# System Design Document - Yellow Project

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

December 17, 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. A study consists of a set of resources on papers, one or several phases, a set of data fields, set criteria and a set of study participants. The papers that are bound to a study are being filtred with a set of inclusion- and exclusion criteria. These criteria are based on data fields such as year, title and so forth. A **PHASE** will consist of researchers that have various roles. One can either be a reviewer or validator. The main responsibility for a reviewer is to review a given number of papers from a set of visible data fields and requested data fields that are defined in each phase. Visible data fields determines what the reviewer will see from a given paper. By way of example, if one had "year" as visible data field one will only see a given year when the given paper is to be reviewed. Requested data field are datafields that are defined but has not been answered. By way of example, a requested data field could be defined as a data field about whether a paper is about soft engineering or not. The reviewer will answer this based on the given set visible data fields. This process is regarded as a review task. Each review task is given to reviewers based on their defined workload. A workload defines on how many papers they are to review. When some task are done and they have conflicting data. By way of example, Reviewer A has returned a task request that differs from Reviewer B's task request. The validator will then be given a task where he must validate which of the task request are accepted. 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**.

The purpose.[1] of the system that yellow are to create, is to create a study-configuration UI that enables a study manager to define a study and their members, resources, phases roles and data field. The Study Configuration UI should also let the study manager manage existing studies by editing or deleting them. Another part of the system that the yellow team is responsible for is to create a Study Configuration Server that can create tasks defined in a study and distribute them to each member automatically. The task destribution workload percentage can either be defined as a fixed number or a user input. The server must also check for conflicting datafields when a given task is complete and then create conflict handling tasks for validators. The server must also be able to store studies, tasks, users and teams within database so they can be retrieved or modified at any given point of time.

[1] The previous assumption on the solution domain was to let the user create a study where he, in addition, was to establish the tasks and distribute them manually in the Study Configuration UI and then submit it to the study configuration server. This assumption was brought up to discussion with the client and was redefined so tasks were to be generated automatically by the study configuration server when he had uploaded a configured study.

## 1.2 Design Goals

- **Ease of use**:
  Goal: It should be relatively easy to set up 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[2]. 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.[3] 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

[2]This goal has been deemed to be out of the scope regarding the system implementation. We assume that the server is being monitored by som system administrator so they will have the responsible for restarting the system and its environment.

[3]We assume that the database is not located on the same computer as it could have been stored in the cloud. However, when developing the system the database is located together with the study configuration server

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 an external database, but just focus on program in front of him

# 2 Proposed Software Architecture

## 2.1 Subsystem Decomposition

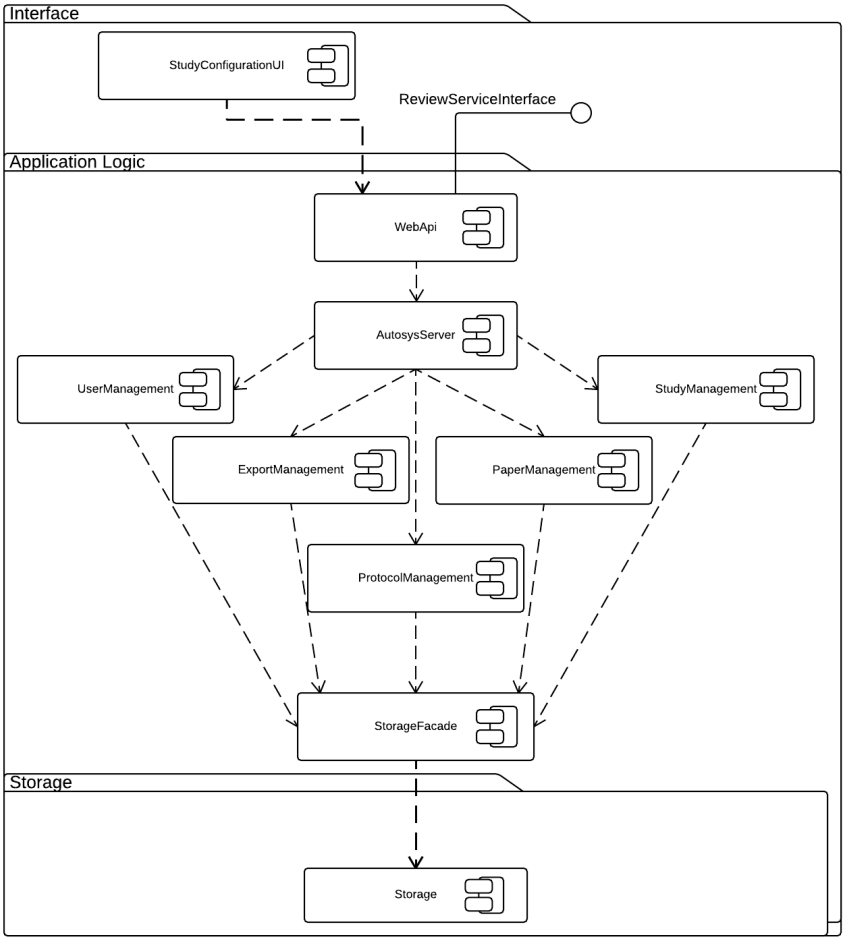


Figure 1: Autosys subsystem decomposition (UML Component Diagram, layers shown as UML packages)

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 become 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.[4] The **ExportMangement** provides

[4]After the discussion with the client. The solution domain for this part has been revamped. The StudyManagement subsystem must be able to retrieve new studies and store them, return studies to the blue team. In addition, the studyManagement subsystem must also be responsible for generating tasks for the reviewers and validators. These tasks are created and distributed automatically with a given workload strategy. The workload strategy can either be implemented with a fixed set of strategies or receive a user-defined workload that are prepared in the Study Configuration UI.

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 coded system, and all components symbolize directories except the WebApi component which have been made a separate project outside of Application-Logics. This was due to the possible complications with a 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/papers that are uploaded to the server. The second set (referred as blue storage) consists of objects that are created and accessed by the blue part of the system(eg. Users, tasks, 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 data fields, tasks, inclusion and exclusion criteria 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.[5] 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.[6] Configuration storage is also well defined and will not change once the study configuration has been made.

[5]This definition as been changed after the discussion with the client. It was previously assumed that the resources were to be stored as a shared resource pool were other studies were able to access. However, this assumption has been redefined so sets of resources are bound to their respected studies. Therefore, the lifetime of these objects are not bound to the program but are rather bound to the lifetime of a study. However, as mentioned in this section this part of the storage is well defined and will rarely change since one can only define a given set of paper to a study when it is being configured.

[6]After descussion with client. It should be said that tasks are created by the server and are therefore managed regarding their storage.

In this scope of the Autosys system, the main focus of persistent objects is set on metadata storage and study configuration storage.[7]

**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. Since the system is being developed in a Microsoft environment, the group has decided that Entity Framework, which is provided by Microsoft, will be utilized. The reason for this is it enables the group to easily define and create a database that can be used with their system.

[7] In addition, taskstorage is also a primary focus.

## 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>><br><<read>><br><<update>><br><<delete>> | <<create>><br><<read>><br><<update>><br><<delete>> | <<create>><br><<read>><br><<update>><br><<delete>> | <<create>><br><<read>><br><<update>><br><<delete>> |
| **Researcher** | GetStudies<br>GetStudyOverview<br>GetResource | GetTasks<br>DeliverTask<br>GetReviewableTaskIDs<br>GetReviewableTasks | <<read>><br><<update>> | <<read>> |

Figure 2: Access matrix for main AutoSys objects

[8]

## 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

[8]Note that Researcher should also be able to get conflict tasks

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.[9]
- *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.[10]

## 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 into two components, a client, and an 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 focuses on the server, if any questions to the user client should occur, please refer to the according Blue SDD.

All user clients will contact a single server. To solve this, the server will be multithreaded, 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 a low amount of users, but should this system expand a new design should be

[9]This is deprecated since an Entity-framework database is being utilized. Therefore it is assumed that this requirement is automatically handled by the framework.
[10]This is also deprecated. The reason is the same as the footnote above this one

conceived.

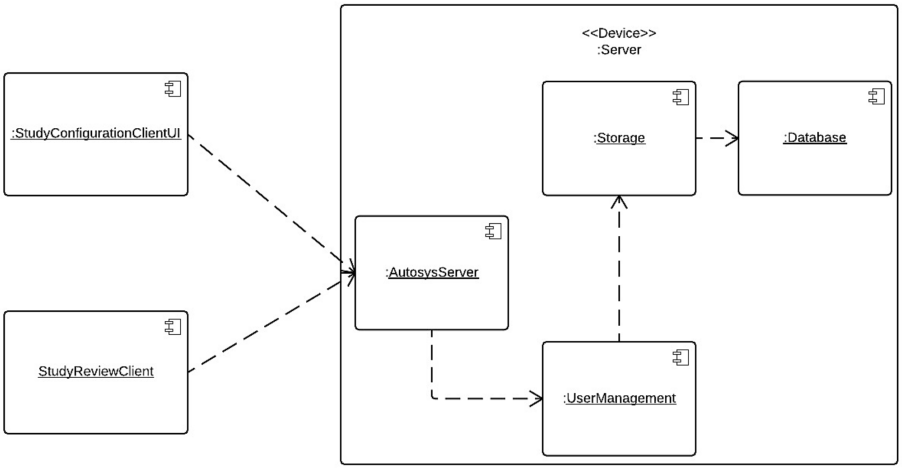


Figure 3: Software mapping of prior subsystem decomposition. Note that the components UserManagement, PaperManagement, and StudyManagement has been collapsed into a single unit called UserManagement

**Program components**
The server will be implemented in C # as requested by the client. By using C #, we can utilize Microsoft expansive database systems and interfaces especially in regards to the database. Additionally, this also makes the program easier to maintain, since C # is a well-known language and utilizes 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 client's 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. It is assumed that 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.