfo

```

1  /// <summary>
2  /// Contains data about a Project. Used in network transmissions and representation on PersistentData.
3  /// </summary>
4  [Serializable]
5  public struct ProjectInfo : ISearchable
6  {
7      public int id;
8      public DateTime startDate;
9      public DateTime endDate;
10     public string subject;
11     public string title;
12     public string department;
13     public string synopsis;
14     public bool submitted;
15
16     public ProjectInfo(int id, DateTime start, DateTime endDate, string subject, string title, string department, string synopsis,
17         bool submitted)
18     {
19         this.id = id;
20         this.startDate = start;
21         this.endDate = endDate;
22         this.subject = subject;
23         this.title = title;
24         this.department = department;
25         this.synopsis = synopsis;
26         this.submitted = submitted;
27     }
28     [ISearchable Members]
29 }
30
31 /// <summary>
32 /// This interface makes it possible to create an accesshandler. it is used for creating e.g. the RMI-AccessHandler.
33 /// </summary>
34 public interface IAccessHandler
35 {
36     // Projects
37     ProjectInfo[] ListProjects();
38     ProjectInfo GetProjectInfo(int id);
39     Messages CreateProject(ProjectInfo pi, out int id);
40     Messages UpdateProject(ProjectInfo info);
41     Messages DeleteProject(int id);
42
43     ... CONTINUED ...
44 }
45
46 /// <summary>
47 /// Small demonstration class that works on projects
48 /// </summary>
49 public class ProjectDemo
50 {
51     private IAccessHandler accesshandler;
52     public ProjectDemo(IAccessHandler accesshandler)
53     {
54         this.accesshandler = accesshandler;
55     }
56
57     public void DemoCreateProject()
58     {
59         ProjectInfo pi = new ProjectInfo(-1, DateTime.Now, DateTime.Min, "Demonstrating InfoStructs", "Project Test",
60             "Computer Sciecnt", "", false);
61         int assignedID = -1;
62         Messages ml = accesshandler.CreateProject(pi, assignedID);
63         if(ml != Messages.OK)
64         {
65             MessageBox.Show("`Error`");
66         }
67     }
68
69     public void DemoGetProject(int id)
70     {
71         // Gets a projects data using its ID.
72         ProjectInfo pi = accessHandler.GetProjectInfo(id);
73     }
74 }

```

The info structs are also easily serializable and can therefore be used in saving the server's data as well as done in the BinaryData class.

Searchable types To allow for the results of a search to be displayed in one DataGridView, everything that can be found using search has a struct that implements the ISearchable. This was done so AccessHandler was able to return a list of different data structures and still be able to display simple common information about the items, such as title and text. The data structures in the list can be typecasted into the correct type. In C# the keyword "is" is used to check the type of the ISearchable object. The "is"

keyword returns true if an object can be casted to the specified type and false otherwise. This means that a `ProjectInfo` in an array of `ISearchables` can be used by doing the following:

Demonstration of casting `ISearchables`

```

1 public void SomeFunction(IAccessHandler accesshandler)
2 {
3     ISearchable[] results = accesshandler.ListItems(ResultType.Project | ResultType.Literature); // Lists all projects and
4     literature.
5     foreach(ISearchable i in results)
6     {
7         if(i is ProjectInfo) // if the result is a project do something you'd do with a project
8         {
9             ProjectInfo pi = (ProjectInfo)i;
10            MessageBox.Show(pi.department);
11        }
12        else if(i is LiteratureInfo) // else if its a literatureinfo, do something with that
13        {
14            LiteratureInfo li = (LiteratureInfo)i;
15            MessageBox.Show(li.author);
16        }
17    }
18 }

```

If a result of a search is somehow connected to a relevance value, like when searching by tags and saving how many times a tag has been used on a result, then the `SearchResult` class is used. `SearchResult` is merely a wrapper class that wraps an `Item` of the `ISearchable` interface, it also implements the `ISearchable` interface itself, returning the wrapped items information for every method in the `ISearchable` interface. `SearchResult` also has a float value that is initialized during a search procedure in that Catalogue component. When searching by free-text the relevance value is assigned using the term frequency-inverse document frequency algorithm [17] to an arbitrary value which when sorted on should reveal the most relevant result based on the input text.

10.3. The GUI

The GUI consists overall of three windows, the Main view, the Project view and the Literature View. Apart from the primary windows we have a number of dialog windows which help the user. We have the following secondary windows in our program:

Login.cs The login form is the first thing the user meet when the user starts the program. This form allows the user to log on to the server, change server settings (opens `LoginConfig.cs`) or create a new user (opens `CreateNewUser.cs`). If the user is not new to the system this form can remember login information for the user.

LoginConfig.cs This window allows the user to change the server's address and port. This form can only be accessed from the login form, since all other windows require the network components running.

CreateNewUser.cs This is a simple form that allows the user to create a new user. This form can only be accessed from the login form for the same reasons as above.

Reference.cs This form contains a list of projects or literature and a filtering function with which to search in the list. It allows the user to add a reference to a project or a piece of literature. This form is used in project view and in the literature view.

GetAUser.cs This form is only used by the project view. It enables adding a user or supervisor to a project. This form is almost identical to the reference form. The only difference to the above is that it contains a list of user in stead of literature.

CreateSuggestion.cs This form allows a supervisor to add a suggestion to a project. It is only accessible from the project view if the current user is a supervisor to the project. The form contains a list of all literature in the system and a reason text box.

About.cs Displays information about the program.

Several standard controls were applied and adjusted to suit our needs, but custom controls for the tag feature and for displaying comments had to be developed as well. Our custom controls are describe in section 10.3.4 on the facing page.

10.3.1. The Main Window:

The main window can be seen on figure 10.2 on the next page. In the design of the main window we have been inspired by Apple iTunes¹. iTunes is a music player and music shop. We think that iTunes have one of the best and most user friendly browsers we have seen. In our program the user needs to brows through a potentially very large amount information. So the user need to be able to filter the information in an easy and efficient manor. We think that the iTunes interface actives that. And that is why we have chosen to be inspired by iTunes. A picture of the iTunes interface can be seen in the appendix D.4 on page 123.

If the user clicks any of the item in the side bar the information relevant to that item is displayed in the data viewing area. For instance if the user clicks "My projects" then the projects that the user is a member of is displayed. If the user double clicks on any of the items in the data viewing area a new window, the project view or the literature view is displayed. Both this windows have tabs containing various information on that item. Above the data viewing area and side bar we have a number of buttons for different function for instance add new project or literature. And in the right side we have a search field. All this things are inspired by iTunes. Just out side the the search field we have a button which when pressed displays the tag cloud.

10.3.2. Project View:

The project view can be seen on figure 10.2 on the next page. It is divided into 3 tabs where the first tab consists of various textboxes containing general information about the project and a TagContainer. The second tab contains information about all the members and supervisors there have been added to the project. If the current user is the creator of the project the user can add new members or supervisors to the project from this tab. The third tab contain all the suggestion, references and reviews for the project. If the current user is a member of the project then the user can add new references or reviews to the project. It is also possible to export all the references and reviews to some type of file containing the sources for the project for instance a bibtex file.

10.3.3. Literature View:

The literature view can be seen on the figure 10.3 on page 70. Like the project view it is also divided into tabs. The first tab has textboxes containing general information about the literature and a TagContainer. The other tabs holds references and comments for the current literature entry. Apart from viewing and adding references in the references tab it is possible read and add reviews to different project. There can only be one review as of one project each review can be commented.

¹See: www.apple.com/itunes/

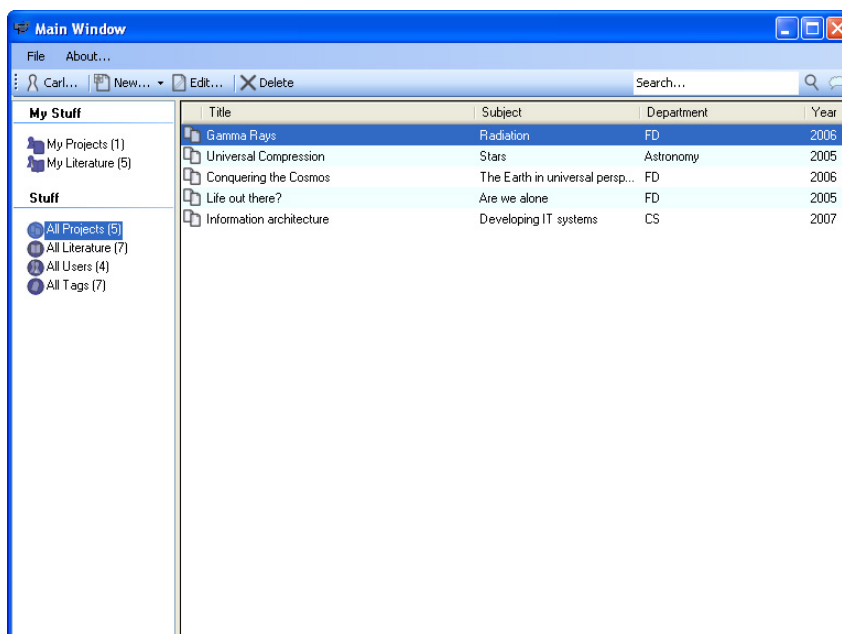


Figure 10.1.: The Main view window.

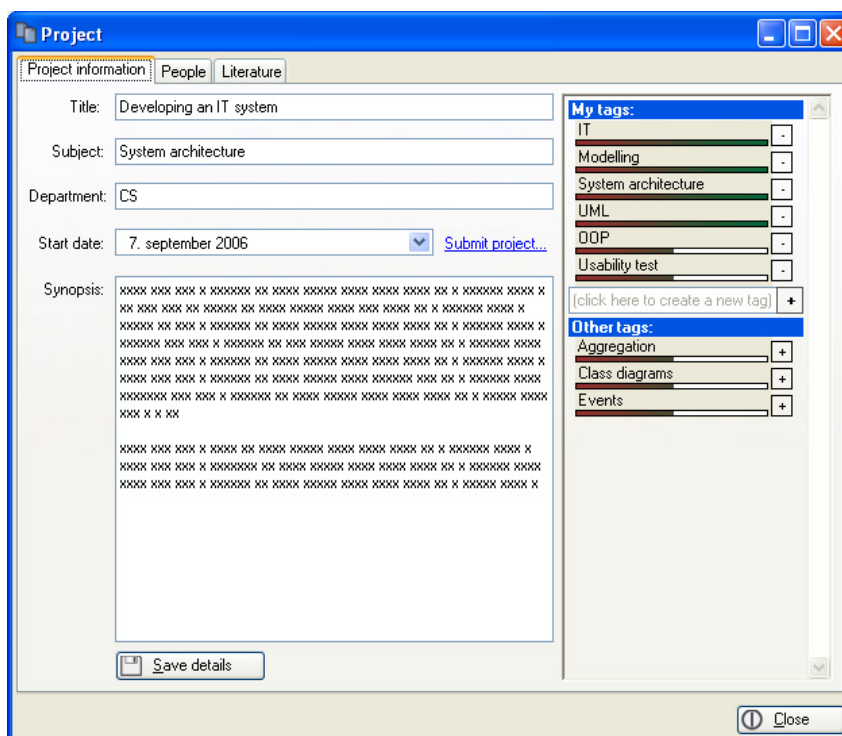


Figure 10.2.: The Project view window.

10.3.4. Custom Controls

In our implantation we have a number of custom controls. The reason we have custom controls is that we have some elements in our program that are not straight forward to implement with windows forms, contain several controls and functions and are used several times on the client. So to avoid writing or copying the same code several times we have created custom controls. A custom control is a form of encasement for functions

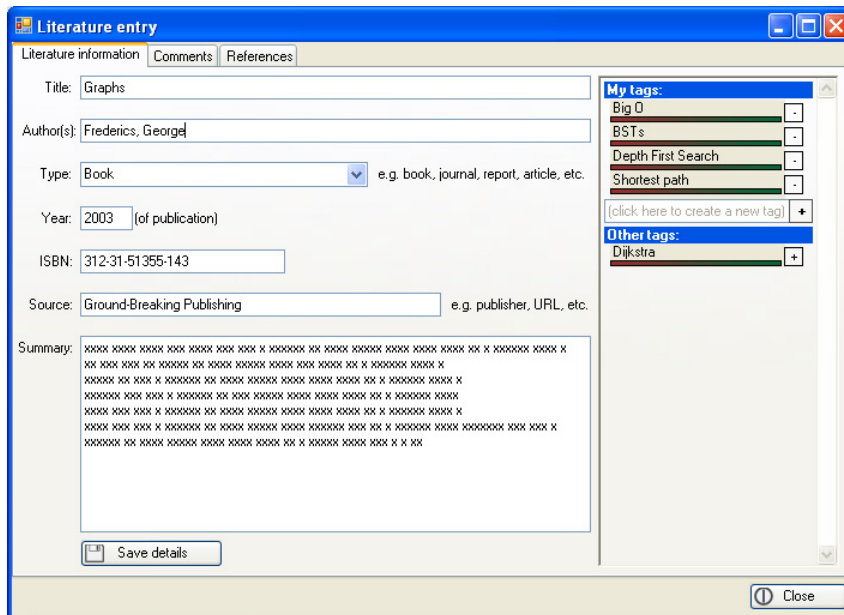


Figure 10.3.: The Literature entry view window.

and controls which has non or very low coupling to the rest of the program. Another advantage of custom controls is that they can be used like any other control in the designer in Microsoft Visual studio.

In our implementation we have the following custom controls:

- Tags
 - Tag container
 - Tag cloud
- Comments
- Doing stuff

We will in the following subsection describe our custom controls and their usage.

Tags

In our implantation we have two ways of displaying a list of tags: The tag cloud and the tag container. The input for both controls is a list with tags and user names where a tag can be repeated. The weight of a tag is calculated by finding the tag used most times. For instance if a project has two different tags used several times by different users: The first tag is used 4 times and the other is used 2 times. Then the most used tags weight is 100% and the other tag which is used half as mush and is therefor weighted 50%.

The tag container (seen in figure 10.4) is used to view, add and remove tags in a project or a piece of literature. The tag container is divided into two parts, in the first part all the current users own tags are displayed in the other all other tags are displayed. Each tag is displayed with a keyword, a status bar indicating the weight of the tag, and a remove button if the current user have added the tag to the current project and an add button if the user has not. The tag container also contain a text box for adding new tags. The tag container is used in the project view and in the literature view.

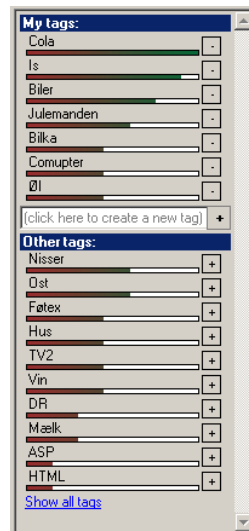


Figure 10.4.: The tag container

The tag container has two special events `TagAdded` and `TagRemoved`. These two events are fired when a user adds or removes a tag from the container.

The tag cloud (seen in figure 10.5) is another way for displaying tags. In the tag cloud each tag is sized after how often it is used. The user is able to sort the tags alphabetically or after weight, and additionally the user is able to instruct the control to show either all tags on the system, or only show their own tags.

The tag cloud has one special event called `TagSelected`. This event is fired when the user clicks one of the tags in the tag cloud.



Figure 10.5.: The tag cloud control

Comments

The comments control (seen in figure 10.6) enables the user to add, show and edit comments. The comment control contains a treeview where the user can navigate the comments and a viewing and editing area. The comment control is used in the project view and in the literature view. Unlike the tag controls the comment control has a very high coupling to the rest of the program since it needs the access handler to work.

Doing stuff

The Doing stuff control (seen in figure 10.7) is a very simple control which only displays the text "Please wait" and a small, animated icon. This control is used to tell the user that the program is working on something. We use this control for instance when the user

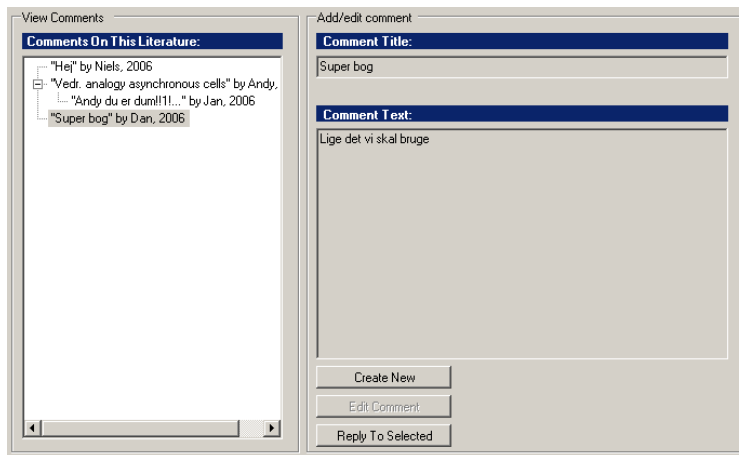


Figure 10.6.: The comment control

logs on the system. This might take a few seconds and instead of the program freezing we start the doing stuff and it tells the user that the program is currently working.



Figure 10.7.: Doing Stuff controle

Part IV

Test

Testing ensures the program works as expected. It can also contribute to a better user experience by highlighting previously unknown flaws in the design and implementation. The system is exposed to three different types of tests, blackbox, whitebox and usability. Black- and whitebox tests are unit test. They test specific parts of the program. Usability tests find previously unknown problems in the system by exposing it to end-users.

Unit tests

When performing blackbox testing the units are regarded as opaque entities where only interfaces are visible. That is, the internal implementation details are neglected. The tests consist of providing the different units' interfaces with inputs and then checking if they return the correct outputs. We test the Catalogue class and the PersistentData component using blackbox testing

Furthermore we have decided to do a whitebox test of a single complex algorithm, the search algorithm in catalogue. That means testing all possible paths inside the algorithm instead of just testing the interface.

For the unit testing we have deployed the third-party NUnit testing framework which is developed for testing code under the .NET framework. It involves including testing code in the target assembly and loading the assembly into NUnit once compiled. To be more specific when a class *B* is to be tested a corresponding test class is written and marked-up with C# attributes provided by NUnit. The public methods of the *B* class are represented in the test class as parameterless methods returning void. In each test method the corresponding method in *B* is fed with test data, and then an assertion is made of the nature of the output. When NUnit is run it finds all testing procedures in the target assembly and runs them one by one and reports any differences between asserted and actual output.

11.1. Blackbox testing

Persistency and efficiency is highly prioritized in the factor table 7.1 on page 42. Therefore we have decided to do blackbox testing on the persistentdata and model layer (catalogue) of our system.

In this section we will only provide a code snippet of the PersistentDataTest class. The snippet does only describe the general idea of the our unit test and therefore many details are not included.

The rest of the test classes and their documentation can be found in the program documentation on the CD-ROM.

The PersistentData layer is tested by using the BinaryData implementation. That means both the IPersistentData interface and an actual implementation is tested. The basic idea behind this test is to generate and save a lot of data and make sure it can be retrieved again. Additionally the retrieved data must be correct. Because the test is written for the IPersistentData interface, it will work on any class that implements the interface.

The following code snippets shows how the comment part of PersistentData is tested. Given the fact that a comment can be commented makes testing complex and therefore interesting.

First the PersistentData is initialized with a number of project and literature entries. Comments are added to the literature entries and all projects are assigned different types of references all of which are assigned comments. All comments also get comments until a particular depth is reached.

Snippet of TestDatabase() in PersistentDataTest.cs

```

1 public static void TestDatabase(int literatureCount, int projectCount, int commentLevel, int commentCount, [snip...] Some more
  parameters [snip])
2 {
3     // Initialize the data and a new dictionary to holds comments
4     DataContainer data = new DataContainer();
5     Dictionary<int, Comment> allComments = new Dictionary<int,Comment>();
6
7     // Create a number of literature and add comments to it
8     for (int i = 0; i < literatureCount; i++)
9     {
10         Literature p = new Literature([snip...] Some parameters [snip]);
11         data.LiteratureTable.Add(p.ID, p);
12         CreateShitloadOfComments(commentLevel, commentCount, data, p.Comments, allComments, CommentType.Literature);
13     }
14
15     // Create a number of projects with references and add comments to these references
16     for (int i = 0; i < projectCount; i++)
17     {
18         [snip...] Project creation and saving [snip]
19
20         Reference r = new Reference([snip...] Some parameters [snip]);
21         CreateShitloadOfComments(commentLevel, commentCount, data, r.Comments, allComments, CommentType.Reference);
22         p.References.Add(r);
23
24         Suggestion s = new Suggestion([snip...] Some parameters [snip]);
25         CreateShitloadOfComments(commentLevel, commentCount, data, s.Comments, allComments, CommentType.Reference);
26         p.References.Add(s);
27
28         Review rv = new Review([snip...] Some parameters [snip]);
29         CreateShitloadOfComments(commentLevel, commentCount, data, rv.Comments, allComments, CommentType.Reference);
30         p.References.Add(rv);
31     }
32
33     // Finally save everything to the PersistentData
34     data.Tags.AcceptChanges();
35     database.SaveData(data);
36
37     ...TO BE CONTINUED...
38 }

```

After all data is created it is saved, then a test is run to ensure the data was correctly saved. All data is loaded from the binary files and back into memory. Now every literature entry and project reference is iterated through and the TestComments() method is called.

Snippet of TestDatabase() in PersistentDataTest.cs

```

1
2     ...CONTINUED...
3
4     // Load the data
5     DataContainer newData = database.LoadData();
6
7     // Iterate through all references and test comments
8     foreach (Project p in data.ProjectTable.Values)
9     {
10         foreach (Reference r in p.References)
11         {
12             Reference r2 = null;
13             foreach (Reference r3 in p2.References)
14             {
15                 if (r3.Info.Equals(r.Info)) r2 = r3; break;
16             }
17             TestComments(r.Comments, r2.Comments);
18         }
19     }
20
21     // Iterate through all literature and test comments
22     foreach (Literature p in data.LiteratureTable.Values)
23     {
24         bool exists = newData.LiteratureTable.ContainsKey(p.ID);
25
26         if (exists)
27         {
28             Literature p2 = newData.LiteratureTable[p.ID];
29             Assert.AreEqual(p.Info, p2.Info, "Literature Attribute Test");
30             TestComments(p.Comments, p2.Comments);
31         }
32     }
33 }

```

This is the TestComments() method which takes two lists of comments and compares them with a number of assertions. It recursively compares comments to comments. The "Assert" methods are used by the NUnit unit-testing tool.

Snippet of TestComments() in PersistentDataTest.cs

```

1 protected static void TestComments(List<Comment> list, List<Comment> list_2)

```

```

2 {
3     foreach(Comment c in list)
4     {
5         // Make a new comment.
6         Comment c2 = null;
7
8         // Check if the ID of the two comments match. If yes set the new comment to point to the one in list_2.
9         // The new created comment is used for further comparisons.
10        foreach (Comment c3 in list_2)
11        {
12            if (c3.ID == c.ID)
13            {
14                c2 = c3;
15                break;
16            }
17        }
18
19        // Test if the comment is actually made
20        Assert.IsNotNull(c2, "Comment Exists Test");
21        // Compare the date
22        Assert.AreEqual(c.Date, c2.Date, "Comment Date Test");
23        // Compare the username
24        Assert.AreEqual(c.Poster.UserName, c2.Poster.UserName, "Comment Poster Test");
25        // Compare the text
26        Assert.AreEqual(c.Text, c2.Text, "Comment Text Test");
27        // Compare the title
28        Assert.AreEqual(c.Title, c2.Title, "Comment Title Test");
29
30        // Make a recursive call if there are more comments in the list
31        if (c2 != null) TestComments(c.Comments, c2.Comments);
32    }
33 }

```

This method creates a given number of nested comments by using recursion. It is called in the `TestDatabase()` method.

Snippet of `CreateShitloadOfComments()` in `PersistentDataTest.cs`

```

1 protected static void CreateShitloadOfComments(int level, int commentCount, DataContainer data, List<Comment> comments, Dictionary
<int, Comment> allComments, CommentType type)
2 {
3     if (level <= 0) return;
4
5     for (int i = 0; i < commentCount; i++)
6     {
7         Comment c = new Comment(snip...) Some parameters [snip]);
8         comments.Add(c);
9         allComments.Add(c.ID, c);
10
11        // Make a recursive call with (level - 1). The level parameter is given in the TestDatabase() method.
12        CreateShitloadOfComments(level - 1, commentCount, data, comments, allComments, CommentType.Comment);
13    }
14 }

```

11.2. Whitebox testing

We have decided to do a whitebox test of the search algorithm because it is one of the most complex algorithms in our system. We have decided to use the basic path test[11].

First the Cyclomatic Complexity[15] must be calculated. To do this a flow chart diagram of the algorithm is created. An easy way to do this is by abstracting the program flow of the algorithm into high level pseudo code as seen on the listing "Pseudo program flow of Search()". The capital letters A to J represent conditions and the lowercase letters o to z represent blocks of statements.

Pseudo program flow of Search()

```

1 Public SearchMethod(string query, ResultType type)
2 {
3     if (A) return; // Check if query.Length < 3. If true return an empty list.
4
5     do o; // Some initializations
6
7     if (B) // Check if (type == ResultType.Literature)
8     {
9         do p; // Search
10
11        if (C) do q; // If anything found add it to the list of results
12    }
13
14    if (D) // Check if (type == ResultType.Person)
15    {
16        do r; // Search
17    }

```



```

18     if (E) do s; // If anything found add it to the list of results
19   }
20
21   if (F) // Check if (type == ResultType.Project || type == ResultType.Review)
22   {
23     do v; // Traverse projects
24
25     if (G) // Check if (type == ResultType.Project)
26     {
27       do u; // More searching
28
29       if (H) do v; // If anything found add it to the list of results
30     }
31
32     if (I) // Check if (type == ResultType.Review)
33     {
34       do x; // More searching
35
36       if (J) do y; // If anything found add it to the list of results
37     }
38   }
39
40   do z; // Calculate the relevance of found results
41
42   return; // Return the results list
43 }

```

The Cyclomatic Complexity[15] can be calculated by either *number of "closed regions" + 1* in the flow chart diagram 11.1 on the following page or by the formula *number of control structures + 1*. In our case that the Search() algorithm has a cyclomatic complexity of 11. A basic set of independent paths in Search() therefore consists of 11 paths. The independent paths in a basic set can be combined to construct any possible path in the program flow. The idea behind basic path testing is testing all independent paths in a basic set and therefore in principle testing any possible path. The definition of an independent path is as follows: "An independent path is path that adds at least one new command or condition relative to already identified independent paths" [2].

From the flow chart diagram 11.1 on the next page a basic set of independent paths is constructed. It can be seen in the table 11.1. The Node Sequence corresponds to the path on the diagram 11.1 on the next page. The "Criteria" column provides a way to access that particular path in the algorithm. The reason for path 7 to be untestable is that condition "F" requires ResultType to be ResultType.Project or ResultType.Review and therefore must at least one of the conditions "G" or "I" be true. The actual whitebox testing can be found in the file "WhiteboxTesting.cs".

Path:	Node Sequence:	Criteria:
1	A	A query with length < 3
2	A,o,B,D,F,z	Searching for an unsupported ResultType
3	A,o,B,p,C,D,F,z	Searching for literature which is not found
4	A,o,B,p,C,q,F,z	Searching for literature which is found
5	A,o,B,D,r,E,F,z	Searching for a person which is not found
6	A,o,B,D,r,E,s,F,z	Searching for a person which is found
7	A,o,B,p,C,D,F,t,G,I,z	Cannot be tested
8	A,o,B,p,C,D,F,t,G,u,H,I,z	Searching for a project which is not found
9	A,o,B,p,C,D,F,t,G,u,H,v,I,z	Searching for a project which is found
10	A,o,B,p,C,D,F,t,G,I,x,J,z	Searching for a review which is not found
11	A,o,B,p,C,D,F,t,G,I,x,J,y,z	Searching for a review which is found

Table 11.1.: A basic set of independent paths of Search()

The figure 11.2 on page 79 shows a screenshot of NUnit running the test classes. The Catalogue and the binary implementation of PersistentData is tested by using black box tests. Furthermore the 10 testable independent paths of the Search() method is tested with white box testing. That is together 29 tests and 97 assertions of which some of

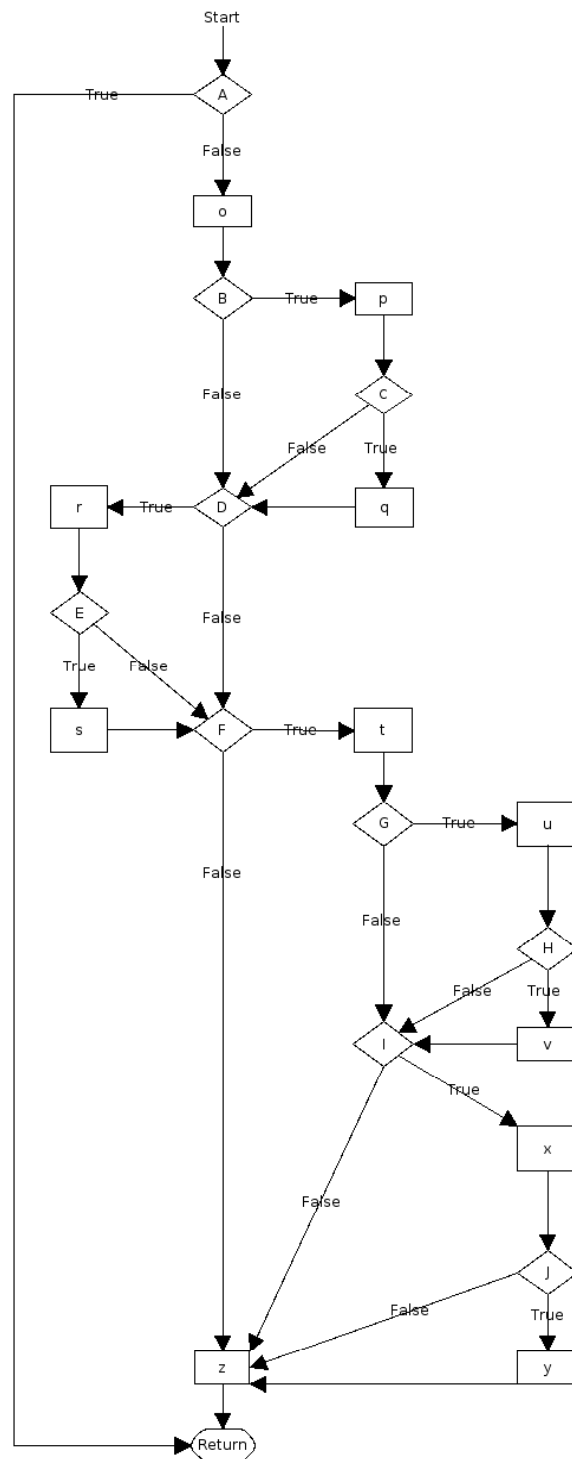


Figure 11.1.: Flow Chart Diagram over the Search() method.

these are recursive and iterative.

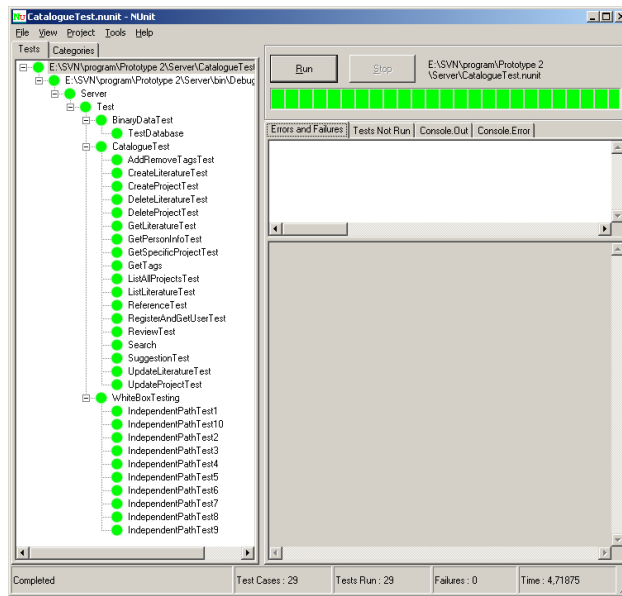


Figure 11.2.: Screenshot of NUnit running the test classes

Usability test

The following sections deal with usability testing of the developed article cataloguing system. First the test plan is introduced and afterwards the results are presented and discussed.

12.1. Plan for the usability test

We deploy the template for a usability test plan from [12] – though minor adjustments to suit our situation are applied.

Purpose

Since it is the first test of the article cataloguing system, the overall purpose of the test is to uncover whether representatives of our main target user group, that is, university students in project groups, are able to use the system to a satisfying extent. Furthermore since the development team has been working on the system for a long period of time, they simply view the system in a different way than outsiders do, and for that reason the developers might overlook issues that outsiders are able to spot.

Test objectives

- According to the factor-table (table 7.1 on page 42) "First time users should be able to gain an overall understanding of how the system works within a time frame of 10 minutes." We want to be able to determine if this goal is reached.
- We are interested in evaluating whether the participants are able to use the main functionality of the system with ease. The main functionality includes:
 - Create a new user and login
 - Create a new project with users affiliated
 - Create a new literature entry and associate it with a project
 - Retrieve specific information from a project or literature entry
 - Search by tags
 - Add a comment
- Through the test we want to acquire an answer to the question: Is the distribution of information and interplay between the GUI windows sensible to the participants?

User profile

The user is a typical university student in their twenties – male or female. Our persona character Brian Jensen (see section 3.1.2 on page 21) is an example of a user. What study programme the user is following is subordinate, since the system is intended to be used in various project groups regardless of their study content. We could have included

project group supervisors as participants, but we focus on our main user group, and that is why we delimit our group of participants to comprise university students in project groups.

Test design – including test environment and equipment

The tests are assessment tests to determine to which extent the system fulfils the demands we have set up. The tests are carried out in the usability lab at the Aalborg University as "think aloud" tests [12]. The latter to ensure that the participants' thoughts with respect to problem solving strategies, considerations, uncertainties and expectations are caught on video for subsequent analysis.

The usability lab is a three room lab: One test room, one tech-room with the video recorders and monitors and one room for observers.

Three or four test participants are engaged in the test. The participants participate in the test individually. All participants are presented for the same exercises – one at a time. Each test is expected to last around half an hour including debriefing.

To capture possible errors or inappropriatenesses in the exercise design, a dummy test is performed prior to the real tests. In the dummy test a member of the development team acts as test participant.

Test monitor role and additional required staff

The same person acts as test monitor through all the tests to minimize the risk of different persons influencing the test results in different ways. After the arrival of the participants, a written statement is read aloud to each participant by the test monitor. See the statement (in Danish) in Appendix A.2 on page 107.

The main role of the test monitor during the test is to remind the participant to vocalize his thoughts. The test monitor is not allowed to provide specific solutions to occurring problems, only to assist with questions of the type that forces the participant to generate his own answers.

After the last exercise the test monitor debriefs the participants.

Apart from the test monitor, who is seated next to the participant in the test room, the following test staff is engaged:

- A technician to operate the video equipment.
- One or more observers situated in the observer room to take notes.

In some situations also a data logger situated next to the technician is required.

Task list

During the test the participant will be introduced to some exercises that he should try to solve. The exercises must be written such that they describe realistic situations that the participant might end up in if he were using the system in real life [12, p. 179].

In Appendix A.3 on page 108 we have listed tasks that we want the participants to solve. This information is presented in task tables followed by templates to the actual exercises that are given to the participants. The essential aspects of the tasks are:

1. Creation of a new user
2. Creation of a new project

3. Creation of a new literature entry
4. Localization of a literature entry
5. Addition of a comment

The Exercises

The exercises handed to the participants are given below but were presented in Danish to the participants as shown in Appendix A.4 on page 111. An expected solution to each exercise is developed and included in Appendix A.5 on page 112 in order to ensure the quality of the exercises.

Exercise 1 – Create a new user

Imagine that you as a student want to use the literature program. Create your own user account (using your first name as user name and a password by choice) and log into the program.

Exercise 2 – Create a new project

The underlined text below was excluded from the test because the program was not able to handle tags correctly at the time where the test took place.

Your group has asked you to create your project with the title "IT i Aalborg" in the program. The following tags (keywords) must be attached to the project: Aalborg, IT, computer, usability and KMD. Furthermore you must add your two group mates "john82" and "gitte_pj" to the project.

Exercise 3 – Create a new literature entry

You have found a book at the library which is relevant to your project. Add this book in the system using the following:

Title: Netværksbogen
Authors: Mølgaard og Nielsen
Year of publication: 2002
Publisher: IDG

After this you must ensure that the book has been attached to your project "IT i Aalborg".

Exercise 4 – Find literature

Your supervisor has suggested a book for your project but only remembers that it already exists in the program and that it deals with "usability testing". What is the full title of this book?

Exercise 5 – Add comment

You want to add a short comment to the book "Handbook Of Usability Testing" by Jeffrey Rubin, that you had to find in the previous exercise. The wording must be: "The book is good because it treats the aspects of a test intensively".

Debriefing

After the completion of the last exercise, a debriefing of each participant took place. This has an important role, since it serves as a valuable way to obtain information that might not have been obtained by just looking at the participants' solutions to the exercises. These questions were as the exercises presented in Danish (see appendix A.6 on page 113). The questions are listed below in English:

- What do you best remember from the test?
- What do you think about the user interface?
- What do you think about the organization of the information at the different windows?
- How was the navigation in the program?
- Was anything bothering you?
- Do you feel that something is missing in the program?
- What did you find good about the program?
- Would you find this program useful in your own course?

Evaluation measures and data to be collected

We apply the following evaluation measures:

Time: The time it took the participant to accomplish an exercise.

Categorization of problems: How critical was the problem the participant encountered, and what caused the problem.

Obviously the rating of problems are not an unbiased, but is down to personal judgment. To reach a more concise rating we compare the test data to determine if a two or more participants had the same problem.

The data collection is done with picture-in-picture-video¹ and written notes from one or more observers.

Report contents and presentation

The test results are reported in section 12.2 and the group has the test videos on DVD.

12.2. Analysis of results

In the following sections the data from the usability test will be treated. First a description of the actual test process will be presented followed by an analysis of the time participants used for each exercise. After this a thorough evaluation of the issues encountered is performed.

¹A large picture of the computer screen is merged with a small picture of the participant (and test monitor).

The course of the test

Due to cancellations from two people, we ended up with three participants, and the tests were conducted in one day. It was three male participants ranging from 21 to 24 years of age. There were two university students and one former. They all had a rather high level of computer experience. The test crew consisted of a test monitor, a technician and an observer.

Unfortunately the system was not completely finished when the test was scheduled, so we had to take the following precautionary measures: Whenever the participants entered an unfinished part of the system, the test monitor had to interrupt and tell the participant the state of affairs. Furthermore a few assumptions (for instance that the system had responded to the users' actions even though no feedback was provided – due to unfinished functionality) had to be made in order for the participants to complete the exercises 2, 3 and 5.

Duration of the exercises

	Participant 1			Participant 2			Participant 3			Benchmark
	Start	End	Time	Start	End	Time	Start	End	Time	
Exercise 1	02:38	03:50	01:12	01:40	03:31	01:51	03:23	04:19	00:56	02:00
Exercise 2	04:49	10:16	05:27	04:40	09:47	05:07	04:40	05:47	01:07	05:00
Exercise 3	10:45	13:05	02:20	10:15	18:22	06:23*	06:16	07:40	01:24	03:00
Exercise 4	14:15	15:23	01:08	18:43	19:25	00:42	08:52	10:20	01:28	02:00
Exercise 5	17:43	20:18	02:35	19:50	23:15	03:25	10:44	11:47	01:03	02:00

Table 12.1.: A table of exercise duration for each participant in the usability test. Glossary: **Start:** The video clock (minutes:seconds) as the participant had finished reading the exercise aloud. **End:** The video clock as the participant explicitly stated that he felt he had finished the exercise. **Time:** The duration of the completion of the exercise i.e. End minus Start. Benchmark: The time constraints specified in appendix A.3 on page 108. *A system breakdown occurred and required a restart. The duration of this incident is subtracted.

Table 12.1 shows how long time it took the participants to solve each exercise. A way to treat the time data depicted in table 12.1 would have been to calculate the mean time [12, p. 260] and other statistics. However such calculations do not make much sense because the participants are asked to vocalize their thoughts. Instead the time data serve as indicators of whether the benchmark requirements specified in the tables: A.1 on page 108, A.2, A.3, A.4, A.5, A.6 and A.7 have been complied.

As seen in Table 12.1 participant 1 exceeds the benchmarks in exercise 2 and 5, participant 2 in exercise 2, 3 and 5 while participant 3 does not exceed any benchmark.

The time data are not strictly comparable because of the participants' different backgrounds. In particular participant 3's fast performance stand out. This is due to the fact that he – contrary to participant 1 and 2 – did not have any prior experience with usability testing and for that reason did not spend as much time explaining strategies and considerations as the others. The advantage of him lacking this experience is that a more realistic portrait of the use of the system is obtained.

When participant 2 attempted to complete exercise 3 he first settled on one strategy and then switched to another later on. This is the reason for the exceptionally long time

he took on this exercise.

In Table 5.1 on page 39 the quality goal "Efficient" is rated "very important". The fact that participant 3 spends maximum 1 minute 30 seconds completing a task suggests that this goal has been reached. A more participant extensive test with no think aloud demands would be a way to try to verify this suggestion. Such a test however is beyond the scope of this project.

Evaluation of issues

In the following sections the encountered issues are shown with a suggestion on how to handle them. All issues discovered are categorized by which views they appeared in. Furthermore we have prioritized the problems according to how frequently they occurred or how urgent we judged them to be. Table 12.6 on page 90 lists the discovered issues along a rating of the severity of each issue. The rating scale used is described by Rolf Molich[10], and has the following steps: cosmetic, serious and critical. The delay², frustration and difficulty in solving the task caused increases from cosmetic through serious to critical. Furthermore does a catastrophe occur if two or more participants encounter the same critical issue independent of each other.

General issues

Issue number :	Issue / Cause	The following experienced the issue:			Possible solution
		P1	P2	P3	
1	It is confusing that there are the subglobal buttons "Save" and "Save and Close" on both the Project and Literature views.	X	X	X	The "Save" buttons could be moved to the context where they are used, which leaves only a subglobal "Close" button to check if the user has unsaved data and asks the user if that should be saved.
2	Tooltips disappear too fast – especially in the search field due to a long text.	X			Either the text should be shortened or the time delay increased.
3	Empty views or lists are confusing.		X		Place a message in these views which explains the reasons for the emptiness.

Table 12.2.: Table showing general issues and possible solutions

Table 12.2 lists the general usability issues found during the test. The "Save and close" button issue 1 is illustrated in Figure 12.1 on the following page and the tool tip issue 2 is illustrated in Figure 12.2 on the next page. The reason that empty views or lists are confusing is that the user might think that something has went wrong and want to know how to make data available in those views.

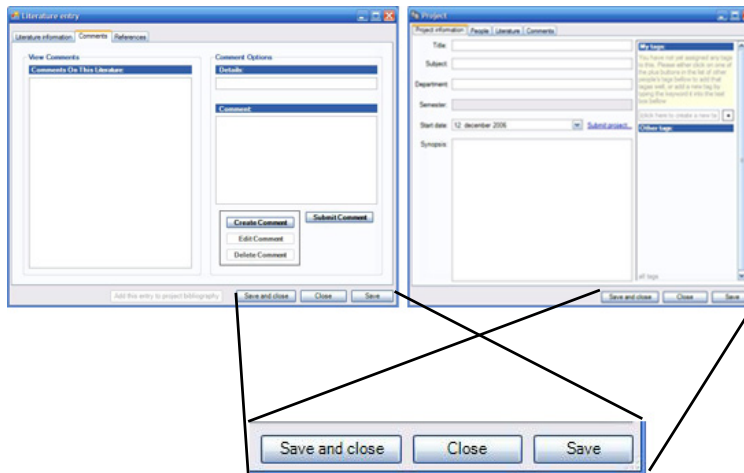


Figure 12.1.: This screenshot illustrates the general "Save and close" button issue 1.

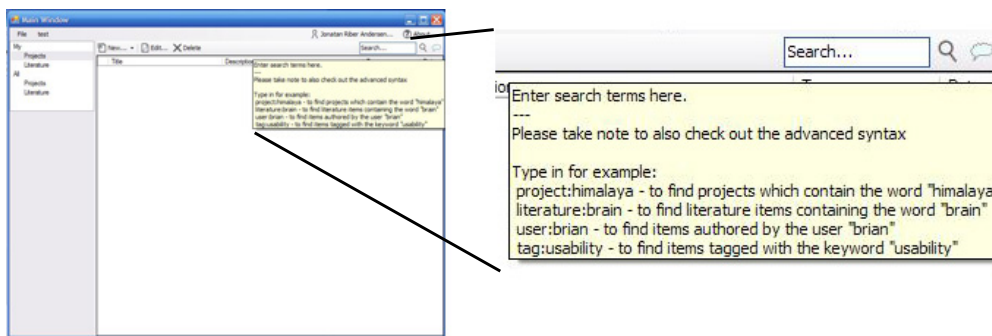


Figure 12.2.: This screenshot illustrates the general tool tip issue 2.

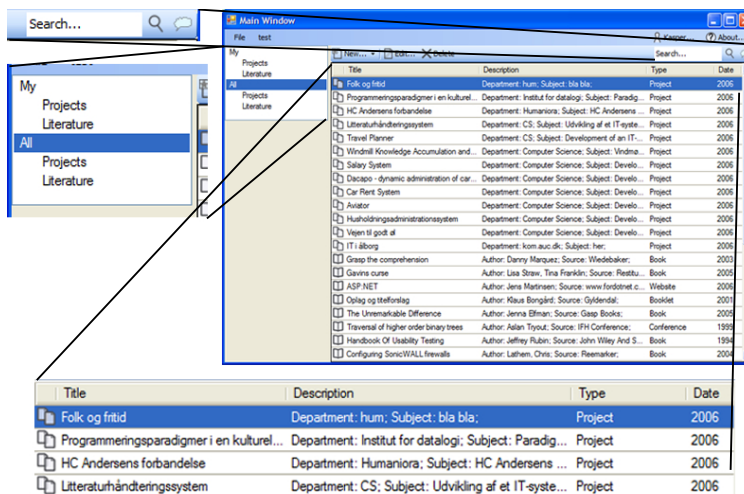


Figure 12.3.: This is a screenshot from the Main view illustrating the "My" / "All" issue 4, the "Search" field visibility issue 5 and the "Description" issue 7.

Issue number :	Issue / Cause	The following experienced the issue:			Possible solution
		P1	P2	P3	
4	The participants were uncertain about the meaning of "My" and "All".	X	X	X	Could be explained in a help menu and/or in the introduction to the system.
5	The participants do not find the search functionalities (Search field and Tag Cloud) in the upper right corner of the main view because they do not attract much attention.		X	X	They can be repositioned or be made more visible – or just left unchanged, since the position is rather standard – as seen on several web pages for instance.
6	When clicking "All" or "My" projects and literature are mixed in the Main view list.		X		Rearrangement of the items in the Main view sidebar such that it is divided into "Projects" with the subcategories "My projects" and "All projects" and "Literature" with the subcategories "My literature" and "All literature".
7	A participant was confused about the "Description" label in the Main window – he thought it would contain a summary of a literature entry.			X	"Description" could be removed. Instead project and literature info could be displayed in the sidebar.

Table 12.3.: Table showing issues in the Main view and possible solutions

Issues related to the Main view

The issues found in Main view are listed in Figure 12.3. The "My" / "All" issue 4, the "Search" field visibility issue 5 and the "Description" issue 7 are all illustrated in Figure 12.3 on the preceding page.

Issues related to the Project view

The issues in the Project view are listed in Figure 12.4 on the following page. The "Add reference" issue explained in issue 11 is illustrated in Figure 12.4 on the next page. The other issues related to the project view are not depicted because they are suggestions from the test participants and not easy to depict.

Issues related to the Literature view

The issues encountered in the Literature view are listed in Figure 12.5 on page 89. The "Details" field issue 12 and the "Create comment" / "Submit comment" issue 14 are shown in Figure 12.5 on page 89. The "Year of publication" issue 13 is illustrated in Figure 12.6 on page 89.

²By delay we mean the extra time needed for the participant to overcome the issue

		The following experienced the issue:			
Issue number :	Issue / Cause	P1	P2	P3	Possible solution
8	Participants were unable to add group members to a project if it was not saved.	X	X	X	Auto-save the project info when another tab is selected.
9	The participant expects that he can create a new literature entry from within the "Literature" tab in the "Project" view.		X		Add a "Create new literature entry" button in the "Add Reference" dialog.
10	A participant expected that he could add a comment to a literature entry in the list of literature associated with the project.		X		Open the "Literature" view when an entry is double clicked.
11	In the tab "Literature" in the "Project" view the participants get confused about the options under the "Add Reference" button.	X		X	The "Add Reference" button should not have dropdown options. These options should instead be represented in the "Add Reference" dialog with suitable labels.

Table 12.4.: Table showing issues in Project view and possible solutions

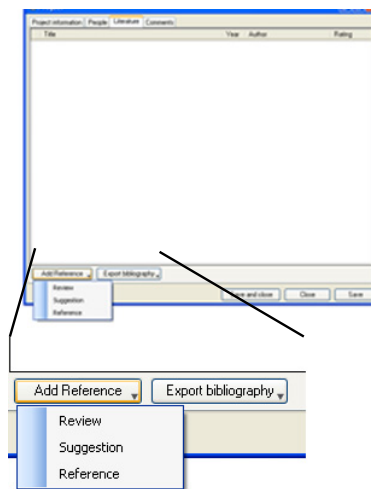


Figure 12.4.: This screenshot depicts the Project view "Add Reference" issue 11.

Issues we have decided not to solve

We have chosen not to implement the low priority issues stated in Figure 12.7 on page 90. In general these issues are suggestions for improvements that we did not want to implement either because we did not find it within the scope of the system or because we did not find the suggestion appropriate.

12.2.1. Discussion

Despite the fact that the system was not completely finished at the time of testing we consider the test a success. A success in the sense that we became aware of some

		The following experienced the issue:			
Issue number	Issue / Cause	P1	P2	P3	Possible solution
12	In the "Comments" tab the participants did not understand the meaning of the "Details" field.	X	X	X	Rename to "Comment title".
13	When clicking the "Year of publication" field in the "Create a new literature entry" dialog with the mouse the cursor position hinders typing.	X	X		Correct the error.
14	In the "Comments" tab the "Create comment" and "Submit comment" buttons might cause confusion.	X			When clicking the "Create comment" button it could be changed to a "Submit comment" button, instead of having 2 buttons.

Table 12.5.: Table showing issues in Literature view and possible solutions

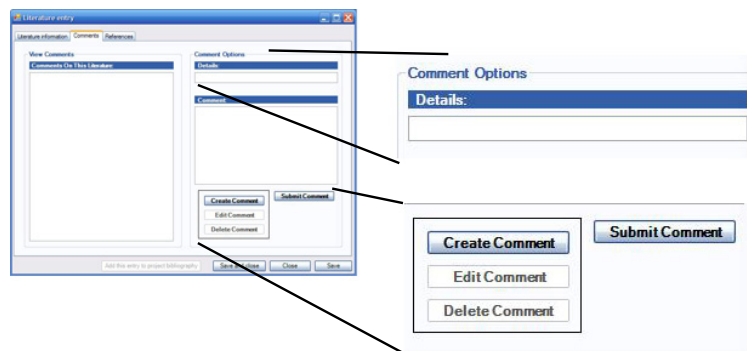


Figure 12.5.: Screenshot of the comments tab in the literature view depicting the "Details" field issue 12 and the "Create comment" / "Submit comment" issue 14.

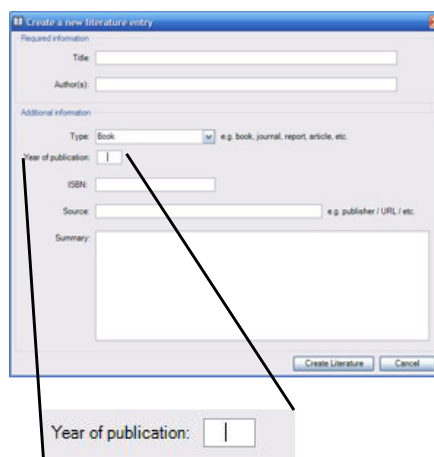


Figure 12.6.: Screenshot of the "Create new literature entry" view depicting the "Year of publication" issue 13.

Issue #:	Problem Severity:
1	Cosmetic
2	Cosmetic
3	Cosmetic
4	Cosmetic
5	Serious
6	Cosmetic
7	Cosmetic
8	Serious
9	Serious
10	Cosmetic
11	Serious
12	Serious
13	Cosmetic
14	Cosmetic

Table 12.6.: Classification of usability issues

Issue number :	Issue / Cause	The following experienced the issue:			Reason for not solving this problem:
		P1	P2	P3	
15	A participant requested further division of "All projects" into more subcategories such as one containing groups.			X	A very interesting idea that we have chosen not to develop because it is a new functionality.
16	A participant requested the option to see a summary for a literature entry in the Main view.			X	We do not implement this because we do not want to present too much information in the Main view.
17	A participant requested a reference showing where to get hold of literature (libraries, databases etc.)		X		We do not consider this a problem. Besides it is a new functionality.

Table 12.7.: Table showing which issues we decided not to deal with

usability issues in the system that we probably would not have encountered otherwise. We note however that the state of the system and range of participants prevent us from drawing firm conclusions.

All three participants were first time users of the system and for that reason they should be suited for helping us test whether the 10 minutes goal (see section 12.1 on page 80) has been reached.

The test showed that the participants did not experience major problems while navigating the user interface, which they confirmed during the debriefing by characterizing the user interface as simple and straightforward. However the meaning of the Main view sidebar categories "My" and "All" was a bit unclear but this uncertainty did not lead to critical problems when solving the exercises. Further more only a smaller number of issues categorized as serious were identified. This held together with our observation of the participants using the system seem to suggest that the 10 minutes goal has been reached.

Concerning the question *Is the distribution of information and interplay between the GUI windows sensible to the participants?*, the short answer is yes. All participants stated during the debriefing that the arrangement seemed fine and logical. The fact that there are only two options under the toolbar option New... and the Edit... button alongside it help increasing the participants' overview of the system and its ease of use.

One of the most interesting problems is related to the suboptions of the Add Reference button (issue 11 depicted in figure 12.4 on page 88). The participants found the meaning of the options "Review", "Suggestion" and "Reference" unclear. The cause of this problem is that we have let a part of the model not relevant to the user show through to the user interface. In system terms a literature object (entry) holds a list of references – thus a review is associated with a literature entry as a reference. But that is not the way the user looks at it. From his point of view a review is directly attached to a literature entry. When he seeks to add a review to a literature entry he most likely does not regard it as adding a *reference* as we have set the stage for in the program.

Unfortunately the tag functionality (#5 in figure 12.3 on page 87) was not working properly on the test day so this otherwise essential part of the program could not be examined. This would have been an very interesting test to carry out due to the unusual nature of the tags; to examine if it makes any sense to outsiders. However the main features of creating users, projects, literature entries and comments were covered. Furthermore was project literature referencing and retrieval of stored data touched.

After implementing possible solutions for most of the issues – we find that our system has a decent level of usability and thus we are of the opinion that our usability test objectives have been fulfilled.

To expand the test it would be interesting to carry out a test with the following characteristics:

- A more realistic setup outside the lab with more participants interacting with the system simultaneously.
- More of the system features covered.
- Long-term use – for instance during an entire semester.

Part V

Study Report

The study report contains academic reflections and decisions made during the process. It covers the most important decisions and subjects of discussion encountered during every part of the program development.

Academic reflection

13.1. Project and team management

Our group composition has had an impact on the work process. Some of the group members had been in a group together before and others knew each other from before enrolling at the university. Furthermore the group consists of six informatics students and one computer science student. We think that this mixed composition has been inspiring. The fact that we have different program design and programming skills has been a good platform for supporting and learning from each other.

To set up initial guidelines for our work we agreed that it would be reasonable to devise a "code of conduct". Furthermore we decided to use a wiki¹ to aid us during the project period. On this wiki we have put our code of conduct, system definition and other project related material.

Our overall work organization was primarily based on the suggestions from the courses offered – more specifically the document standard templates handed out. From these templates we produced tables for both the analysis and design documents. We lined up what each section of the report should contain and who were responsible for each part (see figure 13.1 on the next page).

During this project we had two reviews. The advantage of these – the feedback aside – was the hard and fast deadlines. In fact we feel that those deadlines ensured that the amount of work was almost equally distributed over the entire semester.

Problems encountered

We think that the idea of using the wiki was good, because it could be used for preserving important group discussions and decisions. We were however only using the wiki in the beginning of the project period. The code of conduct stated that we should have frequent meetings for discussing individual areas of responsibility, tasks and deadlines. Furthermore when a group member or a subgroup had finished a task they should present their work to the rest of the group. Those meetings and presentations were to a wide extent not carried out. The result was rather unstructured project management with ad hoc decisions that were not discussed across the entire group.

As mentioned we had planned to hold frequent meetings in the group where we should have discussed the content of the report and the work that each student is working with. We never held such a meeting but left these discussions to the free debate in the group room. We think the reason why these meetings were not held is that non of the group members were good at taking the initiative for such a meeting.

A split of the group occurred along the way, where one subgroup concentrated on programming while another worked on the report and the usability testing.

Regarding programming the client/server architecture required us to develop two applications at the same time. The advantage of this architecture is that it was easy to split the work into two such that some students were focusing on the server and some at the client (as described in section 13.5 on page 99).

¹A wiki is a web page where all the members easily can add, remove and edit the content.

Status the Designdocument				
Section	Status	Rensposible	Deadline	Completed
1. The Task				
1. Purpose				
2. Corrections to the Analysis				
3. Quality Goals				
2. Technical Platform				
1. Equipment				
2. System Software				
3. System Interfaces				
4. Design Language				

Figure 13.1.: An excerpt of the document we used to organize the design document work

The downside of the former division of the group is that one subgroup is highly familiar with one part of the project while another subgroup is highly familiar with another part. Furthermore did the division of programming tasks lead to a certain level of code "ownership".

To conclude: The exchange of information across and internally in the subgroups has been present but not sufficient. Moreover, while having several advantages particularly with respect to efficiency the parallel organization of work has the drawback compared to a sequential approach that things might get anticipated. This can lead to inconsistencies that are cumbersome to correct afterwards. A solution to this, could have been the use of more detailed project planning. That could have been a more detailed work schedule to secure a better overview for all the members in our group. Furthermore there was some slight problems, because we did not do all the exercises of the SAD. This lead to a considerable amount of work prior to the reviews, which could have been avoided if we had worked a bit more with the exercises.

Another issue we have discussed is the "Code of conduct", this document was not detailed enough and it was never revised thorough the process.

Solutions adopted

As we progressed towards the usability test we chose that three members of the group should focus on the usability test while the rest should continue on the development of the program. This were done to ensure that the quality of the usability test planning and to ensure that the program had enough working facilities to actually perform a usability test.

The tables helped us achieving a better overview of the contents of the report and the work needed to be commenced. They were tools to keep track of progress by marking completed parts (according to [6, p. 299]).

Recommendations for the future

We think that it would have been better to carry out frequent group meetings. Therefore we suggest that we in the future add a criteria to the code of conduct about exact

scheduling and agenda format of such meetings. Furthermore a frequently updated record of report and program changes from each iteration – and the arguments for introducing the changes – would have helped us devising this study report.

Furthermore it will be a good idea in the future to deploy a higher level of structure of the group work.

13.2. System and the domain

Process followed

The theme of the semester together with the project proposal made up a strong foundation for the project. We did however decide – based on a short analysis of the target audience – that the system should be using a shared data source for all the project groups. This server/client approach was chosen in order to let the project groups gain from other projects dealing with similar topics.

Initially we worked out a stakeholder analysis in order to better understand who our target audience is. While a part of the group were working on this analysis the remaining part worked on a role model analysis. These two analyses have influenced the entire project. However only the stakeholder analysis is included in the report. The role model analysis gave us inspiration for techniques that could be interesting to apply to our system. One example could be the use of tags represented in a tag cloud.

We also considered if we should use a web-based system because it – contrary to Windows Forms – has no platform requirements for the clients. Furthermore the clients would be able to use the system from any computer with access to the Internet without having to install a program which usually is not possible at for instance a library. Unfortunately this solution was not a possible due to the demands of our semester theme. Therefore both the client and server are developed using C# running on the .NET framework.

The architecture of our system dictates that the usefulness of the program will improve with the amount of users. Optimally the system would be used across different lines of study. By doing this the students might find relevant literature added by other students at another study.

Problems encountered

We encountered no problems in this process.

Recommendations for the future

13.3. Analysis

Process followed

The analysis was conducted in accordance with the "OOA&D"[6] method. This method was presented in the SAD course in shape of the standard for the analysis document.

Our starting point for the analysis was to describe the purpose of the system. The first part of this chapter was the project proposal. With this proposal in mind we devised an overview with help of the FACTOR (See section 1.2.1 on page 8) with the initial requirements for the system, this resulted in the system definition. The next step was the stakeholder analysis, that helped us realize who had influence on the system and the development process.

A recommendation from the SAD was to work with rich pictures in the report to fully identify the problem domain. The rich pictures in our report is not that detailed compared to what was shown at lectures. However we find the idea behind the pictures very reasonable but for our system they were not in so much use, because of the overall simple nature of our system domain. Though if we had been working with a real customer it would have been an very interesting way to illustrate our/their thoughts for the system.

This lead to the definition of the problem domain, describing what problem should be solved. After this we were able to start describing the essential parts of the application domain, where some of the overall classes for our system were found.

Afterwards we made the first class diagram based on the application domain. This diagram was changed after the first review. The first diagram (see figure 13.2) had a lack of structure and did not fully comply with the standard in OOA&D. Therefore the final version (see 2.1 on page 15) of the diagram is quite different compared to the first.

According to the project proposal different users should be able to add literature entries to the system but it could be assumed that only one user is accessing the system from one computer at a time. We discussed this topic a lot in the beginning of the project. The program that we were supposed to develop is aimed at project groups at a university. These project groups are all dealing with the same problem of organizing and finding literature sources. Based on this fact we concluded that it would be better if the different project groups were enabled to share what literature they use, such that groups that work with similar topics can benefit from literature that another group uses.

In order to develop such a system it would be essential that the system is based on a multiuser platform consisting of clients and a server. The server must be able to handle multiple clients at once. We all agreed that this would be the most reasonable solution to satisfy the actors of the application domain. However we had doubts concerning the complexity of this solution. We decided to design the system using this client/server architecture because we thought it would be good to learn about handling communication between different computers and how to handle multiple simultaneous requests at a server.

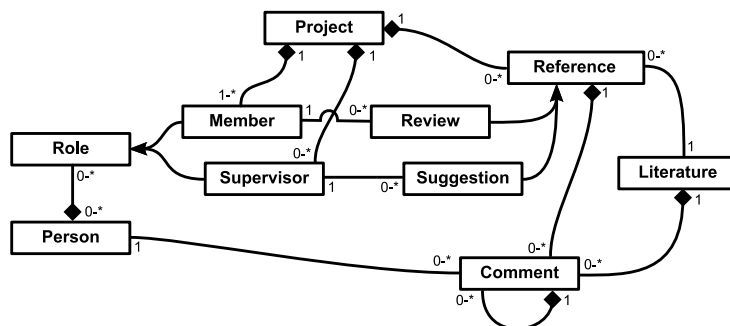


Figure 13.2.: The class diagram before the first review

Based on the class diagram we should describe the classes shown in the diagram. The classes were visualized in state chart diagrams. In the process of creating these we also

experienced some problems cause many of the diagrams turned out to be either not detailed enough, incomplete or hard too understand. In figure 13.3 one of the problematic diagrams shown.

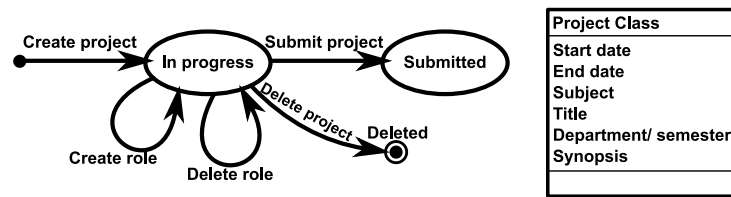


Figure 13.3.: This classdiagram is not detailed enough, in this diagram it is not possible to delete the project when it is already submitted.

In the diagram for the project class a problem arose that could have lead to a flaw in the system and misunderstandings between the developer and the customer . Afterall we consider the statechart diagram as a very useful tool in the development process, and we will certainly use them in later projects together with the class diagrams.

The next interesting topic in analysis was the usage of "personas" (See section 3.1.2 on page 22) that is fictive descriptions of possible users. The personas was used because we do not have any real users for the system. The principle about this approach is to ask yourself "Would Brian like this button?" while you are developing the system. Unfortunately we have not used the personas much during the development because some group members found it very hard to take serious. However we find the main thought about the personas interesting and with or without any users it could be a way to develop the system with respect to the users demands.

Problems encountered

Introducing the Reference class At some point in the process we had a class called *LiteratureDescription* to connect projects to literature. The idea was to rate how relevant a piece of literature was to a project. *LiteratureDescription* was basically what we now call *Review*. It was decided that it often not would be necessary to add a piece of text and a rating to a piece of literature when making a reference to it in the project in question. Thus the *LiteratureDescription* class was specialized into the three classes *Reference*, *Review* and *Suggestion*.

Applying the Role Pattern Early in the analysis process a need for a way to handle information about people in the problem domain was recognized. A class called *Person* was created. Since the system is intended to be used in not only one, but many project groups at the same time it became clear that the *People* class was not sufficient. A way to describe a person's relation to several groups was needed. This is why we applied the Role Pattern [6, p. 80] and introduced a *Role* class with the specializations *Member* and *Supervisor*. The advantage of this construct is that one person can aggregate zero or more roles simultaneously. He can have the role of member of one group while having a supervisor role in another. Or he can be member of more than one group. Furthermore *Creator* is a specialization of *Member*, since the person who creates a project is automatically member of said project.

Delay of Analysis Document Template During lectures the group was informed of a template for the analysis document. The group expected to receive the template in appropriate time, but we did not. Therefore we had some doubt whether or not any material produced by the group would be applicable in the this template, work was somewhat stunted. This was largely a fault of the teaching staff, since they repeatedly reminded the groups that the template was coming, but ultimately the template arrived very late, hastening any work that had to be included.

Experience with OOA&D

OOA&D is well suited for programming tasks where there are few unknowns in the language and framework being used. Analyzing and designing a program using OOA&D without prior knowledge of Object Oriented Programming seems far fetched when using the waterfall model, the OOA&D model specifically specifies iterations, without iterations the model is even more unrealistic.

Recommendations for the future

The system definition could to a wider extent have been used as reference in the steps following the initial one, that is, in analysis, design and implementation. Also following the waterfall model as attempted in this semester is not really recommended. An attempt to develop some preliminary software to get an idea of the extent of the task at hand is recommended.

13.4. Design

Process followed

We have used the design standard provided by the SAD and DIEB courses to structure our design process ². The different parts described in the standard were delegated to different subgroups of the project group. We devised the quality goals and factor-table in the group.

Network communication between clients and server We decided, as mentioned in section 13.3 on page 96, to design our system by the use of a client/server architecture. This decision means that we must implement some kind of network communication. We decided that the three most obvious solutions would be:

RMI because it is a simple way to implement network communication in a C# program.

Web Service because it is easy to use in other platforms.

HTTP because it would give the user access to the system from any computer with access to the Internet and a web browser.

We found that the best solution would be to implement all of those solutions in order to make the system as widely accessible as possible. We did however decide only to implement the RMI because that were the one which would be the most obvious and easiest one to implement in C# and because we wanted to minimize the amount of work to do. By using the RMI it is possible to access classes directly on the remote system.

²a copy of the standards is found in appendix B on page 114

If we had decided on using a Web Service we would need to implement a special class at both the server and client side to handle the communication. If we had used a HTTP server we would have needed to implement the user interface as a HTTP server and not by using Windows Forms.

Even though we decided only to implement the RMI, we did decide to implement the communication between the server and the client as a separate layer in the server such that it would be easy to add further network communication types to the server.

Search interface To strengthen the applicability of the project, we needed a way to make it easier for users to find new interesting literature within the system. The Tags are a crucial part of calculating the relevancy between items in the database. It allows us to implement the *ShowRelevant* function on an item, which lists other items in the system that have the same tags sorted by how many people have added those tags. Users can then easily browse through relevant literature and the projects that has used them.

It is also possible to do free-text searching, which is implemented using a slightly modified *Term Frequency-Inverse Document Frequency* [17] algorithm. This expands the possibility of finding relevant items within the system.

Problems In this context a problem could be "meta noise"³ where some users are adding irrelevant tags to literature and projects. It would be the moderators task to remove meta polluters from the system. While that solves spammers it does not help when people make spelling errors in the tags. This problem is more complex to solve, because it would not be fair to remove users that are bad spellers. Therefore it must be the job of a moderator to remove spelling errors.

Problems encountered

While developing the factor-table (see table 7.1 on page 42) it was particularly difficult to assess the impact of the factors on the system architecture and to rate the factors according to "Priority for succes" and "Difficulty or risk". This was due to the fact that we are novice system developers and thus do not possess the necessary experience to perform such assessments. Without knowledge of which parts of implementation are difficult and what is possible using OOP, designing the actual system is difficult.

13.5. Implementation

Process followed

The implementation process was highly iterative, since unforeseen problems arose as the programming process progressed. When one suggested solution did not work an alternative had to be devised. It also began halfway through the design phase, this was done in an attempt to foresee some of the decisions that needed to be made during design.

To allow more group members to work on the program at once the AccessHandler component was defined as one of the first steps of implementation. Because Client-API provides an object using the IAccessHandler interface, a fake object implementing that interface could be used instead of a working Client-API component. The interface was defined and a dummy class containing test data and fixed responses to method calls was used to allow the Graphical User Interface to be built by one team, while every other component (except AccessHandler) could be developed by a different team. Later in the

³The posting of irrelevant tags like spam or spelling errors

process when basic functionality of the other components were completed, the switch from the dummy object to an actual RMI-Client was hardly noticeable.

Problems encountered

Missing variables Objects like project and literature do not contain enough variables to describe them to a degree that will be satisfactory for all users. While more effort could have been allotted to this assignment, it was de-prioritized on the count of not being relevant to the subjects the group is trying to learn this semester. Deciding which users use which variables are not directly linked with any of the curricula followed, and because adding more variables to a class, like semester to Project, is a trivial process strictly from an OOP point of view it was not deemed important. It is arguably important for the DIEB course, as an insight into what users would expect of a system. But the amount of effort required to reach a satisfactory result was deemed too great.

Low Cohesion on AccessHandler and Catalogue Upon finishing up the functions available on the AccessHandler and Catalogue, it became apparent that Catalogue did indeed have very low cohesion, it contains functions for changing basically everything in the Model. This seemed acceptable in the earlier stages of development, but in the end the sheer size of the Catalogue class makes the program more difficult to understand and read. Splitting up the Catalogue into several Controllers instead of a single would increase the Cohesion. For example splitting the Catalogue into a Controller for each part of the DataContainer structure, one for People, one for Projects, one for Literature and one for Tags. This approach would however cause new problems such as very High coupling between the different Controllers. An example being adding a reference between a Project and a piece of Literature, this would require a call to both controllers with the exact same data in this case, who would be the Creator of the actual Reference object.

DataTable for Tags While it was the goal of the group not to delve into relational databases, implementing Tags, which basically consist entirely of indexes, was implemented using DataTable. The alternative was using several Dictionaries to solve looking up a Tag by a project ID and/or a username.

UniquelyIdentifiable During save and load of the model, it was observed that keeping the cross-references would be invalidated unless it was possible to discern projects and literature by something other than their data. Thus the superclass UniquelyIdentifiable was implemented. UniquelyIdentifiable has a variable ID and a static variable NextID. Upon creation of a new object of a class that implements the UniquelyIdentifiable class, NextID is ensured to be 1 higher than the highest used ID. The Catalogue object calls UniquelyIdentifiable.NextID to get the next viable ID for a project or literature when creating new instances.

Interchangeable Network Model The decision to make the network model interchangeable caused the program to swell in duplicate commands and information. The reason being that accessing objects without having a reference to them requires identifying them with their ID or likewise on each command. If the program was implemented taking only RMI into account, another more object oriented approach could be used. An idea floating around earlier in the process was giving clients access to a ProjectWrapper class that handles all operations on a Project while enforcing all the users permissions. This kind

of use-case controller is more in the spirit of object oriented programming, where the implemented solution is very reminiscent of procedural programming.

Custom controls The tag controls have their own internal datastructure and sorting functions. This is because we wanted to make a control with a very low coupling to the main program. We have succeeded in achieving this, however it means that some functions in the tag controls are very similar to functions in the main program. The functions for sorting tag in the main program was made after the tag controls, but because of the very low coupling the main program can not use this function. If we were to create it again, we would make the tag control so the input for them was a sorted list of tags and not as it is: A list of all tags in the system. Then it would be possible for us to use the same sorting function and datastructure in for instance in the search functions.

For presenting tags on the tag container we wanted to make a small status bar appear under each keyword. It was not possible to use the progress bar in the windows forms library because it was too large. So we decide to make our own small progress bar. This was done using the System.Drawing library. At first it worked great but if the form lost focus or if the user used the scroll bar the tags would disappear. So we needed to override the OnPaint event. Because of the way we were drawing the tag the only way to do this was to redraw all the tags in the container. Normally this is not something the user would notice, but if the user uses the scroll bar the tags start flashing especially if there is a lot of tags in the container. This problem could have been solved by making the tags themselves a custom control, which would then be instantiated inside the tag container. And instead of drawing the status bar we would have used a picture box with a image of the status bar at 100%. Then we would hide part of the image with a label if the tag weight was not at 100%.

A general problem with all our controls is that sometimes there can be problems problems with the designer in Visual studio. The problem is that the designer cannot be displayed. This happens if something is changed in the custom control and the custom control is not compiled. It can also be caused by the custom control's properties acting weird or just what seems randomly with no specific cause.

13.5.1. Data storage

Since our system is based on a server, which is going to catalogue and search for information about literature sources as instructed by the clients it would have been an obvious solution to use a database server for the storage. Another option that we discussed was to use XML for the storage. The disadvantage by using XML is that it requires that all the data must be loaded into the system memory on the server. This will of course induce the demands for system memory on the server as the system grows.

Despite the fact that XML-files have high system requirements on the server we chose to use XML in order to use what we had learned about it in the OOP course. During the implementation of the system we however discovered that the use of XML was rather slow and not easy to use in our system. Therefore we decided to use binary files for the storage. In order to accommodate the possibility of changing the data storage we decided to keep the data storage in a separate layer such that it would be easy to switch to another data storage implementation such as the use of a database server.

13.6. Unit testing

Process followed

We started coding the classes for black box testing the PersistentData and Catalogue pretty early in the project phase. The test classes was rewritten a few times before the actual program was finished. After finishing the program the test classes was updated once again.

The white box testing of the Search() algorithm was done at the end of the project phase.

Problems encountered

In general we found more errors in our own tests than in the actual program. We did encounter some logical errors in the Catalogue in the further development of the program. In theory these errors should have been discovered earlier in the black box unit testing. We did however find several flaws in the Catalogue class. These flaws was mostly related to typing errors caused by copy/pasting which obviously can lead to logical errors.

Due to large structural changes in some of the classes the CatalogueTest class had to be rewritten a couple of times.

Recommendations for the future

Unless an agile style of programming is used, e.g. Extreme Programming[16] the testing classes should first be written after all structural issues has been cleared.

The process of making a white box test of an algorithm was interesting but would properly quickly get incalculable if applied to any larger part of a program.

13.7. Usability testing

Process followed

The first thing we did before the test was to find the different participants for the test. We aimed at having four people for the test and the optimal profile for a participant was centered around the following:

- A person who is studying at Aalborg University due to the large projects we do.
- A person who had not followed the usability-lectures with objectivity in mind.
- Different people with different levels of computer-experience.
- A person that have no personal connections with any members in our group.

Initially we discussed how many participants we needed for the test and we agreed that the optimal would be 3 or 4 people.

At first we made an appointment with a four-people project group from humanistics but due to their project work they bailed out a couple of weeks before our test. Therefore we had to find four new participants. In the end we had appointments with two students from computer engineering and two people who recently abandoned our study.

We did the usability test in the usability lab at the university. To handle the test we were three people in the lab – one test monitor, a technician and an observer. The role

of the test monitor was to manage the test (see A.3 on page 108). The same monitor was used in all the tests to ensure that the three tests was carried out in identical environments/situations. Furthermore he should prevent the test participant from using non-functional parts of the system. The technician was controlling the cameras and the DVD-Recorder. The task of the observer was to act as a buffer by writing notes about aspects of the test situation that the video recording would not necessarily catch.

Before the actual test we initially accomplished a dummy-test. In this test a member from our group acted as a test-participant. This was done to examine if it was possible to carry out the tasks for the test and to make sure that the recording-equipment was functioning correct. Then the three actual tests was executed.

Afterwards we examined the recordings of the tests and took note of the various issues that were found. Then we analysed these issues and categorized them by their respective windows. Then we tried to make possible solutions to the issues by analysing the results with the test participants comments in mind.

Problems encountered

The main problem we had about the test was the fact that it was hard for us to find appropriate participants. The group from humanistics cancelled our appointment and therefore we had to find new test participants within short time.

The next problem we encountered the day of the test was the fact that one of the test participant did not show up and that another participant cancelled his appointment with us. We agreed that we had to have at least three participants so we had to find a new person very fast. The person that we found did not fit the optimal profile for our participants very well because he already knew a lot about usability. But we figured that it would be better to have a participant that did not fit the profile that well than having no test. Therefore our solution was to use a guy from another informatics-group for the test.

In the second test we knew the participant to well. During the test the participant talked a lot and we got the impression that he wanted to please us.

While evaluating the test results we realized that the tasks had some smaller issues. For example we discovered that the data that the participants were to add to the system were insufficient. If we should have avoided this problem the expected solutions in appendix A.5 on page 112 should have been written prior to the execution of the test.

Solutions adopted

We choosed to find new participants, when the group from humanistics cancelled our appointment.

Recommendations for the future

- Not to use participants from our own line of study or from our own social circle, to get more objective answers.
- To start the test-planning earlier in the process.
- Eventually carry out two usability tests, one when we have a functioning prototype of the system and another in the final phase of the development-process.
- To do possible solutions for the usability-tasks prior to the test.
- To do a test with a larger number of participants.

13.8. Conclusion

Our encounter with the object oriented approach to programming has been an educational and interesting experience. Though the decision to work with an interchangeable network architecture proved difficult to implement using the object oriented paradigm, we have found the method to be a powerful and pleasant approach to system development. Deploying the methods described in our OOA&D textbooks we have acquired a decent amount of useful tools that will aid us in future projects.

As expected the usage of the waterfall model resulted in discrepancies between analysis, design and implementation. Had we followed the model to the letter we should have rolled back to the point where the discrepancies initially manifested increasing the workload. This is simply not possible especially in a project such as ours with rather limited resources. Therefore it might have been preferable to use an iterative approach as recommended in the textbooks. This however was not an option due to the courses' linear presentation of the material.

The GUI was developed using an iterative approach which forced us to revisit crucial parts of the UI design. One major revision was in part caused by the results of the usability test which highlighted several issues in the design. Optimally the usability test should have been carried out as the last step of the process because parts of the system were incomplete at the time of testing. However at the end of the project period these parts were all complete – including all non-trivial parts – and we have covered the spectrum of features in object oriented programming that we find interesting.

- [1] Askoxford.com, 2006. <http://www.askoxford.com/>. 9
- [2] Kurt Nørmark AAU. Slides about oop. <http://oldwww.cs.aau.dk/normark/ooop-06/html/ooop.html> (18.12.06). 77
- [3] Dylan Lake David Latapie et al. Wikipedia on wiki. <http://en.wikipedia.org/w/index.php?title=Wiki&oldid=79419119>. 120
- [4] Tim Chambers Denham Grey et al. Folksonomy, revision 78202657. <http://en.wikipedia.org/w/index.php?title=Folksonomy&oldid=78202657>. 44, 120
- [5] Yvonne Rogers Jennifer Preece and Helen Sharp. Interaction design – beyond human-computer interaction. John Wiley & Son, Inc., 2002. 31
- [6] Peter Axel Nielsen Lars Mathiassen, Andreas Munk-Madsen and Jan Stage. Object oriented analysis & design. Marko Publishing, 2000. 3, 8, 12, 37, 41, 94, 95, 97
- [7] Microsoft. C# and .net. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netstart/html/cpcsvbvcjscript_c_.asp (29.10.06). 9
- [8] Microsoft. C# requirements. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconnetframeworksystemrequirements.asp> (29.10.06). 40
- [9] Microsoft. .net framework version 2.0 redistributable package (x86). <http://www.microsoft.com/downloads/details.aspx?FamilyID=0856each-4362-4b0d-8edd-aab15c5e04f5&displaylang=en#Requirements> (29.10.06). 33
- [10] Rolf Molich. Brugervenligt Webdesign. Ingeniøren bøger, Ingeniøren A/S, 2003. 85
- [11] Joseph Poole. Basic path testing. <http://hissa.nist.gov/basicpathtest/> (15.12.06). 76
- [12] Jeffrey Rubin. Handbook Of Usability Testing. John Wiley & Sons, Inc, 1994. 80, 81, 84
- [13] Jan Stage. Forms of interaction. 32
- [14] Thomas Vander Wal. Folksonomy :: Off the top. <http://www.vanderwal.net/random/category.php?cat=153>. 44, 120, 121
- [15] Wikipedia. Cyclomatic complexity. http://en.wikipedia.org/w/index.php?title=Cyclomatic_complexity&oldid=93806810 (15.12.06). 76, 77
- [16] Wikipedia. Extreme programming. http://en.wikipedia.org/w/index.php?title=Extreme_Programming&oldid=93935539 (19.12.06). 102
- [17] Wikipedia. Term frequency-inverse document frequency. <http://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=90045948> (16.12.06). 67, 99

Part VI

Appendix

Usability test

A.1. Test documents**Handouts for test participants****Person profile**

Name:.....

Age:.....

Semester:.....

Present study:.....

Declaration of consent

I hereby allow that group d103a is allowed to use video recordings from this test for further analysis and for the exam.

Name and date:.....

Signature:.....

A.2. Test introduction

In the beginning of each test the following Danish text will be read aloud to the test participants. This is done to ensure that the participant knows the course and framework of the test. Below this Danish text we have made a translation of the text into English.

Først og fremmest: tak fordi du ville være med. Du skal hjælpe os med at teste det litteraturhåndteringsprogram, vi har udviklet. Testen kommer til at foregå på den måde, at du sidder ved computeren med programmet. Jeg vil sidde ved siden af dig under hele testforløbet. I de tilstødende lokaler sidder der personer og observerer det, der foregår. Selvom det kan virke som en eksamenssituation skal du tage forsøge at tage det roligt og koncentrere dig om at løse de stillede opgaver. Husk på, at det er programmet og ikke dig vi tester. Du vil få udleveret opgaverne én af gangen, og jeg bede dig om at læse opgaven højt før du starter på at løse den, så vi sikrer os, at du forstår, hvad vi ønsker at du skal gøre. Ligeledes vil jeg vil bede dig om at fortælle mig, når du mener at have afsluttet en opgave. Undervejs vil jeg bede dig om at tænke højt. Det vil sige, at fortælle mig om dine overvejelser, løsningsstrategier og forventninger. Jeg vil ikke hjælpe dig direkte, eftersom det ikke er meningen med testen. I stedet vil jeg forsøge at stille dig spørgsmål, så du selv kan nå frem til en løsning. Desuden skal du også sige til, hvis du føler du støder ind i problemer eller er usikker på hvad du skal gøre. Hvis du på et tidspunkt kører fast, og ikke kan nå frem til en løsning, vil jeg bede dig om at sige til. Så går vi bare videre til næste

opgave. Efter opgaverne har vi en række spørgsmål, for blandt andet at undersøge, hvad du overordnet mener om programmets funktionalitet og udformningen af dets brugerflade.

English translation

At first: thank you for participating. You are supposed to help testing the literature management system that we have developed. During the test you are situated in front of the computer with the program running. I will be sitting right next to you during the entire test. Observers are situated in the adjacent rooms. Even if it might feel like an examination you should try to relax and concentrate on solving the tasks. Remember that it is the program and not you we are testing. You receive the tasks one at a time and I will ask you to read it aloud before trying to solve the task such that we can be sure that you understand what we want you to do. Furthermore I request you to tell me when you think you have completed a task. During the test I ask you to think aloud. This means that you should tell me about your considerations, strategy for solving the problems and expectations. I will not help you directly since that is not the purpose of this test. Instead I will try to ask you questions such that you can find a solution by yourself. In addition you should tell me if you feel that you get problems or are uncertain about what to do. If you at a time get stuck and cannot find a solution I ask you to let me know. If that happens we just proceed with the next task. When all the tasks are completed we have some final questions in order to uncover what you in general think about the functionality and arrangement of the program user interface.

A.3. Tasks for the usability test

In the following we will develop the tasks that our participants in the usability test must solve. We will first describe the tasks using task tables followed by short task descriptions which are templates to the exercises that the participants will be introduced to.

Task: Create new user

Task Component	Description
Task	Create new user and login
Machine State	Login window
Successful Completion Criteria	The test participant creates a new user with a username and password and finally logs in to the system
Benchmark	Successful login within 2 minutes

Table A.1.: Create new user

The task

- Create your own user account
- Login with the new username and password

Task: Create new project and add users to this project

Task Component	Description
Task	Create new project with keywords
Machine State	Main window
Successful Completion Criteria	The test participant creates a new project with custom data and the project becomes viewable in "My projects"
Benchmark	This task should be completed within 4 minutes

Table A.2.: Create new project

Task Component	Description
Task	Add 2 users to a project
Machine State	Main Window and several preentered entries
Successful Completion Criteria	The project and the 2 users is present on the screen
Benchmark	This task should be completed within 1 minute

Table A.3.: Add users to the project

The task

- Create your own project.
- Add the following users to your project: "john82" & "gitte.pj"

Task: Create new literature entry

Task Component	Description
Task	Create new literature entry
Machine State	Main window
Successful Completion Criteria	The test participant creates a new literature entry with custom data and the literature becomes viewable in "My literature"
Benchmark	This task should be completed within 2 minutes, since no summary has to be entered

Table A.4.: Create New Literature Entry

Task Component	Description
Task	Add literature to existing project
Machine State	Main Window with several entries in the database
Successful Completion Criteria	The literature has been added to the project and can be viewed in the list of literature for the specific project
Benchmark	This task should be completed within 1 minute

Table A.5.: Add literature to existing project

The task

- Add the following book to the system:

Netværksbogen Mølgaard og Nielsen IDG, 2002

- Associate the book with your project.

Task: Find literature entry

Task Component	Description
Task	Find the most used tag for a particular literature
Machine State	Main Window with several preentered literature entries
Successful Completion Criteria	The user finds the tag cloud/rating and a specified tag
Benchmark	This task should be completed within 2 minutes.

Table A.6.: Find most used tag for a literature entry

The task

- Find a book using the keyword "Usability testing". The correct answer is:

Handbook of usability testing Jeffrey Rubin 1994 John Wiley & Sons, Inc.

Task: Add comment

Task Component	Description
Task	Comment on a specific literature entry
Machine State	Main Window with several preentered entries
Successful Completion Criteria	The participant comments on the specified literature entry and the new comment is viewable afterwards
Benchmark	This task should be completed within 2 minutes.

Table A.7.: Add a comment literature entry

The task

- Add a comment to a specified literature entry

A.4. Exercises for the usability test

Below are the exercises that we handed the test participants during the test. The exercises are written in Danish but are presented in English in section 12.1 on page 82.

Opgave 1

Forestil dig at du som studerende gerne vil gøre brug af litteraturprogrammet. Opret dig selv som bruger (Med dit fornavn som brugernavn og et password efter eget valg) og log ind i programmet.

Opgave 2

Din gruppe har bedt dig om at oprette jeres projekt med titlen "IT i Aalborg" i programmet. Følgende tags (nøgleord) skal knyttes til projektet *Aalborg, IT, computer, usability og KMD*. Ligeledes skal dine to gruppekammerater "john82" og "gitte_pj" knyttes til projektet.

Opgave 3

I har fundet en bog, som er relevant for jeres projekt. Opret denne bog i systemet med følgende informationer:

Titel: Netværksbogen
 Forfattere: Mølgaard og Nielsen
 Publikationsår: 2002
 Kilde: IDG

Dernæst skal du sikre dig, at bogen er tilknyttet jeres projekt "IT i Aalborg".

Opgave 4

Jeres vejleder har anbefalet en bog til jeres projekt, men kan imidlertid kun huske, at den **er** oprettet i programmet og at den handlede om "usability testing". Hvad er denne bogs fulde titel?

Opgave 5

Du vil nu tilføje en kort kommentar til bogen Handbook Of Usability Testing af Jeffrey Rubin, som du skulle finde titlen på i sidste opgave. Kommentarens ordlyd skal være: "Bogen er god, fordi den indgående behandler aspekterne af en test".

A.5. Expected solutions to the exercises

In the following section we have developed expected solutions to the exercises given to the participants in order to ensure the quality of the exercises.

Expected solution for exercise 1

Prior to this exercise the program has been started and shows the login form. To solve this exercise Brian clicks the "Create New User.." label. In the following form Brian types in his name: "Brian Jensen". In the next field which is "Department" he types: "Computer Science" and as an e-mail address he types: "brian84@hotmail.com" in the corresponding field. In the "Username:" field he types "brian84" because he wants to use the same user name as he is using for his e-mail. In the last two fields Brian types "fido" which is the name of his dog in the "Password" fields. Afterwards Brian pushes the "Create User" button. The program informs: "User was created!" and returns to the Login form. Brian quickly types in the user name "brian84" and his password "fido" that created before. Finally Brian clicks the "Log in" button which makes the program proceed to the Main window.

Expected solution for exercise 2

Brian realizes that there is a button with the text "New..." and therefore decides to click this button. As he does this the program gives him two options: "Project" and "Literature". Brian therefore clicks on the "Project" option because he wants to create a new project as he was asked to in the exercise. This makes the program show the "Project" form. After a short glance at the new form Brian types "IT i Aalborg" in the "Title" field followed by "IT" in the "Subject" field and "Computer Science" in the "Department" field. Brian afterwards clicks the "Save and close" button in order to save the project. This makes the program store the project and jump to the "My - Projects" view and show the project that Brian has entered in the list. Brian chooses to double click the project that he created before and sees that he can click on the tab "People". In this tab Brian clicks the "Add member.." button which brings him a new form in which he can see all the available users in the system. Brian then browses the list and finds "gitte.pj" and clicks the "Add member" button which makes the program return to the people tab where the newly added member appears. Afterwards Brian repeats the procedure for the user "john82" and finishes the exercise by clicking the "Save and close" button.

Expected solution for exercise 3

Brian starts by clicking the "New..." button that he used earlier in the project creation and instead chooses "Literature". The program shows the "Create a new literature entry" form. Brian fill "Netværksbogen" in the "Title" field, "Mølgaard og Nielsen" in the "Author(s)" field, "2002" in the "Year of publication" field and "IDG" in the Source field. Brian was not provided with a ISBN number or a summary so he simply leave these fields empty. Then Brian Click the "Create Literature" button which make the program return to the main window and switch to the "My - Literature" view showing the newly created book in the list. Brian then double clicks the new entry and the program shows the "Literature entry" form. Brian chooses to click the "References" tab. In this tab Brian clicks the "Create Reference" button and chooses the "IT i Aalborg" project that he created earlier and finally clicks the "Save and Close" button to save the changes.

Expected solution for exercise 4

After a short glance at the user interface Brian sees that the main window has a search field in the upper right corner. Brian quickly enters "usability testing" in this search field and the program returns one result showing a book with the title: "Handbook Of Usability Testing" which Brian Claims is the title of the book that the supervisor wanted to suggest the group.

Expected solution for exercise 5

Brian double click on the book he found in the last exercise which make the program show the "Literature entry" form with detailed information about the book. Brian sees that the book has a "Comments" tab which he then clicks. In this new tab Brian sees that there is a button called "Create Comment" which he then clicks. By clicking this button the program makes the two fields for "Details:" and "Comment Text:" available together with a new button called "Submit comment". Brian realizes that he must write some kind of a comment title in the "Details" field and therefore write "Good book" in the "Details:" field, and proceeds by entering: "The book is good because it treats the aspects of a test intensively" in the "Comment Text" field. Brian then clicks the "Submit comment" button to store the comment. Finally he clicks the "Save and close" button and claims that the exercise is completed.

A.6. Debriefing questions for the usability test

1. Hvad husker du bedst fra testen?
2. Hvad synes du om brugerfladen?
3. Hvad synes du om organiseringen af informationen på de forskellige skærbilleder?
4. Hvordan var det at finde rundt i programmet?
5. Var der noget der generede dig ved programmet?
6. Føler du at der mangler noget i programmet?
7. Hvad fandt du godt ved programmet?
8. Ville du kunne se en nytte i dette system på dit eget studie?

Analysis and design standards

On the following pages can be found the analysis and design standard documents that were handed out during the semester.

Analysedokument: Standard

Denne standard er en modificeret version af standarden i OOA&D-bogen, kapitel 16.2.

Opgaven. Kortfattet beskrivelse af dokumentets baggrund og sammenhæng med det formål at formidle overblik til reviewer.

- 1) **Formål.** Den overordnede hensigt med systemudviklingsprojektet.
- 2) **Systemdefinition.** Sammenfatning af IT-systemets helhedsegenskaber. Jævnfør BATOFF-kriteriet i afsnit 2.7.
Brug stakeholders til uddybning af elementet anvendelsesområde.
- 3) **Omgivelser.** Beskrivelse af relevante forhold i omgivelserne. Kan blandt andet kan omfatte rige billeder. Se afsnit 2.3.
- 4) **Problemområde.** Uformel og kortfattet fremstilling af centrale fænomener i systemets problemområde.
- 5) **Anvendelsesområde.** Uformel og kortfattet fremstilling af aktører og arbejdsopgaver.

Problemområdet. Beskrivelse af klasser, struktur og dynamik. Se del II.

- 1) **Klynger.** Klyngestruktur. Se afsnit 4.2. Giver et overblik over klassediagrammets grund opbygning.
- 2) **Struktur.** Klassediagram omfattende generaliserings-, aggregerings- og associeringsstrukturer. Se kapitel 4.
- 3) **Klasser.** Klasserne beskrives enkeltvis. For hver klasse beskrives:
Definition. Kortfattet karakteristik af klassens objekter.
Adfærdsmønster. For eksempel beskrevet med et kommenteret tilstandsdiagram. Se afsnit 5.2.
- 4) **Hændelser.** Hændelsestabel samt sekvensdiagrammer for relevante fælles hændelser. Se kapitel 3.

Anvendelsesområdet. Samlet beskrivelse af brug, funktioner, grænseflader samt andre krav til IT-systemet. Se del III.

- 1) **Brug.** Beskrivelse af systemets samspil med omgivelserne. Se kapitel 6.
 - a) **Oversigt.** Aktørtabel, der viser, hvilke aktører og brugsmønstre interaktionen består af.
 - b) **Aktører.** Aktørspecificationer for alle aktører.
Brug personas til uddybende beskrivelse af en eller flere aktører.
 - c) **Scenarier.** Overordnet beskrivelse af systemets faciliteter og brug.
 - d) **Brugsmønstre.** Brugsmønsterspecifikationer eller tilstandsdiagrammer for brugsmønstre.
- 2) **Funktioner.** Beskrivelse af edb-systemets funktionalitet. Se kapitel 7.
 - a) **Komplet funktionsliste.** Liste af funktioner med funktionstype og kompleksitetsvurdering for hver funktion.
 - b) **Specifikation af funktioner.** Komplekse funktioner specificeret i relevant detalje.
- 3) **Brugergrænsefladen.** Sammenhængende fremstilling af centrale krav til IT-systemets brugergrænseflade. Se kapitel 8.

- a) **Mål for brug.** Prioriteringsskema med mål i forhold til brugbarhed (usability) og brugeroplevelse (user experience).
- b) **Begrebsmæssig model.** Karakteristik af den grundlæggende form hvorunder brugeren interagerer med systemet. Kan udtrykkes i forhold til aktiviteter eller objekter.
- c) **Interaktionsform.** Beskrivelse af den eller de interaktionsformer, som forventes anvendt brugergrænsefladens elementer.
- d) **Generel interaktionsmodel.** En komplet oversigt over interaktionsrum for hele brugergrænsefladen og de opgaver, der hører til hvert interaktionsrum.
- e) **Den tekniske platform.** Skitse af den tekniske platform og grænseflader til andre IT-systemer og apparater.

Anbefalinger. Argumentation for det videre udviklingsarbejde.

1. **IT-systemets nytte og realiserbarhed.** En vurdering af kravenes relation til omgivelserne og de tekniske muligheder.

Strategi. Anbefalet strategi for det videre udviklingsarbejde.

Udviklingsøkonomi. Estimat af ressource- og tidsforbrug ved det videre udviklingsarbejde.

English version

Standard for design document for 2. review:

The standard is based on "skabelon for et designdokument" p. 298 chapter 16 (OOA&D). There are though some changes:

Chapter 1 and Chapter 2 are the same.

The Chapter 3 architecture is changes to

- 3.1 Design criteria and requirements crucial for the architecture
- 3.2 Generic design decisions
- 3.3 Componente architecture (as 3.1 in the book)
- 3.4 Exemplary design (the design of at least one – better two use cases)

Chapter 4 Components are changed with regards to the userinterface.

4.3. Userinterface component

4.3.1. Presentation model

This paragraph shows the class diagram with all interaction spaces (classes) of the user interface component. Each class is described in the diagram with input and output attributes and actions.

4.3.2. Interaktion spaces (classes)

This paragraph contains a description of each of the interaction spaces not contained in any other interaction space – often a window. The description contains:

- Interaction form
- A drawing of the physical design
- Reference to the use-cases, that they are used in

All other components are described as suggested in the chapter 16. .

In the component paragraphs the interfaces to other components should be described. The chapter will serve as a summery for the design work carried out until the review.

Chapter 5 can be omitted or included.

A new Chapter 6. Programming should demonstrate for the reader that the architecture could be implemented by illustrating the implementation of the exemplary use cases. The group should include **parts** of the code, to make it possible to comment on how well design and implementation fit each other.

The amount of material for 2. review:

In chapter 1 and 2 we expect a description with the scope of all the system.

In chapter 3, 4 and 6 we expect the description to cover at least the design decisions necessary for at least one non-trivial use case, but more is very welcome.

Technical Memos

C.1. Passing data from server to client

Summary Passing a reference to the original Project object is BAD because the client can then bypass the Accesshandler when changing stuff.

Factors Security or something

Solution Creating a struct which contains the id and title of the project.

Motivation Better security

Unsolved problems none

Alternatives ... Profit?

C.2. Using a DataTable structure for storing Tags

Summary Reperesenting a tag in the model is problematic. It has to support lookup using a uniqueid, a username or the tag name.

Factors Persistent Data, Effeciency

Solution Using DataTable class from System.Data

Motivation DataTable is designed to store data. It represents a relational table as used in database management systems. DataTable has the functionality we seek using minimum amount of data representation.

Unsolved problems DataTable is a part of the Database implementation in .NET which, in this project, is not used elsewhere.

Alternatives Implementing 3 different dictionaries containing ids, usernames and tag-names and allowing lookup that way.

C.3. Saving DateTime using XML

Summary DateTime.ToString() does not represent the precision of the actual DateTime object thus equals FAILS!

Factors FRUSTRATION!

Solution use DateTimeToFileTime() and recover using DateTime.FromFileTime(long);

Motivation AVOID FRUSTRATION!

Unsolved problems none plz

Alternatives Override DateTime and fix the ToString()

C.4. Error Handling in Persistent Data

Summary Whether or not the Persistent data storage returns an error message or throws an exception on error.

Factors Error Handling & Persistent Data

Solution Using Exceptions on the persistent data layer.

Motivation Because any error on the persistent data is Fatal. The Persistent data should handle the errors and if it cant the program throws an exception.

Unsolved problems Preventing said exceptions.

Alternatives Using an error message and ignore that the program is unable to save for a while (or ever?)

C.5. UniquelyIdentifiable

Summary Project class and literature class needs to be uniquely identifiable.

Factors Persistent Data

Solution Adding a new static variable to Project and Literature classes.

Motivation Trouble with comparering objects after sending them over network.

Unsolved problems None

Alternatives None

C.6. XML

Summary We have chosen not to use the C# build-in XML Serializer.

Factors Persistens on files

Solution Use some alternative C# XML-classes, eg. XmlWriter/XmlReader and Xpath for traversing.

Motivation The build-in C# XmlSerializer cannot be used on objects without a parameterless constructor. That means we should change our overall structure.

Unsolved problems None

Alternatives Do not use XML at all.

Role model analysis of del.icio.us

del.icio.us is "a social bookmarking website"¹, which means that it is a place where everybody is free to submit bookmarks to, and let everybody else look at them. This is nothing new, a lot of places will let you do this. What is so interesting to a literature management system such as the one this project attempts to create is the organisation of these bookmarks. This is done through a folksonomic[4] system and presented to the user as shown in figure D.1.

In the figure all the terms are listed alphabetically, and you will notice that some of the terms are considerably larger than others (for example "blog" and web2.0" are much larger than "game" and "sex"). This indicates that those terms are more popular to the users of this particular community.

D.1. Folksonomy

Folksonomy is a term first used by Thomas Vander Wal[14], who created the word from the two terms "folk" (old English word meaning people) and "taxonomy" (Greek word meaning classification management) to describe a phenomenon that had developed on the web, for example in a project undertaken by the World Wide Web Consortium, called Annotea. Here, information is organised not in a tree structure (a simple menu structure, as used in many web shops and personal and corporate websites) or as a mindmap (the wiki[3] paradigm), but in a structure defined by the users of that information by way of tags.

The idea is to let each user (identity) assign tags (metadata) to pieces of information (objects), using a vocabulary common to the group to which the user belongs (community). See also figure D.2 on the next page for a graphical description of the connections. The importance of a single tag to a single piece of information is defined by how many

¹<http://del.icio.us/about>



Figure D.1.: The del.icio.us tag cloud

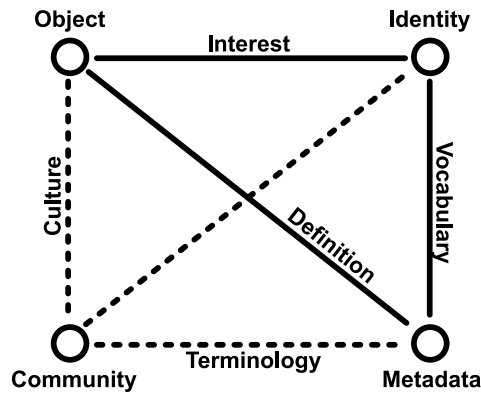


Figure D.2.: The Dual Folksonomy Triad ([14])

users decide to assign that tag to the information, and the importance of a single tag is similarly defined by how many times it has been assigned to something.

D.2. Item relevance

The tag cloud, however, is not the only representation of the information. Each piece of information has a representation of its own, where tags relevant to it can be found. In the case of del.icio.us the tags in these listings are shown in order of importance in these listings, as shown in figure D.3 on the following page. The first image in the figure shows the default view, which presents the user with those items that have been tagged (saved in del.icio.us terms) last, while the second shows the view of the most popular items with the same tag (in this case "news").

D.3. Relevance

The relevance to a system such as ours is, that organizing such large amounts of data as a literature database, and more importantly navigating it and discovering new information in that database, can become difficult if you have no clear way of finding it, other than searching through clear text. If you have to think the terms up yourself, you might miss something that others have thought of.

D.4. iTunes

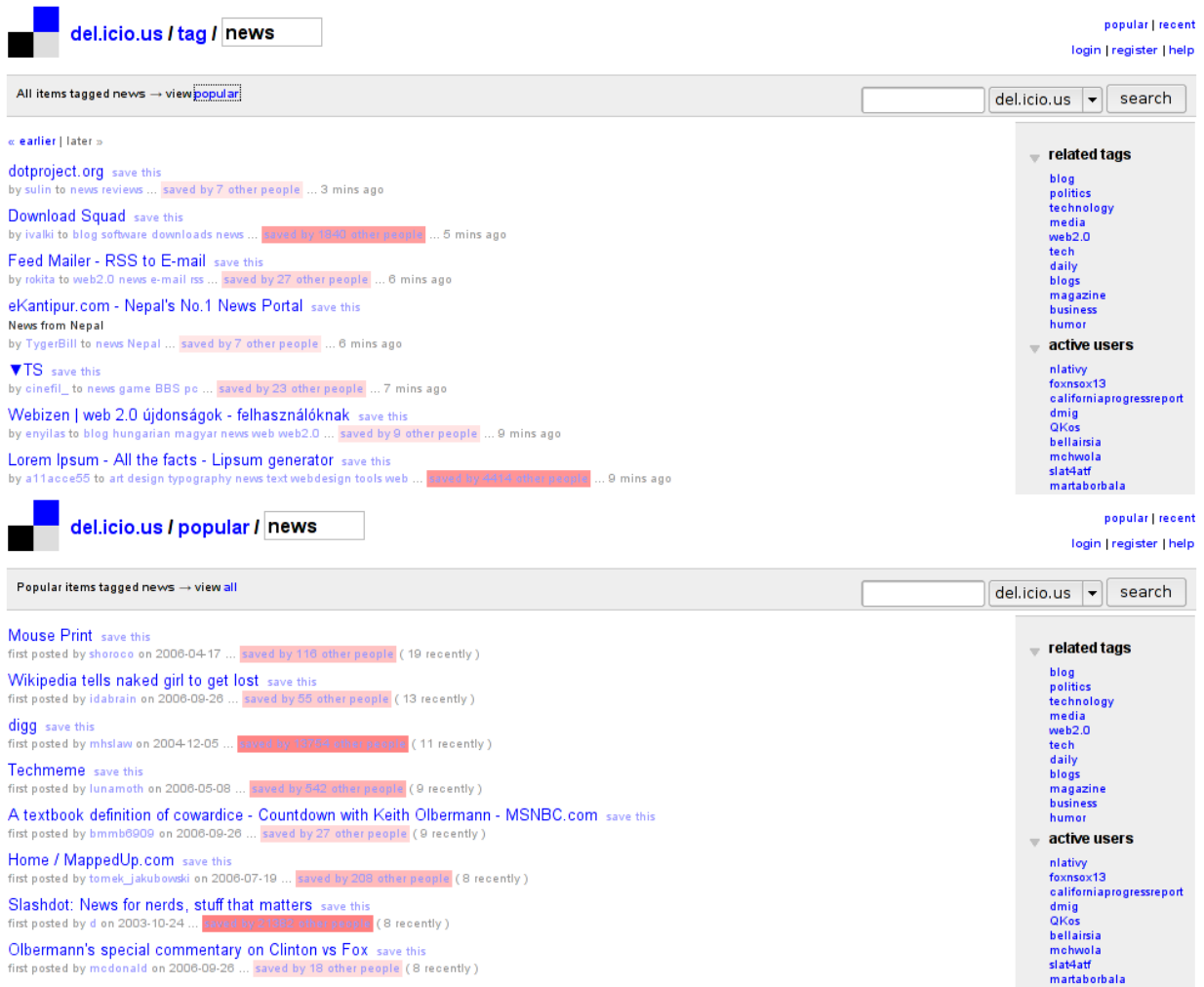


Figure D.3.: Tag presentation on del.icio.us

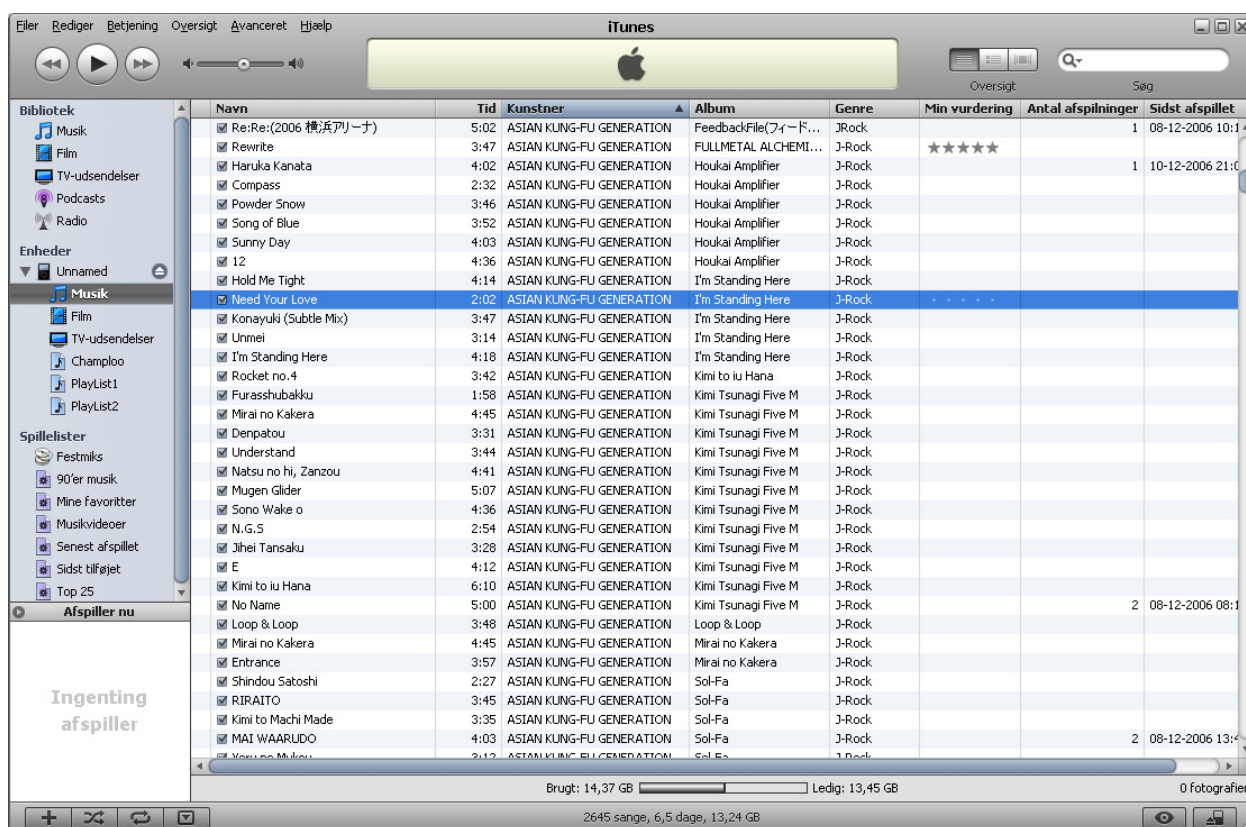


Figure D.4.: The Apple iTunes interface.