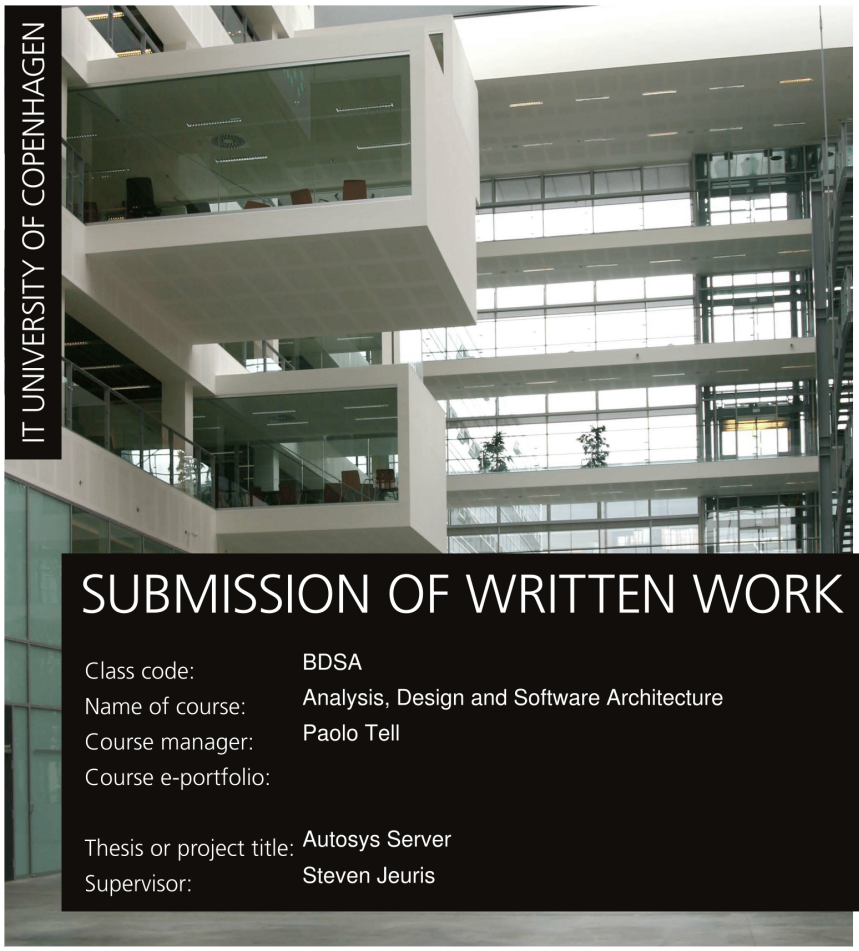


IT UNIVERSITY OF COPENHAGEN

SUBMISSION OF WRITTEN WORK

Class code:	BDSA
Name of course:	Analysis, Design and Software Architecture
Course manager:	Paolo Tell
Course e-portfolio:	
Thesis or project title:	Autosys Server
Supervisor:	Steven Jeuris

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Dennis Thinh Tan Nguyen	01/04/1993	dttn@itu.dk
2.	Jacob Mullit Møniche	22/04/1992	jmmo@itu.dk
3.	Thor Valentin Aakjær Nielsen Olesen	14/02/1995	tvao@itu.dk
4.	William Diedrichsen Marstrand	25/07/1993	wmar@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Exam Project: Autosys Server

Dennis Thinh Tan Nguyen, Jacob Mullit Mniche,
Thor Aakjr Nielsen Olesen, William Diedrichsen Marstrand

17. december 2015

Indhold

1 Requirement Analysis Document

Requirement Analysis Document - Yellow Project

William Diedrichsen Marstrand, Dennis Thinh Tan Nguyen,
Jacob Mullit Miniche, Thor Valentin Olesen

December 16, 2015

Contents

1 Introduction

1.1 Purpose of the System

Today, the amount of data has grown to a level where it is starting to challenge researchers who need to extract the best available research on a specific question. As a result, researchers apply systematic studies on big data sets where they exploit meta data to classify subsets with useful data. This requires smart data processing tools that can use meta data to narrow down relevant research data. The purpose of such a systematic review is to sum up the best available research on a specific question. This can be achieved by combining the results of several studies. In this regard, the system in this project is comprised of two parts, client and server side. Our system scope is restricted to the server side and shall support the configuration of summarized research data relevant to a given research question.

1.2 Scope of the System

The server shall provide teams with tools to conduct secondary studies (SMS or SLR). It should support activities of planning and conducting a review distributed in a research team. The server shall make sure that all data is stored for use in setting up a study configuration requested by the client. The system should be able to import information from a bibtex file to a database. The allows to populate the database with existing data. Security matters (e.g. user authentication) are not taken into primary account due to the scope of the project. In order for the system to fulfill the previously described purpose, it has to support the tasks described in the "Proposed System" section. Among these, the system shall support management of distributed research systems to work on a study. The reviewers should be able to export data sets and filter them using inclusion and exclusion criteria. Finally, the system should allow specific sets of data to be reviewed and screened by specific members of a research team.

1.3 Objectives and Success Criteria of the Project

The system should be easy to deploy and install. It should include an installation and user manual used to describe how to configure and prepare research papers for screening. It should be easy and quick to distribute relevant data and the overview should outcompete the ones achieved in other third-party programs such as e.g. Excel. The system should define rules for which data goes to whom to achieve a succesful screening of paper and efficient data extraction. The yellow system has to provide a user interface from which the blue team can extract data based on user roles and rules. The system has succeeded if users in the blue team can query the yellow system for relevant studies and tasks based on a given study configuration. Specifically, the yellow system should collect research and aggregate stacks of research material based on a research question. The blue system should then efficiently be able to extract a subset of primary studies provided from the screening of the search hits in the yellow system. The configured data may then be used by reviewers in the blue system (visualize, sort, export and categorize). Finally, the system should be able to replicate an existing study.

1.4 Definitions, Acronyms, and Abbreviations

- Systematic Studies: methodology used to sum up the best available research on a specific research question or topic.
- Yellow system: server side also referred as "server". The main responsibility of the yellow system is to store, send, validate and manipulate data from the database and also configure a study.
- Blue system: client side also referred as "client" The main responsibility of the blue system is to visualize data from the database according to user demands and to manage teams.
- Study: the whole work process from initiating a research to narrowing down relevant research evidence. A study consists of different phases where data is continuously synthesized and approved by users with different roles. The end result is a final set of primary studies used to clarify the research question.
- Primary and Secondary study: secondary research is defined as an analysis and interpretation of primary research. The method of writing secondary research, is to collect primary research that is relevant to a given topic and interpret what the primary research found.
- Phase (stage): is a given set of review tasks. Each phase is dependent on each other sequentially and is completed in a fixed order. Each phase details how task requests are handled and handed out. Optionally, a user role can proceed to the next phase even though other user roles have not finished the same phase yet (e.g. two concurrent review phases).
- Task: an assignment in a given phase in a study. A task is defined by a unique id, a set of visible data fields (unmodifiable), a set of requested data fields (modifiable) and a type. A task type can either be a request to fill out data field(s) or a request to handle conflicting data field(s). By way of example, a phase could involve review tasks assigned for two reviewers. A validator could then analyze any inconsistencies between the work of both reviewers in a second phase.
- Data Field: a data type that determines the type of a criteria used in a task to fill out data or solve conflicts.
- Research Protocol: the rules on how a given study is configured (work flow).
- Study Configuration: the configuration of a given study. A study configuration involves defining the research question, phases, assigning a team for the study, choosing data fields and assigning roles for all team members.
- Inclusion/-Exclusion criteria: criteria that is evaluated throughout the whole lifetime of a given study. By way of example, a criteria could be whether the data is from later than 2005. The criteria is used along the way to synthesize the data. As opposed to the classification criteria that is only used in the end of the study.
- Classification Criteria: classification criteria is used to group data fields in a given study. By way of example, one could classify two groups for the same set of primary studies. The classification criteria determines what one is looking for in the data. It is defined in the study configuration and is used in the end of a study upon data extraction. This is used to approach or investigate the same data in different ways.
- User: a person who is either a manager or a researcher. A manager can create a team while also being a researcher. A researcher can only have one role in a specific phase of a study.

1.5 References

Tell, Paolo, and Steven Jeuris. Autosys: Supporting Distributed Teams Performing Systematic Studies. 2st ed. Copenhagen: ITU, 2015. Print.

1.6 Overview

The rest of the document will describe the current systems available and the proposed system in our project along with the requirements collected from users, customers and stakeholders. The system requirements are used to generate system models such as potential scenarios and use cases.

2 Current System

The current system is an Excel based application that contains data about contents of articles and facilitates searches on academic topics. It is possible for multiple users to contribute data and one can identify if another user is contributing data on the same article. The contributed data is mostly keywords, which makes it easier to search and compare articles. To search for articles, a user must submit keywords, which the articles will be ranked after. Articles are also ranked after the amount of keywords they have in common with other articles. The current system works but is increasingly difficult to manage. Thus, a new system which solves the negatives while keeping all of the functionality that Excel already supports is recommended. This can be done with a separation of the data and the user interface, by creating a dedicated database and a dedicated user interface. This will ensure that new functionality will not be limited by the database structure. The negatives and pros of the current system are described below:

2.1 Pros of current system

The current system pros are as follows:

1. Excel licenses are cheap and widely used by customer base
2. Simple and easy to augment and implement changes.
3. Data is safe from hostile users if excel files are read only.
4. Easy to implement version control on .csv files.

2.2 Negatives of current system

And the systems negatives are as follows:

1. The data structures can be hard to manage, especially as the system grows in scope and content.
2. The system does not support multiple users editing the same files at the same time.
3. The system interface leaves a lot to be desired.
4. New features becomes hard to implement as the system grows.

3 Proposed System

3.1 Overview

This section will describe the requirements of the system. In other words, it will clarify the purpose of the system with functional and non-functional requirements. The functional requirements will describe the services that the system should provide and how the system will behave. The non-functional requirements will describe the constraints of these services and are used to measure the quality of these.

3.2 Functional Requirements

The system must support the following functional requirements:

- The user must be able to define the phases of a study.
- The user must be able to create and manage classification criteria for different studies.
- The user must be able to create and manage inclusion and exclusion criteria.
- The user must be able to configure future phases in a study.
- The user must be able to store information about delivered tasks.
- The user must be able to create and manage teams and members.
- The user must be able to assign teams and members in the configuration of a study.
- The user must be able to filter research papers based on demographic information specified by the user.
- The user must be able to screen imported research papers. The result should be an export of papers consisting of primary studies (set of relevant papers).
- The user must be able to export studies as plain data sets in different formats, e.g. as comma-separated-values(cvs) format.
- The system must support a role engine which identifies user rights. Possible roles to be supported could be a "viewer" and "validator" role.
- The system must be able to handle multiple client requests concurrently.
- The system must be able to store quality ratings of research papers based on coverage according to specific research criteria.
- The system must be able to extract data samples from specific studies. These are used to clarify any inconsistencies between different reviews of the same study. The data sample is accepted or rejected given in percentage terms - and is part of a phase in the study.
- The system must be able to generate a research protocol that describes the rationale, objectives, design and methodology of the data and configuration of a study.

3.3 Nonfunctional Requirements

The system must support the following nonfunctional requirements:

3.3.1 Usability

- The user must be able to configure a study through the use of at most 3 application windows.
- A user with no background in IT should be able to understand the entire set of tools, which the system provides in approximately 30 minutes.
- The user manual must provide knowledge about the interfaces provided by the system. This includes how a study is configured, how to manage teams and members and operations that the system support in general.

3.3.2 Reliability

- System restart should be done within 30 seconds upon system failure.
- The latest stored data should always be available through a backup.

3.3.3 Performance

- The system should identify the role of a user and load the appropriate studies within 15 seconds.
- A user request for specific data should be handled no more than 10 seconds after it is received.
- The time before a response is sent should be no more than 30 seconds after the request is received.
- The system should support at least 200 users.

3.3.4 Supportability

- The server is maintained by the supplier of the system.
- The system must be fully functional independent of the other subpart of the complete system (the blue component).
- The system study configuration user interface must be replaceable.

3.3.5 Implementation

- The system should run on any Windows operating system newer than Windows XP.

3.3.6 Packaging

- The set up and installation of the server is done by the provider of the system, independent of the client.
- An employee with no prior knowledge about the system must be able to install and set up the server within a period of 15 min.

3.3.7 Legal

- The system should be publishable as Open Source Software in accordance to the GNU General Public License v2.0.
- The system should store data on a local Microsoft SQL Server provided by the IT University of Copenhagen following the rules issued by the institution. It should be able to host it on other platforms like, e.g. MySql.

4 Scenarios

This section contains various scenarios regarding operations between the user and the system.

Scenario name	bobStartsNewStudyConfiguration
Participating actor instances	bob:`Researcher` System:`AutoSys System` Configuration UI:`StudyConfigurationUI`
Flow of events	1. Bob the `Researcher` has to start a new research and opens the client from `StudyConfigurationUI`. 2. Bob logs into the client and navigates to the "Study Configuration" page. 3. Bob specify two reviewer and one validator and defines a research question based on some inclusion- and exclusion criteria to specify what papers should be returned. 4. Bob confirms his study configuration and sends the request to the server by pressing "ok". 5. The `AutoSys System` returns an overview of the study configuration to the client.

Table 1: Scenario when a user creates a new study configuration

Scenario name	ClientFilteringOperation
Participating actor instances	server:`Server` client:`Study Review UI`
Flow of events	1. `Server` is receiving a request from `Study Review UI`: Filtering keywords Design pattern, 2005, A gang of four 2. `Server` validates clients authentication. User credential is accepted. 3. `Server` measures all studies based on the given keywords. 4. `Server` finds 20 studies and a list is formed. Each study element in the list contains data about it. 5. `Server` replies to the clients request by sending the article list of found articles. 6. `Server` returns an overview of the `Study Configuration` to the `Study Review UI`,

Table 2: Scenario when a user sends a request with given filtering keywords trough the `Study Review UI`.

Scenario name	ClientGetsToManyRelevantPapers
Participating actor instances	server:`Server` client:`Study Review UI`
Flow of events	1. `Server` is receiving a request from `Study Review UI`: Search keywords 2001,2002,2003,2003,2004,2005,2005,2006,,2007 2. `Server` validates `User` authentication. `User` credential is accepted 3. `Server` measures all articles based on the keywords. 4. The list containing papers exceeds 10.000 papers, and an error will be displayed on the users screen. The error will also instruct the user to narrow his search to be able to display his search results

Table 3: Scenario when a user has requested to many papers during one request.

Scenario name	ExcludingPapersAboutDesignPatterns
Participating actor instances	server:`Server` client:`Study Review UI`
Flow of events	1. `Server` is receiving a request from `Study Review UI`: Search keywords Design pattern, 2005, A gang of four, ExcludingAllNonAssignedArticles. 2. `Server` validates `User` authentication. `User` credential is accepted. 3. `Server` searches for articles with the given Search Keyword, but are excluding articles which are not assigned to the user. 4. 10 relevant articles was found but 5 was excluded because of the exclusion criteria. 5. A list with the remaining 5 articles is returned from `Server` to the `Study Review UI`

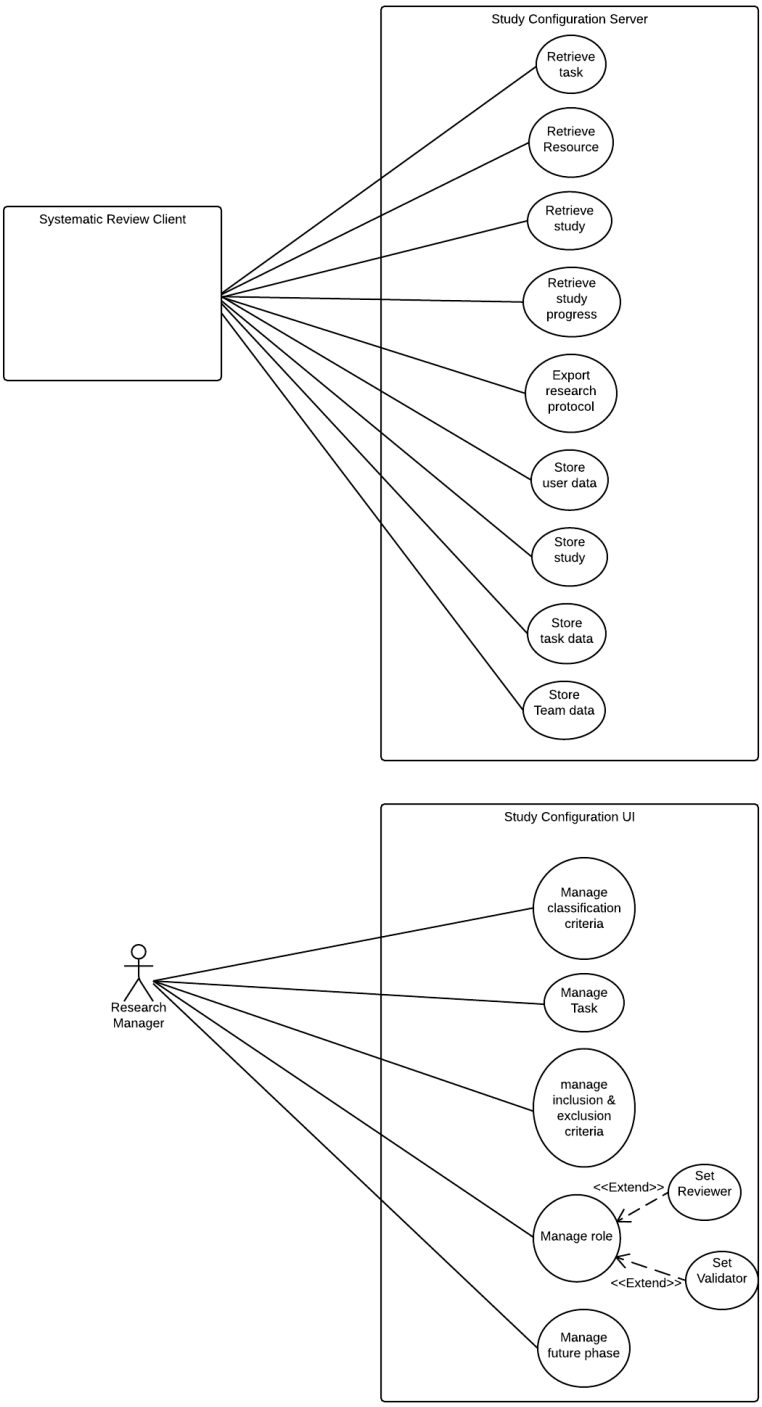
Table 4: Scenario when a user wants to exclude some papers.

Scenario name	ClientRequestWithInvalidUser
Participating actor instances	server:`Server` client:`Study Review UI`
Flow of events	1. `Server` is receiving a request from `Study Review UI` on task retrieval for a `User` named "bob" which is an invalid `User`. 2. `Server` validates `User` authentication. 3. The given `User` is not valid because it does not exist in the database. 4. A response is sent to the `Study Review UI` detailing why bob does not have access to the `AutoSys System`.

Table 5: Scenario when a invalid user is trying to get access to the server.

5 Use Case

5.1 Use Case Diagram



5.2 Use Case Description

Use case name	`ExportProtocol`
Participating actors	Initiated by `Client` Communicates with `Server`
Flow of events	1. The `Client` sends a request for a specific study. 2. The `Server` validates the request sends back data detailing the study. This include which protocol the Study use and which phases it has been through. 3. The `Client` requests a research protocol from the study. 4. The `Server` receives the request and makes a validation of the `Client`'s user account. 5. The `Server` exports the requested research protocol to the `Client`. 6. The `Client` receives the research protocol.
Entry condition	• The `Client` is logged in and connected to the internet and click on the export protocol menu (provided by the Blue Team)
Exit conditions	• The `Client` has received the selected data from the `Server`
Quality requirements	• The `Client`'s request is received by the `Server` within 10 seconds. • The `Client` starts receiving the study report no later than 40 seconds after it has been requested.

Table 6: Use case: ExportProtocol

Use case name	`RetrieveStudyInformation` using: `ManageStudy`
Participating actors	Initiated by `Client` Communicates with `Server`
Flow of events	1. The `Client` activates the Manage Study use case. 2. The `Client` specifies the study information to retrieve. 3. The `Client` confirms the retrieval of details regarding the Study. An error message is displayed if the provided data is insufficient 4. The message is sent to the server where the senders credentials are validated in the database. 5. The `Server` sends back the study information 6. The `Client` receives the requested study information.
Entry condition	• The `Client` is logged in and connected to the internet.
Exit conditions	• The `Client` has received the study information.
Quality requirements	• The `Client` should receive the study information maximum 30 seconds after requesting them.

Table 7: Use case: RetrieveStudyInformation

Use case name	`StoreTaskData`
Participating actors	Initiated by `Client` Communicates with `Server`
Flow of events	1. The `Client` selects a specific study. 2. The `Client` chooses to upload a completed task. 3. The `Server` receives the request and makes a validation of the user account. The validation goes through because the user uploading the completed task is the same as the one who received it in the first place. 4. The `Server` stores the data from the completed task in the database. Then it sends back a respond to the `Client` letting it know, that the request was received. 5. The `Client` receives the respond from the `Server`.
Entry condition	• The `Client` is logged in.
Exit conditions	• The `Client` has received a response from the `Server` indicating that the completed task was received and stored in the database.

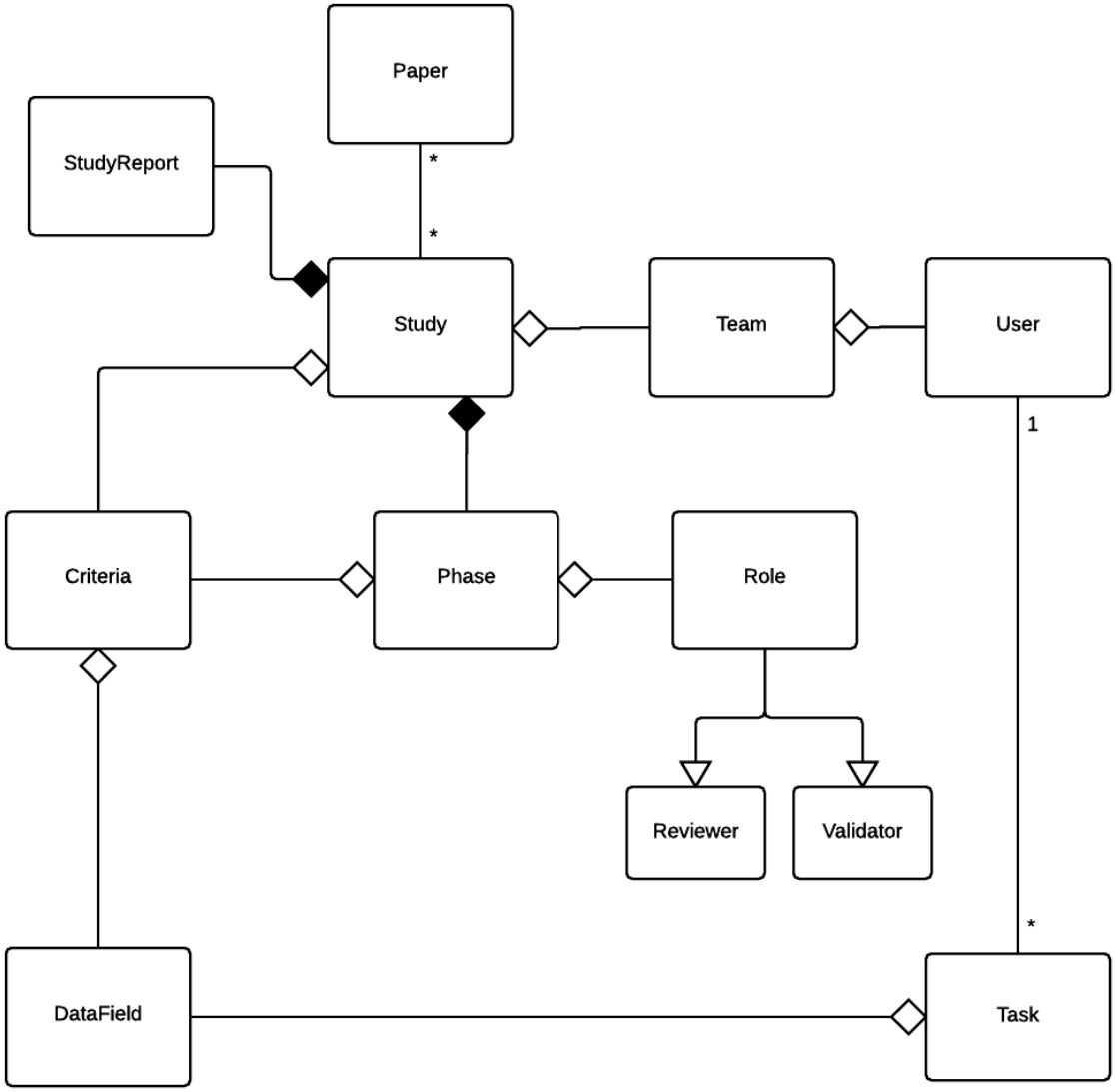
Table 8: Use case: StoreTaskData

Use case name	`ManageStudy` using: `ManageClassificationCriteria`
Participating actors	Initiated by `Client` Communicates with `Server`
Flow of events	1. The `Client` navigates to the `Server` page. 2. The `Client` invokes the `ManageRole` use case to define `Role` for each `User` that are appart of the `StudyConfiguration`. 3. The `Client` invokes the `ManageClassificationCriteria` use case to define classification criterias for the `StudyConfiguration`. 4. The `Client` invokes the `ManageInclusionAndExclusionCriteria` use case to define what to exclude or include to the `StudyConfiguration`. 5. The `Client` invokes the `ManageTask` use case to specify which `Task` a `User` should have. 6. The `Client` reviews all changes made and submits the `StudyConfiguration` to the `Server`. 7. The `Server` invokes the `ValidateUser` use case to validate access rights for the `User` of the `Request` 8. The `Server` accepts the `User` and the `Request` proceeds. 9. The `Server` invoke the `StoreUserData` use case to update the `Role` for each `User` which was specified in the `StudyConfigurationUI`. The `Server` will also invoke the `StoreStudyData`, `StoreTaskData` use cases to save the study and the tasks for each `User`. 10. The `Server` returns an overview of the `StudyConfiguration` to the client.
Entry condition	• The `Client` wants to create a new study.
Exit conditions	• The `Client` has received a `StudyConfiguration` overview
Quality requirements	• The `Client`'s study is created within 10 sec, after the server has received the request.

Table 9: Use case when a `Client` wants to create a new study through the `StudyConfigurationUI`

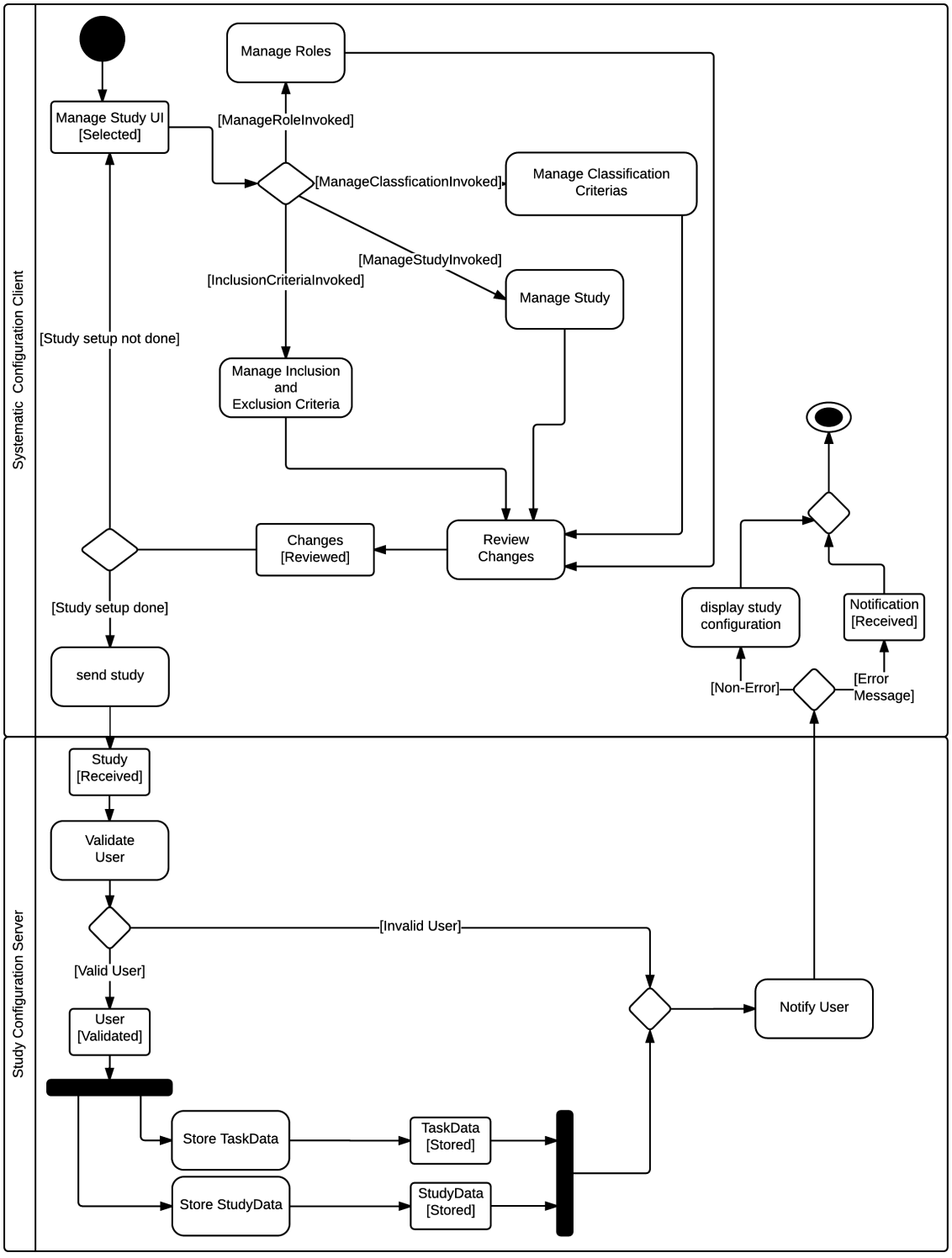
6 Object Model

6.1 Entity Object Model

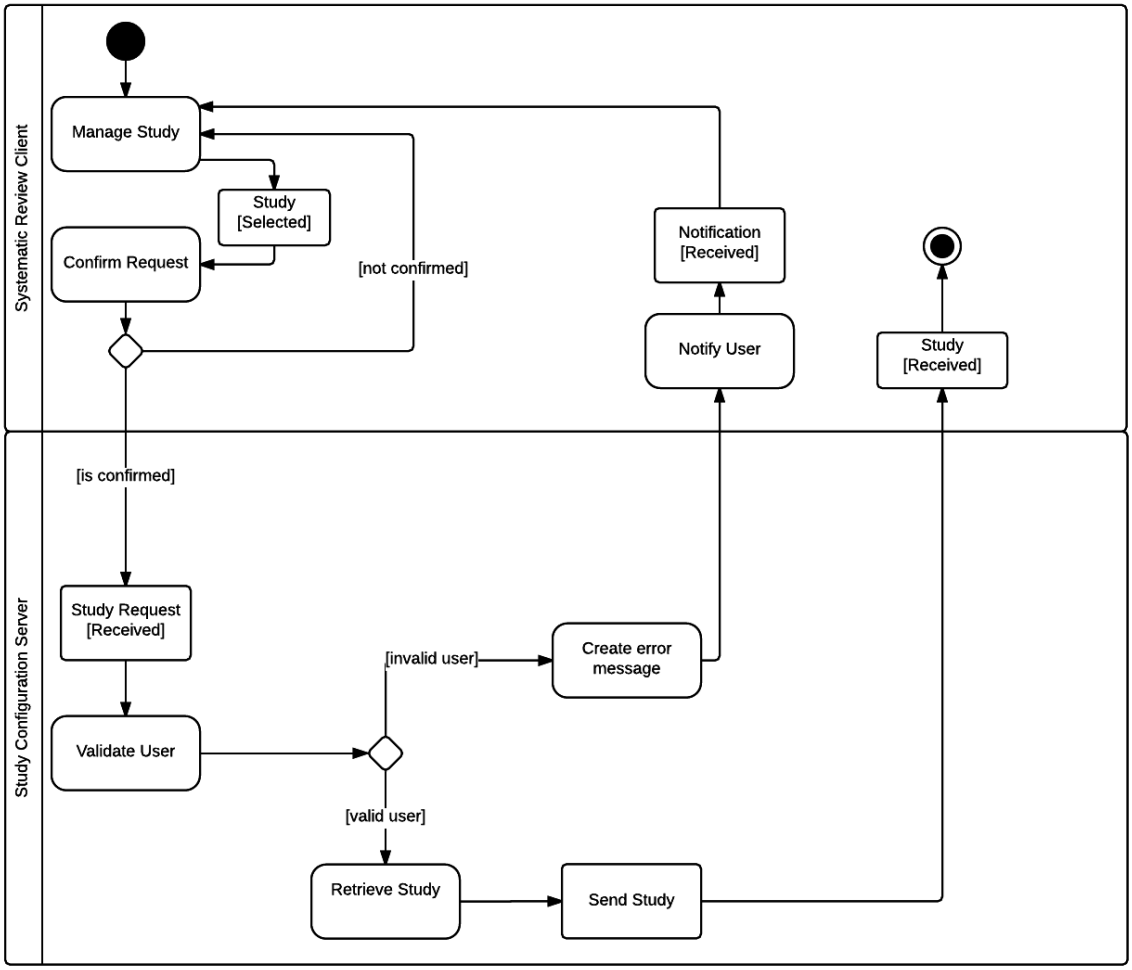


7 Dynamic Model

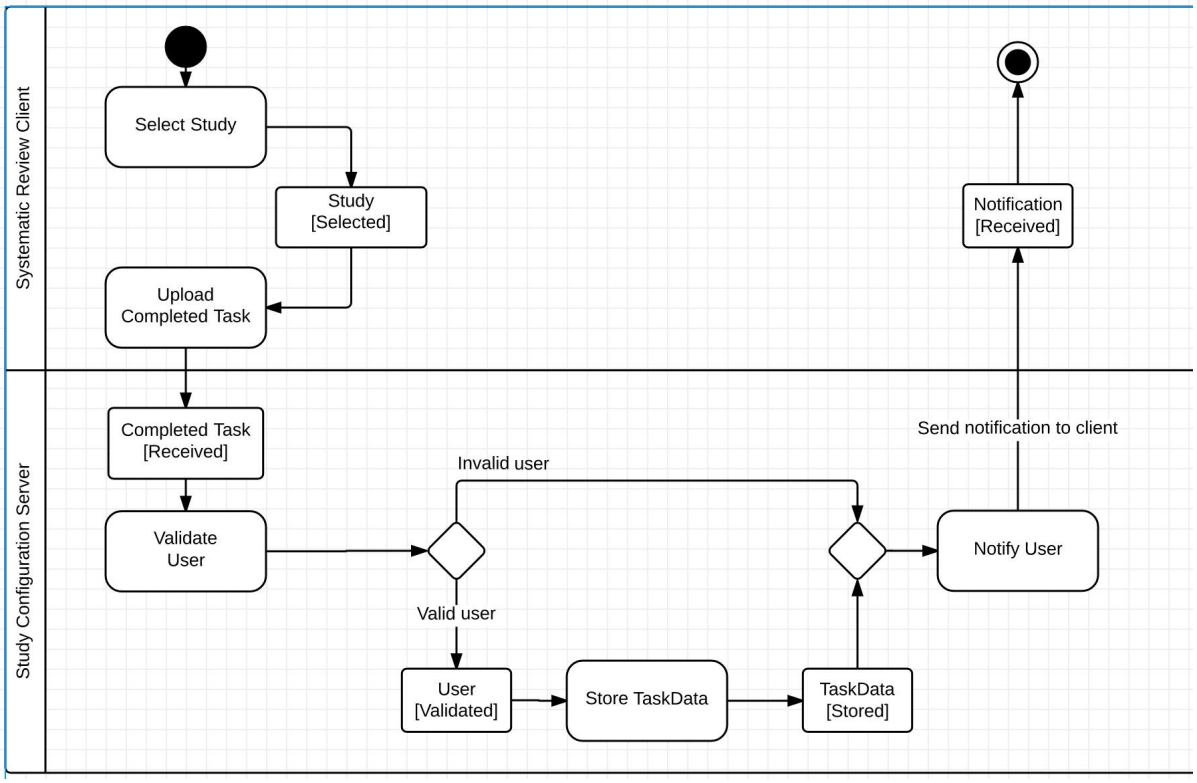
7.1 Activity Diagram of ManageStudy Use Case



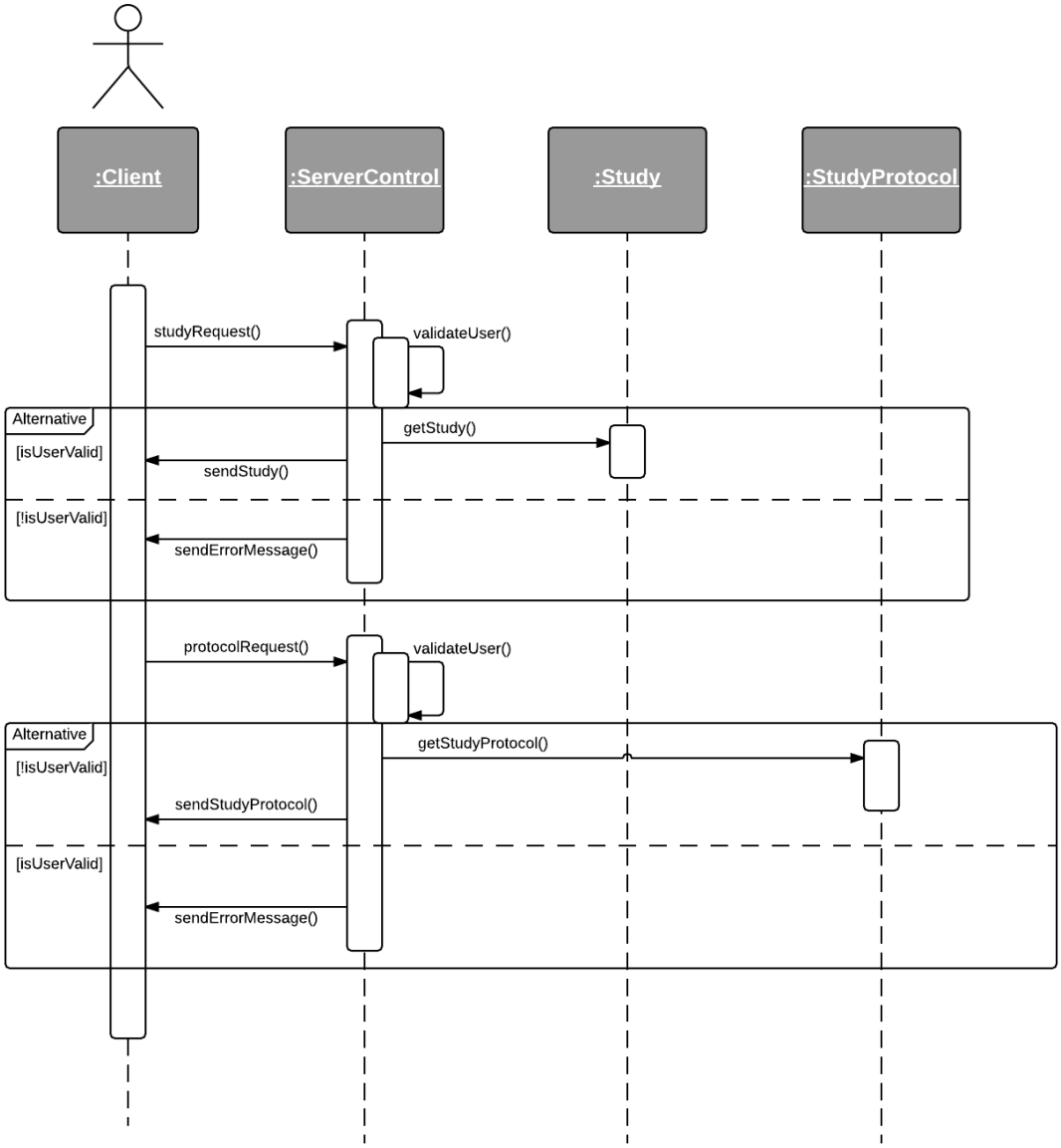
7.2 Activity Diagram of Retrieve Study Information Use Case



7.3 Activity Diagram of StoreTaskData Use Case



7.4 Sequence Diagram of ExportProtocol Use Case



8 Glossary

- Systematic Review Client: the client part of the system providing a user interface for requesting data about papers, reviewing papers, and validating papers.
- Study Configuration Server: the server (including the database) where all data is stored, and user requests regarding papers are handled.
- Study Configuration UI: the user interface supporting work tasks regarding the configuration of studies, e.g. study tasks or study phases.
- Request: all operations that invoke server functionalities.

2 System Design Document

System Design Document - Yellow Project

William Diedrichsen Marstrand, Dennis Thinh Tan Nguyen,
Jacob Mullit MÃ¸iniche, Thor Valentin Olesen

December 3, 2015

Contents

1 Introduction

1.1 Purpose of the System

In **AUTOSYS**, the main client is the **SystemReviewClient** who needs to set up a **STUDY** consisting of one or several **PHASES**. A **PHASE** will consist of one or more **TASKS** which need to be completed before moving on to the next **PHASE**. The **StudyParticipants** can work in different **ROLES** which we anticipate will be either as **REVIEWER** or **VALIDATOR**. Also the **StudyParticipants** will be working together on a **STUDY** in **TEAMS**, but multiple teams will not be working on the same **STUDY**. Furthermore **TEAMS** working on the same **STUDY** will be considered as one big **TEAM**.

1.2 Design Goals

- **Ease of use**:
  Goal: It should be relatively easy to setup a study configuration because it makes up the foundation that all study work processes rely on.

  The end user may have a low level of computer expertise potentially resulting in the wrong setup of a study. This can happen because the user cannot find or access the resources required for setting up a study or they become frustrated if they have to go through many windows. However, these usability traits should not compromise with the system functionalities. The system must still be sufficiently complex in order to provide a variety of ways to setup the study configuration. A primary focus on usability traits has been chosen due to the scope and time span of the project. Functionalities might be sacrificed to achieve this. This design goal is a refinement of the non-functional requirement usability in Requirement Analysis Document (section 3.3.1).

- **High Reliability**:
  Goal: The server and study configuration ui must be reliable, handle system failures and wrong user input.

  The server should handle system failures (e.g. exceptions or network failures). Due to the client-server architecture, it is important that the server can automatically reboot upon system failures. As a result, other clients can continuously access the server. Also, the server should ensure that incomplete data transfers are resumed in case of network failures. Further, the Study Configuration UI should handle invalid user input and ensure the input is only sent to the server when correct. These decisions ensure that the work of the end user is handled properly and the server is available when needed. Speed might be sacrificed to achieve this. This design goal is a refinement of the non-functional requirement "high reliability" in Requirement Analysis Document (section 3.3.2).

- **Scalability**:
  Goal: The response time of the system must not degrade dramatically with the number of users and concurrent studies.

  The system is used by multiple teams with several users working on different studies. The work carried out by these users could be done concurrently and so the system will have a big workload when multiple requests are sent to the server. Since the users need study data quickly to conduct their research, it will be cumbersome if the users have to wait for a long time to get the data. Thus, the server should support quick data access for users. However, the system must do this in a way which takes memory into account to avoid an extensive resource consumption.

- **High performance**:
  Goal: to ensure a quick and responsive user interface by using a database, which facilitates fast data processing.

  Performance is the key to this project. Each and every component of this project speed is determined by how fast the database can respond. The user might feel the UI is slow and sluggish, due to a slow database. Contrary, a slow and sluggish UI can feel more responsive and quick with a fast database. A faster database can furthermore serve more users without them noticing the increased load on the database. Ideally the user should never notice the program communicates with a external database, but just focus on program in front of him

2 Proposed Software Architecture

2.1 Subsystem Decomposition

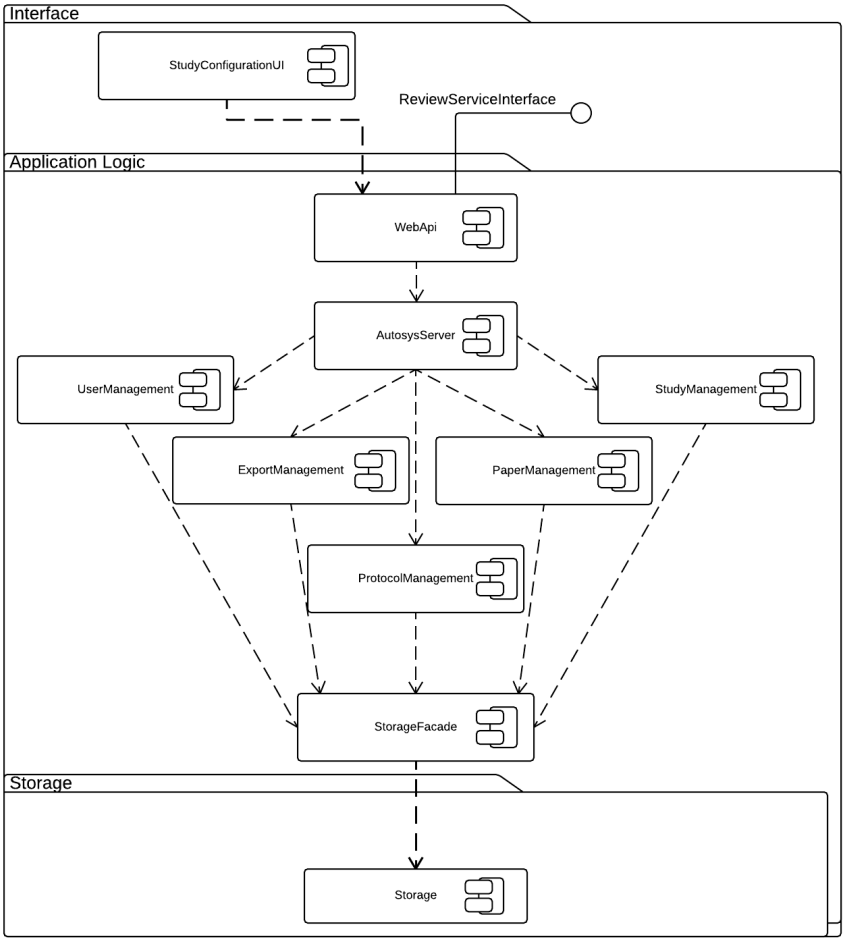


Figure 1: Autosys subsystem decomposition (UML Component Diagram, layers shown as UML packages)

The figure above shows a three-tier architectural style has been applied in the decomposition of the system. This architecture has been chosen instead of a two-tier client-server approach. Consequently, we avoid closely combining the application data and application logic on the server. The three-tier style results in a division into three independent tiers; Interface, Application Logic, and Storage. This makes it possible to update or change the individual tiers without affecting the whole application. In this way, the scaling and management of the system becomes more flexible as opposed to a client/server architectural style. Finally, it is worth noting that still allows for high security and easier administration of the data through the centralized data access.
From the functional requirements of the Autosys RAD document, the following needed services are defined by grouping related operations that share a common purpose:

- Study Configuration
- Task Handling
- Paper Screening
- Study Export
- User Validation
- User Management
- Research Protocol Generation

The **StudyConfigurationClient** provides the service related to configuring a study. It is a front end for users used to initiate all use cases related to setting up or configuring a **Study**. The **ReviewService** provides an interface for different Review Clients to communicate with the **AutosysServer**, such that the system is not dependent on the review client - ultimately resulting in a more flexible design. The **WebApi** will be providing the user validation services for allowing user access to all other Management components in the application logic (see Figure: 1). From the **WebApi** a **ReviewServiceInterface** will be provided, which supports the use of an adapter pattern to communicate with different/changing client systems. The **UserManagement** component is providing the user management service, since it holds the responsibility for handling all CRUD operations regarding teams and individual users. The **StudyManagement** component provides the study configuration service by allowing users to set up and manage study information. The **ExportMangement** provides the study export service allowing users to export Studies as plain data sets, e.g. csv files. The **PaperManagement** provides the paper screening service by running filtering mechanisms on the research papers to get relevant papers according to the Study Criteria and Classifications. The **ProtocolManagement** provides the services related to generating a Research Protocol and manages

export of protocols to studies based on their configuration. The **Storage** interface is used to decouple the storage abstraction from the application logic so that the two can vary independently (Bridge Pattern). The **AutosysStorage** represents the subsystem for storing the user data, study data, and research papers. All packages in the UML Diagram 1 symbolize different projects in the programmed system, and all components symbolize directories except the WebApi component which have been made a separate project outside of ApplicationLogics. This was due to the possible complications with compilation of files which could arise when trying to combine both a web application project and a console application in one project.

2.2 Identifying and Storing Persistent Data

Identifying persistent objects

Autosys deals with three sets of objects that must be stored. The first set (referred as metadata storage) consists of research data such as metadata on articles which are accessed during conduction of secondary studies. The second set (referred as blue storage) consists of objects that are created and accessed by the blue part of the system(eg. Users, roles, etc). It needs to be persistent to track the progress of the study and who is involved. The third set (referred as configuration storage) consists of data for a study configuration that are created by the study configuration UI. The data defines the study configuration such as research questions, tasks, inclusion and exclusion criterias etc.

Metadata storage is well defined and will rarely change during the lifetime of Autosys. These changes may occur whenever new research data has been created or removed and thus create a need for an update of the set. Objects within blue storage are managed and defined by the blue part of the system. Hence, we decide to let the blue part decide how to manage and access these persistent object through a generic interface. Configuration storage is also well defined and will not change once the study configuration has been made.

In this scope of the Autosys system, the main focus of persistent objects is set on metadata storage and configuration storage.

Selecting a storage strategy

By selecting and defining a persistent storage, strategy enables us to deal with issues related to storage management. The main design goals of the yellow part of Autosys is to be reliable, scalable while also having high performance. It is therefore decided to implement a database management system since it allows concurrent queries and provide transaction mechanisms to ensure data integrity.

Compared to a flat file storage method, a database management system will also scale to large installations with many researchers that are to conduct studies simultaneously. However, to allow future upgrades or changes it has been

decided that the storage strategy is not solely dependent on a database management system. Thus, the storage subsystem will provide an abstract interface that enables other kinds of storage to be coupled. The users of Autosys are not able to change the storage, however if an update is ever needed for the storage strategy, a system administrator are able to do so.

2.3 Providing Access Control

This section aims to show how the yellow system handles requests from clients based on which actor they represent. AutoSys is a multi-user system with a manager and researcher actor. A researcher can both be assigned the role of a reviewer and validator since only their tasks are different. We have drawn an access control matrix depicting the allowed operations on the entity objects for both actors. In summary, a Manager can create Users and Teams while Researchers can create Studies and receive Tasks. Ideally, all actors must first be authenticated based on their role before they can modify any object in the system. One would then use access control lists on each object to check the access privileges of the user. Below is the access matrix for main AutoSys objects. Note that the study object is comprised of a StudyReport, ResearchProtocol and a Phase with Criteria. Tasks are generated in a Phase. Also, we have omitted method names for Users and Teams that only have CRUD operations. Finally, we assume that a Manager cannot also be a Researcher.

Actors/Objects	Study	Task	User	Team
Manager	<<create>> <<read>> <<update>> <<delete>>	<<create>> <<read>> <<update>> <<delete>>	<<create>> <<read>> <<update>> <<delete>>	<<create>> <<read>> <<update>> <<delete>>
Researcher	GetStudies GetStudyOverview GetResource	GetTasks DeliverTask GetReviewableTaskIDs GetReviewableTasks	<<read>> <<update>>	<<read>>

Figure 2: Access matrix for main AutoSys objects

2.4 Designing the Global Control Flow

When selecting components for the interface and storage subsystems of Autosys, we effectively narrowed down the alternatives for control flow mechanisms for the yellow part of Autosys. The AutosysServer is located on a web server. The AutosysServer awaits requests from the blue part of autosys or the studyConfigurationUI. For each request the AutosysServer receives, a new thread is made, which enable it to parallel handle the requests. This results in a more responsive system. By example, the AutosysServer can process and handle a given process x while another process awaits a respond from the database. However, the stumbling block of threads is the increased complexity of the system resulting from the usage of threads. To establish a sturdy design with concurrency taken into consideration, one will define the following strategy for dealing with concurrent accesses to the shared storage:

- *Boundary objects should not define any fields.* Boundary objects should only hold temporary data correlated with the current request in a local

variable.

- *Entity objects should not provide direct access to their fields.* All changes and accesses to a given object state should be done through dedicated methods.
- *Methods for accessing state in entity objects should be syncronized.* By using thread synchronization mechanism provided by C#, only one thread can be active at a time in an access method.
- *Nested calls to synchronized methods should be avoided.* When creating synchronized methods, one must make sure if a nested method call can lead into calling another synchronized method. The reason for this is to avoid deadlocks and must be avoided. If nested calls are unavoidable, one should either relocate the logic among methods to or impose a strict ordering of synchronized method calls.
- *Redundant state should be time-stamped* The state of an object can periodically, be duplicated. By example, two researchers may create objects with the same state and can lead to conflicts. To avoid this, objects should be time-stamped or have another unique identifier.

2.5 Hardware/Software mapping

Systems
This program is inherently a documentation tool which map research articles goals and conclusions in a searchable context. This requires the program to be stable and reliable, especially in the context of data preservation.
The program is split in two components, a client and a external database server. The user will run a client on his computer which will communicate with the server which runs the application logic and storage. The user client will feature data preservation, to enable offline functionality, but only to a limited extend. This report primarily focus on the server, if any questions to the user client should occur, please refer to the according Blue SDD.

All user clients will contact the a single server. To solve this, the server will be multi threaded, which allows for multiple users interacting with the server at the same time, but At the current scope of the program, this is possible due to low amount of users, but should this system expand a new design should be conceived.

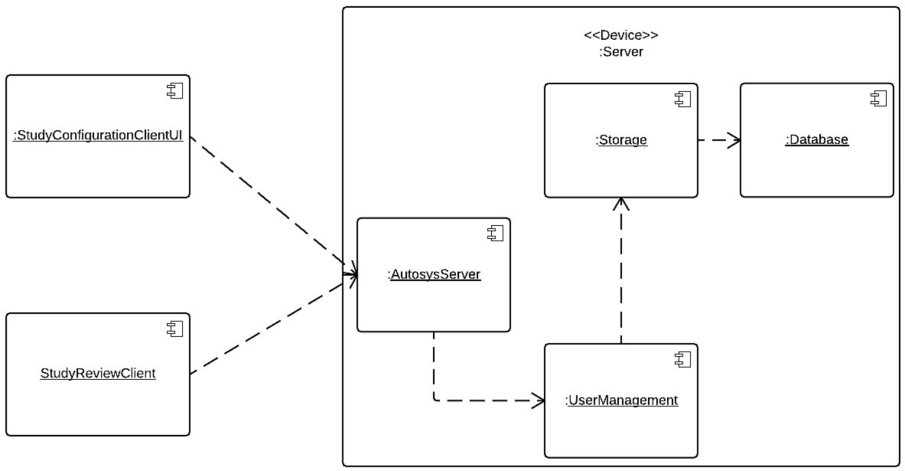


Figure 3: Software mapping of prior subsystem decomposition. Note that the components UserManagement, PaperManagement and StudyManagement has been collapsed into a single unit called UserManagement

Program components
The server will be implemented in C Sharp as requested by the the client. By using C Sharp, we can utilise to Microsoft expansive database systems and interfaces especially in regards to the database. Additionally this also make the program easier to maintain, since C Sharp is a well known language and can utilise the .NET framework. As communication between the server and the clients, we plan to communicate with HTTPS request containing JSON object. This will make it easier to implement future changes and even completely replace the clients user interface with a more modern solution like a web page

To achieve fast database responses, we will implement an entity framework database, which utilizes Microsoft's .NET framework to optimize database queries. This takes advantage of optimized queries and features implemented by Microsoft database experts. Furthermore, a future feature may be implemented to cache the result of large queries for quick access. This allows the server to respond hastily to large queries, which have previously been used. This will only be implemented for queries on data that is less likely to change. As a result, a memory trade-off may occur, which can be negated by using limits on how much memory and how many queries are cached. By way of example, the last x number of query results to the server could be cached for repeated use. The design goal is a refinement of the non-functional requirement "high performance" in the Requirement Analysis Document.

3 Object Design Document

Object Design Document - Yellow Project

William Diedrichsen Marstrand, Dennis Thinh Tan Nguyen,
Jacob Mullit Møiniche, Thor Valentin Olesen

16. december 2015

Indhold

1 Introduction

This object design document has been written to provide the reader with an insight to our design choices throughout the project. To support this, class diagrams will be used to highlight where design patterns have been used. Finally, this is related back to the SOLID principles. The class diagrams are used to showcase our familarity with the techniques elaborated in OOSE throughout the course. Note that this document only focuses on important classes and provide a complete detailed description only of those. The purpose of the UML techniques is to better convey our design choices and final code structure.

1.1 Interface documentation guidelines

To improve communication and understanding between the developers, the group has defined a set of documentation guidelines that must be followed strictly in order to have a consistent codebase structure and design.

Base Guidelines

- Every class are named with singular nouns.
- Every class and methods are to be documented before they are implemented. The documentation must be precise in regarding for what the given class or method is responsible for. This will enable developers to be more effective in their work because they do not have to spend time on analyzing poorly written documentation or the implementation itself.
- Every method must be defined in a way that they only have one responsibility. Long methods should be refactored into helping methods or moved to other appropriate sections of the system.
- Every non-trivial parameters and returns are to be documented.
- Every Method signatures must be written with camel-case and they must follow C# formation standards. This improves readability for other c# developers that are to read the code.
- Error status must be returned as exceptions and propagated to appropriate layers. One should avoid catching general exception since this may mask problems that have nothing to do with the system. Instead of just catching a concrete 'Exception', one must catch specific exceptions that may be relevant to process such as "OutOfMemoryException"that would preferably be propagated to the user.
- One should use generic classes where to seem to be fit. For an example, one should utilize an interface such as IList¡object¿ instead of a concrete implementation such as 'List¡object¿'. This makes it easier to refactor the code since it is not tightly coupled to some concrete class.

- Dependency injection must be considered when creating new classes and methods. System modularity will make it easier to extend or refactor the system.

It should be noted that the group had decided to follow these base guidelines as much as possible. However, it should be considered that exceptions may occur and some rules may be broken. These exceptions must be brought up into descussion or reconsidered before proceeding with the given activiety.

2 Design Decisions

3 Object design choices

3.1 Facade Pattern: MainHandler Facade

Multiple handler classes were made with their own well defined area of responsibility in an attempt to uphold the Single Responsibility principle. This resulted in several handlers and thus a complex system of handlers which we considered a problem, since it made the understanding of the system more complicated than if the functionality was gathered one place. In this case a Facade Pattern seemed useful, since it would make it possible to wrap the complex set of handlers and offer a simple way to access the required handler functionality using a MainHandler Facade as a facilitator. The use of the facade pattern in this case is not strictly as the general guidelines describe, since no interface is provided for the handlers to implement, but the principle of having one facilitating class providing the functionality of the subsystem classes is respected. Another important point was the way a MainHandler facade could decouple the subsystem of handlers from the using classes. Even though the limited functionality provided through the facade would mean some what less flexibility than if all handlers were used, the conclusion was made that based on the just mentioned pros, the use of a facade would be primarily beneficial.

The repository design pattern has been used as a layer in the code to separate the business code in our Application Logic and the database. The business logic accesses data from data stored in the Storage package where the AutoSysDbModel represents a local database in SQL Server 2014. This has been done to isolate the data layer to support unit testing and support swapping out the database technology. The code maintainability and readability is increased by separating business logic from data or service access logic. The entity-related repositories in the Storage pack separate the logic that retrieves the data and maps it to the stored entity model from the business logic by using the AutoMapper framework. The repositories are used to perform basic CRUD operations on entities that are required for permanent storage in the database. The repositories can query the data source layer (database, either local production database in SQL Server 2014 or in-memory database using fake connection in the Effort Framework for integration tests). The repositories do not map the

data from the stored model entities in the Storage package but AutoMapper is used to map the data source to a business logic entity in the StorageAdapter classes in the ApplicationLogic. These adapter classes are likewise used to store persist changes in the business entity to an Entity Framework database. Thus, the repository separates our business logic from the interactions with the underlying database.

4 Repository Pattern

4.1 Benefits

Using the Repository design pattern has centralised our data logic and helped create a flexible architecture that can be adapted as the overall design of the application evolves. Without the repositories we would have to access data sprinkled throughout our codebase. By way of example, updating a table would require to find every spot in our code that relies on it and update all of them. With the repository pattern we only need to update a single class and the rest of the code is unaffected. This corresponds well to the Single Responsibility Principle stating that each thing should have a single reason for change. The repository helps us isolate changes and limit the scope of an entity model to a single level of abstraction. Ultimately, increasing readability and maintainability.

4.2 Cons

The only downside is the fact that our lacking knowledge on the domain objects has forced us to refactor the data representation of all stored model entities numerous times. As opposed to having a clear understanding of the business logic and mapping these objects correctly to the permanent storage from the start. Also, we struggled choosing whether the repositories should be async or not. Consequently, we chose async assuming the program could possibly run with a hosted database. In that case, async maks it possible to use CPU resources that are otherwise not usd while threads are waiting for an answer from database queries. Finally, we did not have a clear opinion on when to dispose the DbContext used to interact with our database. Initially, we disposed the context for each database method call in the repositories. Instead, we now implement IDisposable throughout all interfaces between the WebApi and the Storage layer to dispose after each client request is invoked and completed instead.

4.3 Test Reflection

All the functionalities have been tested by mocking an interface of the AutoSysDbModel instead of using a test stub. We should have started on the integration tests earlier to validate that our repositories and the stored model entities and their relationships are actually compatible with a database. Initially, we tested if the logic in the repositories are called but not if the entities were actually

stored correctly. For this purpose, we used the Effort framework to create an in-memory database on a fake connection and validate the CRUD operations on a real database while checking if the entities were properly interrelated with the foreign key and data attribute functionalities supported by Entity Framework.

4.4 Conclusion

We have performed numerous design choices in the actual implementation of the repository pattern. Mostly, the challenges considered have been whether we should use a generic repository or several concrete entity-related repositories and mapping the entities. We have chosen latter to achieve Separation of concerns and reduce unnecessary unit testing complexity. As a result, we end up with more repository classes but also a significant readability and maintainability improvement and easier unit testing. The repository methods only need to change if the flow of data access changes and it is not impacted by the code in the business logic. Thus, working with a repository built on SOLID adds a lot more speed to future addition than without proper separation.

4 Testing Strategy

Testing Strategy

Dennis Thinh Tan Nguyen, Jacob Mullit Møiniche,
Thor Valentin Aarkjr Olesen, William Diedrichsen Marstrand

16. december 2015

Indhold

1 Introduction

This document will describe our reflection of the testing phase, our current implementation of our tests and the test we plan to incorporate in the future. We hope, by including tests in our early project stages will help us to better our design and catch errors which otherwise would have slipped past us. We firmly believes bugs and errors are easier to correct if the are caught early, and it will prevent them from snowballing and pick up size and scale.
We do not believe we are able to catch and eliminate all bugs and errors, but we have faith in a strong presence of tests will strengthen our products robustness

2 Unit Testing

We are mainly using unit testing to make it easier to detect and corrects faults, and to allow parallelism in the testing activities which is essential in our work, as every group member works on individual parts of the program simultaneously. We will incorporate Test Driven development, due to it helps us focus on testing, even from the earlier phases.

Black Box By testing all classes as they are developed, we make sure that all units are tested. The unit tests are automated which makes it easy to do regression testing, and in that way always have an idea of the robustness and quality of the program. We use black box testing to conduct tests of units before they are actually implemented. The reason for this is, that we know the functionality which the unit should support and as such we can create the tests based on this knowledge and check that our code does what is expected of it, by running the tests after the implementation. More specifically we use **equivalence testing** from black box testing to reduce the number of test cases. This has a huge impact in the cases where inputs and outputs are tested. In the equivalence testing we have give boundary testing high priority since it is often edge cases which make systems fail.

White Box After a feature is fully implemented it is then marked as ready for white box testing. The white box testing should be conducted by a team member who did not work on the implementation. This approach has been chosen to let other team members get knowledge of the code which they themselves did not work on. Furthermore, a tester who have no prior expectation to how the code should work might find errors the original writer would have fought of.

The goal for the white box testing is not to get path coverage, since this would be to comprehensive and not practically possible because of the many possible paths. Instead the goal is set to 90-95% branch coverage on essential components. The reason why this percentage is chosen and not 100% is that some cases will always be hard to cover, and thus this would require to many resources compared to the payoff from the tests.
The main goal is to test all major parts and all essential functionality of the program and a percentage of 90-95% would very likely achieve this.

Mocking To make sure that we can unit test classes that make use of the database without affecting the database we use mocking. By mocking the database we can simulate some of the database's functionality, and run the methods depending on it without ever calling the database. This means that the tests do not have any impact on the database, and as such we make sure that the database does not break when tests are run.

3 Integration testing

Unit Integration testing is a as the name apply a tests which components are working correctly together. It assume the the individual components are working correctly and is strictly testing for compatibility errors. This test approach allows us to test interconnectivity of classes instead of viewing them as isolated systems, and as a result gives us a broader image of which systems are effected of defect sub components.

3.1 Different types of integration testing

How to use integration test depends on your test methodology (Top down or Bottoms test strategy).

- **Top down test strategy** : implement integration testing by first testing the high level components integration and from there working downwards to the lower level components. This allows high level component and subsystems to be tested early in the development but comes with cost of delaying some test of the more complicated integrations test due to the lower level components have not been implemented yet.
- **Bottoms up test strategy** : implement integration testing by first integration test low level components and from there iterative integration test each ascending components and modules integration in the system. This ensures the programs module are integration tested early in the test phase, but post pone all high level components integration tests to late in the project

3.2 Design choice

Given the two different test options we went with a bottoms up test approach. This is due to how we have choose to implement the system which is a iterative bottoms up approach. Since we are striving for a Test Driven Development then the lower level components are always tested first. While we are black box testing the subsystem it becomes convenient to create a integration test while we are testing the system. This will lead to the more complicated components will be tested last which will be a challenge, but hopefully our extensive work with the lower level components will ease the challenge of testing the most complicated subsystems

4 System Testing

4.1 systemtesting

This system testing strategy guide has been written to ensure that the complete system atleast complies with the functional and nonfunctional requirements described in the RAD document.

4.2 Product, Revision and Overview

AutoSys is a dedicated provisioning service that provisions study related data to the client, based on task requests defined in the Study Configuration. The AutoSys Configuration Server is used to define the configuration of a given study for a team of researchers. The server stores study data and provides the client with the appropriate requested data. Also, the server delivers review tasks for the client used to conduct studies. Thus, the purpose of the system is to provide a tool support for conducting a study by a team of researchers
See RAD for more information.

Tabel 1: Planned releases

Version	**Release Date**	**Changes**
Initial Release (v1)	2015-12-10	
Revision 1	2015-12-18	

4.3 Features to be tested

The following features will be tested as a minimum to verify that the system conforms to the most important functional requirements:

- Create a team with a manager
- Create a Study Configuration
  - Assign a team
  - Define a phase
  - Create inclusion, exclusion and classification criteria
- Filter research papers using comparison operators
- Export papers in csv
- Handle multiple concurrent client requests
- Generate a research protocol

The following features will be tested as a minimum to verify that the system conforms to the most important nonfunctional requirements:

- The user must be able to configure a study using at most three application windows
- The system should restart within 30 seconds upon failures.
- The system should handle a client task request within 15 seconds.

4.4 Features not to be tested

- Platform compatibility on devices running other OS than Windows 10
- Database compatibility level Import and export of pdf files
- All features in the blue system

4.5 Environment Configuration

Tabel 2: Platform environment

	Macbook Pro (15-inch, 2015)	**Dell XPS 13**
Operating system	Windows 10 (64-bit) Bootcamp	Windows 10 (64-bit)
Memory	16 GB	8 GB
Hard disk	500 + GB, SSD	500 + GB, SSD
Processor	2.5GHz Intel Core i7-4870HQ	Intel Core i7-6500U
Graphics card	NVIDIA GeForce GT 750M	Intel HD Graphics 5200
Screen	15.4-inch 2,880 x 1,800 Retina screen	13.3? QHD+ (3200 x 1800)
Browser	Chrome	Edge
Network adapter	802.11ac wireless	100 MB Ethernet

4.6 System test methodology.

The system will primarily be tested from the functional requirements (from RAD) and the nonfunctional requirements (SDD). However, other activities such as Pilot testing, Acceptance testing or Installation testing might be used to further verify the system requirements. By way of example, we could select a group of potential users to test the common functionality described previously. The functional tests will be carried out by finding test cases from the use case models in the RAD document. The goal is to find differences between the functional requirements and the system.

4.7 Initial Test requirements

Test strategy (this document), written by test personnel, reviewed by product team, agreed to by project manager

4.8 System test entry and exit criteria

Entry Criteria
The software must meet the criteria below before the product can start system test.

- All basic functionality must work.
- All unit tests run without error.
- The code is frozen and contains complete functionality.
- The source code is checked into the Git repository.
- All code compiles and builds on the appropriate platforms.
- All known problems posted to Git.

Exit Criteria
The software must meet the criteria below before the product can exit from system test.

- All system tests executed (not passed, just executed).
- Results of executed tests must be discussed with product management team.
- Successful generation of executable images for all appropriate platforms.
- Code is completely frozen.
- Documentation review is complete.
- There are 0 showstopper bugs.
- There are fewer than $< x >$ major bugs, and $< y >$ minor bugs.

Test Deliverables

- Automated tests in $< framework >$
- Test Strategy and SQA (Software Quality Assurance) project plan
- Test procedure
- Test logs
- Bug-tracking system report of all issues raised during SQA process
- Test Coverage measurement

4.9 Test Cases

- Add test case for ExportProtocol use case
- Add test case for RetrieveStudyInformation use case
- Add test case for ManageStudy use case

5 Usability Testing

To get a better understanding on the user of the Autosys system and their usage of the "Study Configuration UI", one must perform usability tests to get their feedback. It is worth to mention the design goal: Ease of use where a study con figuration should be relatively easy to setup since it makes up the foundation that all study work processes rely on. (See System Design Document, Section 1.2 Design Goals).

At the initial design of the user interface various paper mocks up will be created and used to conduct scenario tests combined with think-aloud tests. The user will perform pseudo actions on the mocks while explaining the testing team what they are doing and why they are doing it. This enables the testing team to notice whether a given mock is living up to given usability tasks and if there are any usability defects that must be addressed. This will be conducted as an iterative process until some final mockups have been created.

A final mock will be chosen and will then be implemented. In some given point of time, a prototype will be created and used for conducting prototype tests. For this project, a horizontal user interface prototype will be used. This enables the test team to have a notion on how users interact with an actual interface and can thus address issues that may occur. The test team may want to test if the system is fit for use, ease of learning and task efficiency. Fit for use because it is important to check whether the user can perform all of his tasks. Ease of learning is to check how well the user is able to interact with the interface correctly and task efficiency is to check how well the user is performing his tasks. All of these usability measures are then taken into consideration and used to optimize the user interface.

When the system has most of its functionalities implemented, a product test will be conducted to test if the implemented functionalities are working as requested.

5 Project Management

Project Management

Thor Valentin Aakjr Olesen, Jacob Mullit Mniche,
Dennis Thinh Tan Nguyen, William Diedrichsen Marstrand

December 16, 2015

1 Group Collaboration

1.1 Organising the Project

This is a short description of the various group structures we have used to organise our group throughout the BDSA course at ITU. To establish a common ground and similar expectations for the project, we reused a group contract from prior work. The contract provided a relaxed and open minded work environment. However, we still felt a need for a clearer distinction between group work and private life, which lead to an official definition of meeting hours. In other words, each group member specified in which time span he is available and vice versa. This allows us to avoid trespassing on group member's spare time.
Initially, we kept the meetings informal without a Mediator, but with a Note Taker. A Mediator would only get elected upon sudden conflicts that required a more planned approach.
Even though consensus was reached on the group contract, we still lacked a dedicated strategy for the weekly assignments. Tasks were randomly delegated to group members without any direct understanding of the actual required workload of the task. Consequently, an unbalanced work distribution sometimes occurred among team members, which ultimately led to otherwise avoidable group conflicts.
To solve this problem, we iteratively improved our workflow by experimenting with different techniques, e.g. Scrum. We could not implement a pure Scrum implementation due to limitations like time constraints, but features like the task board proved especially valuable. The task board helped us visualise the current tasks and quality control already solved tasks. On the task board each task starts in the **Backlog** area. Then we move the most vital tasks to the **Current Sprint** area. From here each group member is assigned to a task. When a task is completed, it is moved to the **Review** area. Then it is evaluated by other group members. The task is either approved and moved to the textbfDone area or failed and moved back to the **Current Sprint** area. As a result, all tasks are reviewed and continuously kept track off.

1.2 Division of Tasks

Whenever new tasks were brought up at group meetings, they were initially divided into parts and subcomponents that could be iteratively delegated to each member of the group. By way of example, the requirement analysis document sections were marked as tasks, which could either be worked on individually or in groups. The group members have chosen tasks independent from time consumption and scope as one does not know this until having worked with the given task.

As a baseline, each member was expected to work on a given task single-handedly. However, if a distributed task was exceedingly great, it was either divided in more subtasks or assigned to additional members. We did try to analyze if a given task had a tremendous impact on the whole project and was thus regarded as a task that demanded everyone's attention.. By way of example, when working on the Design Goals in the system design document, the whole group was required to work on this section before they could proceed with the document.

The distribution of work/tasks for each part of the project can be seen on figure 1, figur 2 and figur 3. Notice how each task is divided into sections that are delegated to each member. Each member had some main tasks that they were responsible for (which can be seen on the high percentage). However, one may also notice that other members do have a smaller percentage on some given tasks. This is because the given task was either reviewed or too demanding. As a result, other members that had finished their originally assigned tasks would place their resources onto other tasks.

RAD	Introduction	Current System	Proposed System	Scenarios	Use Case	Object Model	Dynamic Model	Glossary
Dennis	10%	0%	20%	50%	15%	0%	40%	25%
Jacob	0%	100%	20%	50%	15%	30%	25%	25%
Thor	85%	0%	30%	0%	15%	30%	10%	25%
William	5%	0%	30%	0%	55%	40%	25$	25%

Figure 1: RAD Work Distribution

SDD	System Purpose	Design Goals	Subsystem Decomposition	Persistent Data	Access Control	Global Flow	Hardware
Dennis	0%	25%	5%	95%	5%	95%	0%
Jacob	0%	25%	0%	0%	0%	0%	100%
Thor	0%	25%	10%	5%	90%	5%	0%
William	100%	25%	85%	0%	5%	0%	0%

Figure 2: SDD Work Distribution

Code Skeleton	UserManagement	ExportM	ProtocolM	StorageM	WebAPI	StudyM	PaperM
Dennis	50%	5%	5%	5%	60%	25%	0%
Jacob	0%	0%	0%	0%	0%	50%	50%
Thor	20%	0%	60%	90%	0%	25%	0%
William	30%	95%	35%	5%	40%	0%	50%

Figure 3: Code Skeleton Work Distribution

1.3 Cooperation Tactics and Tools

The group work was coordinated primarily by using tools for planning, version control and communication. We generally applied a loose version of scrum to manage the project and become familiar with the SCRUM methodology. In practice we tried to keep each other updated every time we met (partial stand up) and would try to give each other an overview of three things accordingly; what did we do last, what are we planning to do today and finally whether anything is blocking this purpose. If anything was blocking a team member from continuing his work, the SCRUM facilitator would try to find the required help to solve this. We also established official meeting hours and contact periods to separate study related activities from social life. This was done to cope with the otherwise stressful environment that team members felt due to the heavy work load in this semester. Communication tools such as Facebook and Messenger were used to keep in contact and inform the group about practical information. Finally, the collaboration tool "Trello" was used to keep track of everything. Note that all members are assumed to stay updated about changes on both Facebook, Git and Trello. **Trello** is a collaboration tool we use to organize the project into so called "boards". In one glance, Trello tells you what's being worked on, who's working on what, and where something is in a process. The board represent a combination of the different phases used in SCRUM and the Waterfall Model. We have a Backlog board that corresponds to the Product backlog containing all possible features and requirements in the system. Secondly, the Sprint board contains a backlog with a work that must be addressed during the next sprint (usually one for each week). When team members finish a task it is tested and reviewed by another member in the other boards and finally put in the Done board.

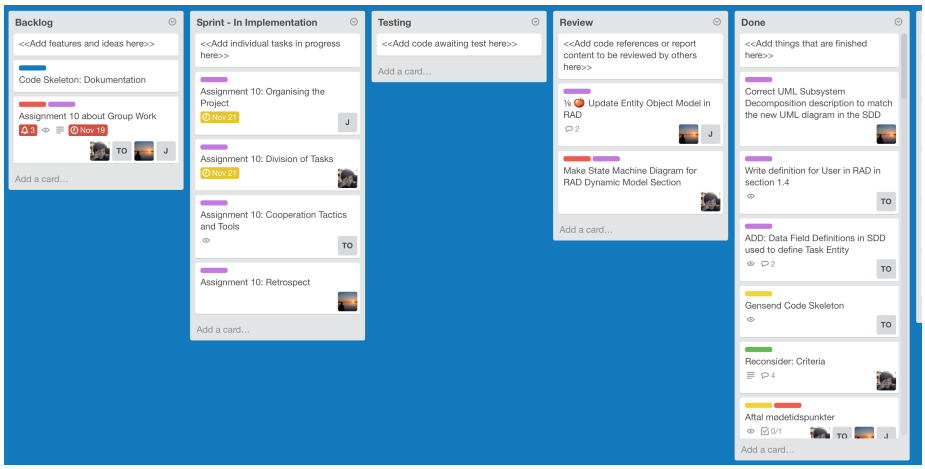


Figure 4: Trello Board

1.4 Retrospect

When thinking of the group work in retrospect, it has become clear that certain things could have been done differently, which would probably have improved the communication, cooperation and efficiency in the group. The first thing which should be mentioned is the attempt to follow the SCRUM method, which only was accomplished partially, since one of the core features (the stand up meetings) was not followed consistently by the group. If these meetings had been upheld, it would presumably have strengthened the communication. As a result, this could potentially have lead to fewer misunderstandings and less miscommunication during the work.
During the beginning of the group work, Trello was not used properly, which meant that the group work did not become as structured as it could have been. By using the Trello board probably it would also have made the first issue with SCRUM easier to handle, since it would have been possible to structure the SCRUM using this tool. Also, the use of a Trello board would have made the planning and distribution of tasks a lot easier.
Better communication could also have been accomplished by the use of a Trello board combined with a more structured use of the Facebook group, e.g. by setting up some guide lines on how and what to write. By scheduling strict deadlines and communicating more about them, some of the unfortunate mistakes with missing content, which happened during hand ins could have been avoided. Further, a better set of rules for VCS when writing the documents in LaTex could have prevented some critical compile errors.
In the beginning, the working hours were very flexible and mostly decided based on people's job schedule. This lead to occasionally late working sessions and meetings where only parts of the group could attend. Thus, it had an impact on the stress level in the group, which was why a decision was made to make a schema containing the office hours where people could be contacted and why a planning a fixed meeting schedule for the week was made. This should have been done a lot earlier in the work process, since this initiative created a better working environment for most of the group members.
Besides the group work, it has become clear that a better use of TAs and application domain specialists throughout the course would have been rewarding, because this would have resulted in a better understanding of the application domain from the beginning, which would have meant less resubmissions and a better foundation for future work. Some of these challenges mentioned above might originate from a delegation of tasks happening too fast, and thus the group did not always take the time to talk about the theory and establish a solid and common knowledge before beginning the work. In this way, the approach became much more practical with a "fail faster" mentality, that also had its pros because a lot of practical experience was achieved quickly. However, some resources could have been saved by using slightly more time on the theory before trying to solve the tasks. ======= When thinking of the group work in retrospect it has become clear that certain things could have been done differently which would probably have improved the communication, cooperation,

and efficiency in the group. The first thing which should be mentioned is the attempt to follow the SCRUM method, which only was accomplished partially since one of the core features (the stand-up meetings) was not done by the group. If these meetings had been upheld it would probably have strengthened the communication which could potentially have lead to fewer misunderstandings and less miscommunication during the work.

During the beginning of the group work Trello was not used probably, which meant that the group work did not become as structured as it could have been. By using the Trello board probably, it would also have made the first issue with SCRUM easier to handle since it would have been possible to structure the SCRUM using this tool. Additionally, the use of a Trello board would have made the planning and distribution of tasks a lot easier.

Better communication could also have been accomplished by the use of a Trello board combined with a more structured use of the project's Facebook group e.g. by setting up some guidelines for how and what to write. By scheduling strict deadlines and communicating more about them, some of the unfortunate mistakes with missing content, which happened during hand ins could have been avoided. Furthermore, a better set of rules for VCS when writing the documents in LaTex could have prevented some critical compile errors.

In the beginning, the working hours were very flexible and mostly decided based on people's job schedule. This lead to occasionally late working sessions and meetings where only parts of the group could attend. Moreover, it had an impact on the stress level in the group, which was why a decision was made to make a schema containing the office hours where people could be contacted and why a planning a fixed meeting schedule for the week was made. This should have been done a lot earlier in the work process since this initiative created a better working environment for most of the group members.

Besides the group work it has become clear that a better use of TAs and application domain specialists throughout the course would have been rewarding, because this would have resulted in a better understanding of the application domain from the beginning which would have meant fewer resubmissions and a better foundation for future work. Some of these challenges mentioned above might originate from a delegation of tasks which was too fast, and thus the group did not always take the time to talk about the theory and establish a solid and common knowledge before beginning the work. In this way, the approach became much more practical with a "fail faster" mentality which also had its pros because a lot of practical experience was achieved quickly, but maybe some resources could have been saved by using slightly more time on the theory before trying to solve the tasks.

2 Individual Reflection

2.1 Individual Reflection of the Work Thor

The thing that struck me the most, was the changed structure of the project. We have been used to work on projects with predefined tasks and requirements. As opposed to this project where we rely on our own observations and efforts. Initially, I did not understand that we were supposed to treat our teaching assistants and lecturer as people from within the application domain (e.g. consultants). I was rather skeptical about this approach but now understand how important it to be able to analyze customer needs in a given domain and derive a solution from these requirements. It may feel like a conjured environment, since we do not keep contact with actual customers but the similarity is close and has helped challenge my work principles.

In regards to the group work, we started out with a flexible meeting schedule, which worked most of the time but I personally prefer a more planned approach. Sometimes we did not manage to review the work of all team members, which ultimately resulted in assignments with missing or inconsistent content. Consequently, we ended up using Trello to achieve a better overview along with a time schedule that all team members agreed with. This helped improve the cooperation and coordination of the project dramatically.

As in the prior projects, we used version control (Git) allowing us to work simultaneously on the mandatory assignments and code. This worked very well but required a strategy on how to use Git in order to avoid merge conflicts. Thus, we thoroughly went through on how to use branches and specific naming conventions when working on separate parts of an assignment or code project. I think we should have done this from the start, instead of having many merge conflicts requiring refactoring. This along with proper coding conventions minimize the overall time spend refactoring and makes what we produce more consistent and correct from the start. Consequently, I personally wrote a document about naming conventions and branches in Git, which has been read and understood by all team members. I had a bad experience with the lack of these agreements in the First Year Project were people would have different opinions about documentation, placement of brackets and version control. This ultimately resulted in many extra hours of work close to the deadline, which could have been avoided.

The work load was equally distributed among team members and I did not find any personal issues working together. This is based on previous experience working with the same people. Thus, we already know our different personalities and have thoroughly established the different skills that we have to complement each other. The only new conflict we had in this specific project was based on disagreements about time and priorities. In other words, some members were fine working on the project all days of the week while some members preferred

a clearer separation between work related activities and social life. We had to solve this by making an actual document showing which times of the week each member is available or does not want to be disturbed. This will probably affect my choice of group for the next project, since I think a general agreement of work hours and time spent on the project is important to establish a common goal and work ethic.

2.2 Individual Reflection William

The major issues I have encountered during the project was in regards to:

1. Time
2. Understanding the application domain
3. Working with LaTex using Version Control
4. Partly Unstructured Work and lacking Communication
5. Hand-In of Assignments

In regards to working on LaTex, I found that the communication and the rules for using git was too vague. This resulted in many merge conflicts and lost data, because the guidelines were not clear enough. The work was done without enough structured planning in the beginning, which resulted in people not being sure about who was assigned to which specific assignment. Instead people had a more general idea about which parts of the assignments they were working on. This made it hard to know exactly who was responsible for what, and so it was hard to know, who to go to when having/spotting a problem. About the communication, the lack of this (also in the beginning) resulted in missing hand ins or hand ins with missing sections or wrong/duplicated UML diagrams. Also the documentation of the development could have been better if versioning had been done in the beginning of the project. New realeses of the program and the different design documents should have been made frequently during the work to reflect design choices and evolution of the program better.
Another major struggle has been understanding the application domain. In the beginning the application domain seemed very confusing, but not enough effort were made to generate questions for the domain expert to explain the domain. This was a huge mistake, which had a seriouse impact on the project during the implementation, because the development of the program went in a wrong direction. This resulted in a wrong implementation, which had to be chaged quickly 2 days before the handin deadline. Because of this no functioning implementation of the program was ready for handin.

2.3 Challenges in the Cooperation and Coordination

Some troubles were encountered in accordance to when and how much to work. This was because there were conflicting opinions about how or even if it should

be acceptable to declare oneself for unreachable because one would like to separate the study from the social life.

2.4 Individual Reflection Dennis

3 Challenges while working on my part of the project

One of the greatest challenges I encountered regarding the project was to have an understanding on the application domain but also an understanding on how the requirement analysis document and system design document was to be written. This resulted in some sections being incorrectly written and was consequently required to be rewritten.

Final edition
As written previously the greatest challenge that I encountered throughout the project, was indeed the understanding of the application domain. This resulted in the program not being implemented as required. However, after I had a personal meeting with Steven and Paolo I was then truly able to understand the application domain which I shared with the group. Yet, this was unfortunate too late. However, I did manage to create and implement a study configuration UI that was to reflect the actual process of defining a study. Taken the UI into consideration, being solely responsible for creating a UI while also learning a new framework was also really, though, but I did get out from the process.

4 Impediments with regards to the team cooperation

Some impediments I encountered was the lag of communication and team planning at the initial end of the project. We did not delegate the work probably and that resulted in some group members having a greater workload than others. The lag communication resulted in some work being incorrectly made or some required work resources not being shared correctly so others can continue their activity. By example, when some UML was made one may forget to share these to the whole group so they can proceed with their section/activity.

Another thing regarding communication is the fact that we did not agree on common terminologies. Therefore, one may use one terminology while another uses a dierent terminology. Consequently, the report ended up being inconsistent and was therefore required to be xed. This could all be avoided if we had a clearly communication on what was to be done and how we achieved

Final edition
Something I really is missed was to meet up with my group in the weekdays. This would allow us to have a deeper discussion on issues instead of going around with them alone. I'm not implying that the others didn't work. They really did and I appreciate it, but since we only meet two times a week while also being confused on the application domain was not really optimal regarding creating a

proper program.

5 Retrospectively change something with regards to my approach to the cooperation within the team

I would have been clearer on my communication to avoid misunderstandings. I would also change the initial approach of the project in that sense of better planning and group coordination. It may have yielded greater results if we all had researched a bit more on the application domain, but also, how the requirement analysis document and system design document was to be written, so we could have avoided aimlessly assumptions on how they were to be made.

Final edition
Reflecting on the written above I did improve my communication. I was able to explain the application domain for the whole group which did help to steer development in the right direction. Nonetheless, I would in the future utilize TA's and other experts if I sense the slightest form of misconception because going through all of this again is not worth anyone's time and energy.

6 Cooperation improvement throughout the project

The group did, fortunately, mature throughout the progression of the project. The group had created xed work schedules and additionally utilized a Trello board. This has yielded great results in terms of planning and communication. The xed work schedules enable us to plan and propose what task has to be prioritized and accomplished while a Trello board allows the group to evenly distribute the workload among the members. If the group did this from the beginning, countless work hours and resources could have been allocated on some other sections of the project. Nevertheless, the group had apprehended this issue and has in that sense improved.

Final edition
Improvements did occur as mentioned above. Also, we ended up communicating more but that was, unfortunately, near deadline.

6.1 Individual Reflection Jacob

7 Individual Reflection Jacob

The biggest challenges I have faced in this project has without a doubt been the UML diagrams and understanding the the terminology used by the user. It was especially hard to understand the users needs and what was required of our solution. Not properly understanding the user domain made it especially hard to create scenarios and use cases which I struggled with. This could of course have been avoided by asking the client more questions, but regrettably. // Furthermore. In the code implementation we experienced once again our knowledge of of the application domain was lacking which lead to last minute changes to the program, redesigned tests and people were moved from developing on the project to patch-working, which hurt our ability to deliver a satisfying program.

7.1 Teamwork

I think the teamwork in this group has progressed smooth and without too many hiccups. There has been incidences in which the group had to deal with team members disappointing each other, but each and every time it has been resolved swiftly and without trouble. I think we have avoided a great deal of internal conflict by defining office hours in which team members could be contacted in. This has greatly helped group members with different sleeping patterns to separate work from spare time.

7.2 Retrospective

I think we could have saved ourself from a lot of hurt by implementing and experimenting with methodologies like Scrum earlier in in the process. I noticed we became much more organised after experimenting with a dedicated task board to visually represent our current progress. From my own experience, I can highly recommend this course to teach these methodologies earlier in the course.

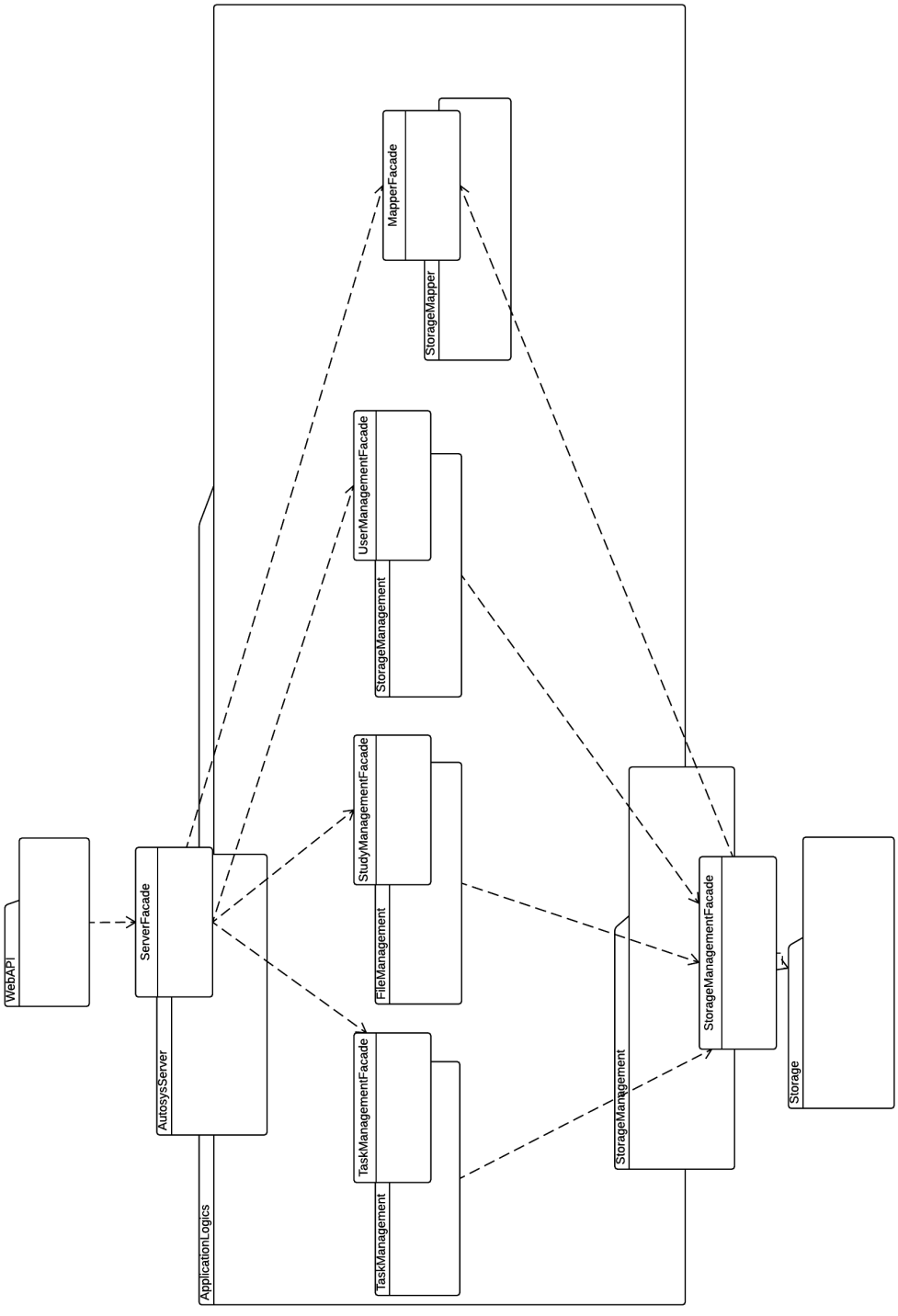
6 Literature

7 Appendix

7.1 Ideas For Future Application Logic Structure

On the following page is an altered Subsystem Decomposition UML Diagram for the programs application logic, which better match the necessary logic for the program to function as required by the application domain.

Use of Facades
To reduce the complexity of each subsystem a facade could be used to provide only the exact functionality required for the other subsystems in the logic to work. Having one point of access for each subsystem could make it easier for group members less familiar with a particular subsystem to use its functionality when working on another subsystem. If new functionalities should be exposed from a subsystem this would have to be reflected in the facade as well which makes the structure less flexible than providing access to all components in a subsystem. This is worth noticing but still the advantages achieved through the overview of the facade is arguably bigger than the reduced flexibility.


WebAPI
ServerFacade
AutosysServer
ApplicationLogics
MapperFacade
StorageMapper
UserManagementFacade
StorageManagement
StudyManagementFacade
FileManagement
TaskManagementFacade
TaskManagement
StorageManagement
StorageManagementFacade
Storage

7.2 Log

Logbook - BDSA15

Dennis Thinh Tan Nguyen, William Diedrichsen Marstrand, Jacob Mullit Mniche,
Thor Valentin Aakjr Olesen Nielsen

December 16, 2015

Contents

1 Log for date: 11/11-2015

Start: 14:00 — End: 16:00

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins

- Set up Trello and Pomello
- Set up Log book
- Initialized C# program structure

Sprint Planning

- Sudy ConfigurationUI (Web Service Interface)
- Internal communication among subsystems
- Subsystems Classes
- Storage Interface
- Work Load
- Prviding Blue Team API
- Dependency Injections
- Using Adapter Pattern

2 Log for date: 11/12-2015

Start: 13:00 — End: 16:00

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- WebAPI (Fra Steven) - Adapter Pattern, Facade (Autosys Server)
- Initialized and refactored project structure - packages to projects
- UML Subsystem refactored
- UML Entity Object refactored
- Entities first

Branching Rules

- First create a local branch
- Then publish the branch so it resides on oring/master
- Required start name for naming convention for branches: implementation/, testing/, documentation/, bugfix/, refactor/

3 Log for date: 14/11-2015

Start: 12:30 — End: 15:10

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche attended.
- Thor Valentin Aakjr Olesen Nielsen not attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- Creating overview of every subsystems
- Updated LucidChart of subsystems
- Defining classes and their responsibilites of every subsystems (Application logic and Storage)

Sprint Planning:

- Implement design decision for the new codeskeleton

4 Log for date: 11/16-2015

Start: 16 — End: 21

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche not attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- Code Skeleton completed.
- Pull requests reviewed.
- Merged branches into master.
- Discussed about work processes and how we contact each other outside ITU. Overall conflict is that study related activities seem not to be separated from other activities thus causing a tense stressful environment for certain members. One proposal is to keep track of each members time board so that people know when they can contact each other or when they should not be disturbed. This will be done with a document showing everyones schedule and using status tags on Trello.

Sprint Planning:

- Need to plan future work and time for meetings to achieve a clear distinction between activities related to ITU and others.
-

5 Log for date: 11/17-2015

Start: 9:00 — End: 12:00

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche not attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- Cleaned up code skeleton
- Added missing classes
- Implemented WebApi

Sprint Planning:

- Update SDD and RAD (UML subsystems, datafield definition, user definition, update entity object model)
- Create time schedule for all group members and dertermine meeting hours

Work planning:

- A Google Document has been made where the group members have written the time intervals where they are okay being contacted.
- In the document mentioned above working schemes have been made as well, indicating when group meetings are held.
- This is to separate study related activities from social activities thus avoid a stressful environment

6 Log for date: 11/24-2015

Start: 9:00 — End: 16:00

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- Implement code skeleton feedback
- Connect implementation tasks with functional requirements
- Fix UWP vs standard application issue

Sprint Planning:

- Need to fix subsystem dependencies with TAs by possibly using
- Connect functional requirements with implementation tasks
- Plan coding tasks for the next 4 weeks

Implementation priorities based on functional requirements subsystem dependencies:

Application logic

- UserManagement will be implemented first because WebApi and StudyManagement depend on it. We need to define how a User is to be represented so that SM can use WA.
- StudyManagement should be implemented secondly because its criteria and phase classes are used to define a study configuration. ProtocolManagement depends on a finalized study configuration.
- ExportManagement and PaperManagement can be implemented independent from rest of project.
- All subsystems depends on Repository with predefined CRUD operations.
- WebAPI depends on all subsystems based on their methods. We need to define what methods each package handler holds in order to develop our WebApi. The WebApi will be based on the deliveries from Steven.

Interface

- StudyConfigurationUI is independent from all subsystems in application logic. It is based on a web api that uses the server in the application logic. Should be able to retrieve teams stored in database and create a study define passed to the application logic.

User Validation The application logic will not handle user validation (e.g. manager or researcher), which is handled in the interface part of the system between yellow and blue part. The only thing application logic considers is whether a given user exists in the database. Roles are checked in Study Management based on an enum flagtype related to a user, e.g. a user can have a "Reviewer", "Manager" or "Validator" role determining which tasks are to be returned. A Reviewer will receive review tasks and a Validator will receive conflict tasks. Blue team will validate users based on whether they represent a Manager or Researcher.

Code Skeleton Design Changed to reduce coupling

If all packages have a facade interface used by other subsystems it allows single responsibility. Also allows team members to work on subsystems independently. We do not touch each other's code this way. A sub system should be able to be pulled out and replaced without causing trouble. By way of example, ExportManagement depends on the ProtocolManagement and its a predefined protocol. A using statement for Protocol is used in ExportManagement. Thus, we need to change the UML and use a dashed line between ExportManagement and ProtocolManagement. The dependency resides in where the object comes from and not how it is passed (e.g. from WebAPI).

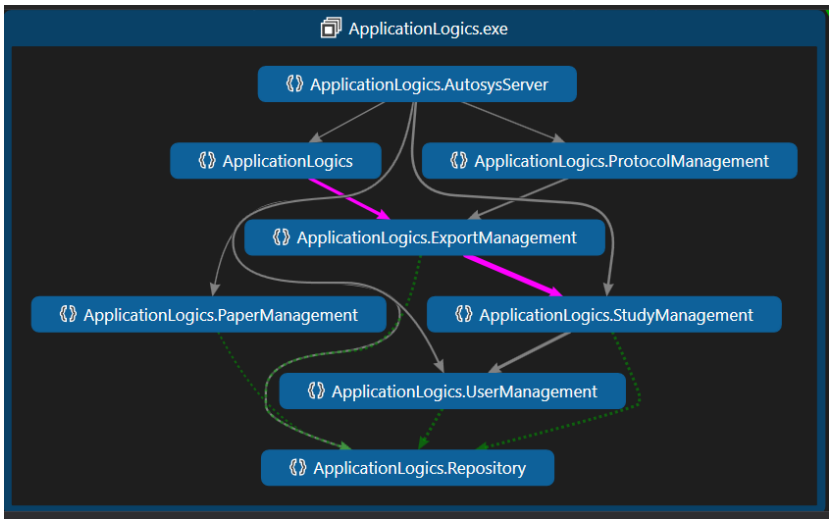


Figure 1: Subsystem Dependencies

7 Log for date: 11/26-2015

Start: 9:00 — End: 14:00

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche not attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- Rework database design
- Working on Criteria
- paperManagement

Sprint Planning:

- Talk with TA about database design
- Discuss Test driven development

Work planning:

- Add new title in each logbook, which detials any radical design changes.
- Thor and dennis has been working on a revicited entity relation model to be able to create the database
- We have desided to use a microsoft database to avoid any conflict between different types of databases
- We don't need to worry about database queing. Aparently c # will automaticly handled that aspect. According to Mokkel TA , each task will automaticly queue up to be proceessed. I don't think this will solve it on the client side, but that is hardly our problem
- You might want facilitate regular expressions in the search to enable more advanced searches. Instead of asking does this name appear in the text, then you can ask, I want to sample papers where this name appeared at least 8 times.

Work accomplished today:

- E
  R model created

- Subsytem dependency resolved: Create exportable item Interface To expose methods for formating exporting Criterias, Protocol and Criteria will implement CRUD
- Database Dispose have been replaced by Using blocks
- Database connection is maintained upon methods calls
- Created Facade package in application logic, used to translate entities to storage
- Repository have been moved to storage, and will now contain concrete repsoitories for all entities
- **William** is working on the Paper management class
- **Jacob** is working on the Phase and Criteria classes
- **Dennis** User management class
- **Thor** Storage, IFasade

8 Log for date: 12/01-2015

Start: 9:00 — End: 16:00

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- Testing strategy
- Shared data model required for use in repositories, stored model entities and testing
- Agreed on test syntax methodToTest_condition_expectedbehavior
- AutoMapper framework used to translate objects from application logic to DTO objects and Stored model entities
- Can now connect to a local database created from code first (AutoSys-DbModel). Requires installation of SQL Server 2014 and login with (localdb)/myssql
- Agreed to prepare questions and think about potential conflicts involving work of others when implementing things. We now put more focus on preparations before group work. Members are expected to prepare questions and identify potential challenges involving the work of others.
- Reimplemented ids for entities in application logic where needed.

Sprint Planning:

- Sections from Testing Strategy document has been assigned to individual group members. Expected template delivery on Thursday the 3rd of December.
- Documentation boards on Trello are to be complete in order to clean up Trello for new tasks.

Current work:

- Thor is currently working on implementing the Storage repositories containing CRUD operations for all model entities. He will first implement the tests for each repository. This is now possible due to the agreed data model (had conflict with Paper and Criteria entity).

- Jacob is currently writing blackbox tests for the CriteriaHandler, working on a simple sorting operations which let a user sort Articles based on criterias and is working on Phase
- William is curently working on Papermanagement and exploring the nUnit framework (Which help reducing redundant test cases).
- Dennis working on black box testing and Fasades

9 Log for date: 12/03-2015

Start: 9:00 — End: 16:00

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- We have agreed to make shared decisions and planning in the first hour both Tuesday and Thursday.
- Thor will look at Jacobs pull request and check Phase and Criteria. Thor will check Phase and Criteria, implement Tag and Entry entities and fix Foreign Key functionality in model entities using ICollections for lists and dictionaries with objects that cannot be saved directly in the database. Remember to write System Design strategy before repositories. Finally he will make tests for storage repositories and implement repositories based on these.
- Talk about branching methodology. Issue can be seen in the branch "EntityRefraction" holding too big merge conflicts.
- Dennis will write on usability testing. Dennis is wrapping up on UserManagement. Need to refine documentation and tests. Dennis will present AutoMapper to the rest of the group.
- Merge AutoMapper BaseClass changes from Dennis and Jacob's pull request.
- Talk about Object Design model.
- William will first write on unit testing in testing strategy. He will finish PaperManagement. Then work on ExportManagement implementation and testing. Will begin mockups for GUI.
- Jacob will write his integration testing strategy. Jacob would like to work on functionality used to search through criteria. Jacob wants to work on search functionalities using criteria.

Done:

- Implemented AutoMapper with profiles for each subsystem.
- Wrote testing strategies.

- Fixed stored entities and foreign key functionality.
- Added Tag and Entry entity used for BibTex parsing
- Refactored and cleaned up code.
- Bug fixing.
- UserManagement tested and documented.

Sprint Planning:

- UserManagement to be done.
- PaperManagement to be done (not ExportManagement).
- Storage repositories to be done.
- Adapter between WebApi and Main Handler to be done.
- MainHandler and AutoSys Server need to be finished next week.
- Start looking at UI.

Design choices:

- **Export**: William would like to know how we import and export bibtex files and protocols. William suggests that we store bibtex files on computer as text file, we parse them in program as Paper objects and Paper will hold references to resources on the computer. This allows to find resource on computer with key from Paper. Pdf file is sent as HttpRequest as DataStream. The tricky part is to update the pdf in the directory when the bibtex is updated. However, this is assumed non vital.
- **Import**: Should BibTex parser be customized when parsing or should it have a default setup? Do we assume that the bibtex file is already imported from the start? Blue Client sends raw bibtex file to Server, this should be parsed into database. We need to take raw bibtex file and save as Papers. Blue team will support UI to import bibtex file to database. They need to do this with user defined tags outside the server. We accept bibtex syntax but could allow input like "Retard" instead of "Article". Suggestion is to compare bibtex file to default tags in system. If no match it will not become invalid but just save the bibtex file with this info. Undefined tags will be defined as new bibtex tags in the bibtex parser. All well defined is saved as papers in database. All undefined tags will be saves as new bibtex entries in the database. Need to save new Tags as strings in the database. This allows to search for things in the database with bibtex tags that only exist. Field entry type (Article) can just be chosen, true false. Fields that do not exist on all papers, system needs to know if field exist. It can either compare with existing list of fields in

database. Or it can look up in database with Papers (could potentially be a big list). The field type could also have a reference to papers with this type. Solution is to have a Tag entity holding all fields as strings retrieved from id references in Paper.

- Decision: entries and fields are saved in the database and mapped with the Paper. Paper will have a entry type (Article), list of fields (enums restrict new fields) and resource reference int (used to find pdf). We will no longer make checks in the bibtex file but only import the whole file.
- Our system becomes more flexible by allowing users to upload bibtex files with custom tags that fit their Research but is not generally found in bibtex format. Tags should be stored in the database.
- New entities: Entry with string EntryType (required) e.g. History article, List of Paper ids and Tag with string TagType, list of Paper ids
-

10 Log for date: 12/14-2015

Start: 9:00 — End: 16:00

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- UI should be done.
- Integration test of backend (database, mapping, web api and UI backend).
- ODD design patterns and class descriptions.
- RAD and SDD cleanup.

Sprint Planning:

- Jacob is working on Web API and communication to Application Logic layer.
- Dennis is almost done with UI front end and will start on UI backend.
- Thor is finishing Adapters and fixing dependency injection of repositories in adapters. He will optionally make AutoMapper profiles convert entities from logic to storage.
- William is working on BibTex parser and tests.

Tomorrow plan :

- Thor is making integration test of database with EF and repositories (either a separate local test database with new connection string or in-memory using Effort framework.
- Dennis is reworking UI and implementing backend.
- Jacob is still finishing Web Api.
- William is working on Study Configuration logic (upload and parsing bibtex, criteria, datafields and phases for creating studies).

Goal :

- UI is required to create a study and backend needs serious work from Dennis.
- Application logic to be implemented by William and Thor.
- Jacob has to finish Web Api.

11 Log for date: 12/15-2015

Start: 9:00 — End: 16:00

Member attendence:

- Dennis Thinh Tan Nguyen attended.
- Jacob Mullit Mniche attended.
- Thor Valentin Aakjr Olesen Nielsen attended.
- William Diedrichsen Marstrand attended.

Meeting pins:

- Study Configuration and Phase work flow reworked completely after meeting with Steven and Paolo.
- Changed objectives in order to achieve a minimum functional solution.
- Integration test of backend (database, mapping, web api and UI backend) (Thor)
- Update logical entities in Storage after new work flow design.
- Discussed ODD design patterns, trade off-s and class descriptions.
- Convert system testing design document to LaTex (Thor).
- Collect all discussions and design choices from notes and insert in ODD (Thor).
- Insert footnotes for all requirements in RAD and tell about modifications or changes in Appendix (Thor).
- Make class diagrams for most important classes.
- Clarify design choices on architecture, mapping and repositories.
- RAD and SDD cleanup.

Goal :

- UI is required to create a study and backend needs serious work from Dennis.
- Application logic to be implemented by William and Thor.
- Jacob has to finish Web Api.