UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

# Software Requirements Specification, Technology Neutral Process Design and Software Architecture Specification


Maree, Armand
12017800

Watt, Brenton
14032644

Meyers, Charl
14024633

Tom, Elton
13325095

Tswene, Kamogelo
12163555

Molefe, Keletso
14222583

Spazzoli, Lorenzo
13304862


March 9, 2016

# Contents

# 1 Introduction

In this document it would be specified how a system would be developed for the department of computer science at the University of Pretoria in order for them to replace their current, not so efficient Microsoft Office Excel spreadsheet, system with a more concurrent and reliable option.

# 2 Vision

The client requires a system that would allow them to retrieve and submit meta-data about academic papers published by the department of computer science at the University of Pretoria. Certain users would be able to create new papers and the project leader would then be able to control certain properties of the project, like the current progress of the paper. It should also have a web interface where changes can be made and also an Android app.

# 3 Background

What lead to the project is the fact that that the client is facing a problem with the current system they are using. The spreadsheet which does fulfill its task of storing details about user's publications, does not have a user friendly interface, and does not allow researchers to see any changes made to the system immediately, it is also not a concurrent system and is deemed less reliable. The client wants a system that would improve the manner the users would be able to view their submissions and be able to modify the system to their liking, with changes immediately visible to researchers. Another aspect which lead to the development of a new system which would simplify the collaboration between the University of Pretoria users and users from other universities, is that instead of having a system which is local; a web interface could be used which is more accessible. The lack of ability to easily access previously entered user information data such as oneâĂŹs username or contact details also was a factor that lead to this project, as it would be more convenient and efficient to have to have an interface that would for example have a drop down list of all the user information data on the system.

# 4 Architecture Requirements

## 4.1 Access channel requirements

The system requires 2 interfaces:

- Web interface
- Android app (Could also be a mobile website)

## 4.2 Quality requirements

The following quality aspects needs to be addressed:

- **Scalability:** Less a 100 users will use the system.
- **Security:** Passwords will be hashed with the SHA512 hashing algorithm.
- **Reliability:** An automatic partial dump backup will be done everyday at 03:00 and each back up will be kept for a week. On top of this a full dump will be done every 3 days and will be kept for 2 weeks. Alternatively the backup will be done by an external program.
- **Audibility:** Every change should be logged and the person responsible for that log will be recorded. This will allow the administrators to track which users change what.
- **Maintainability:** Data that is deleted will only be moved to another database that might be a little slower. This will help alleviate some pressure off the main database.
- **Cost:** The total hours is estimated to be 120 at a cost of R500 per hour.

## 4.3 Integration requirements

The system should allow universities to connect to each other using an API. This would allow them to collaborate on projects and also to track the papers that are being written. The Android app should also be able to connect to the server via the API.

The protocol that will be used to transfer the data between the server and the clients will be the Hyper Text Transfer Protocol Secure (HTTPS). And the third parties that integrate with the system will access it either through the web interface, the mobile app or the provided API.

Some quality requirements that have to be considered are:

- **Security:** The data that is sent over the internet should be encrypted and it has been decided to use the Secure Socket Layer (SSL) encryption algorithm.
- **Reliability:** The reliability of the transfer of the data is dependent on the reliability of the internet connection.

## 4.4 Architecture constraints

There are currently no architecture constraints that the client mentioned.

# 5 Functional requirements and application design

## 5.1 Use case prioritization

**Critical:**

- There must be some web hosting server in order to be able to host a web interface, this interface will be the main access channel to the system as it makes it easier to access the system on a computer or non-Android device. If there is no web-server then there will be no web interface that can be accessed from research groups all over the country. This defeats the whole purpose of the system.
- The web interface needs to be accessible from the Internet preferably so that other research groups outside the University are able to access the system.
- The system needs to be able to save data in some way. If there is no possible way to store data then the system might as well not exist, because the purpose of the system is to store meta data about research papers.

**Important:**

- Users must be able to download a bibliography of papers released in a selected period.
- An Android app must be available that can access the system to make it easier to access the system on the go.
- Users must create their user profile manually and will not be able to link the system with a web service in order to create their profile.
- No item should ever be deleted in the system. Everything is kept. Old unaccessed data may be archived but never deleted. There needs to be some sort of recovery or backup system that enables a user to restore accidently deleted data.
- The system must be able to handle a max of at least 50 concurrent users as there is a possiblility that at least 50 users may access the system all at once. There will also be an estimate of at least a 1000 users. The system must be able to handle that amount of users.

**Nice-to-have:**

- The system must be lightweight.
- The system must show the completion of a paper in percentage that a user specifies after editing the paper.

- The system may link with Google Calender to synchronise due dates and alerts for the paper a user is busy working on.
- The web interface can be designed to work on mobile phones for those who want to access the system on the go but do not have an Android device.

## 5.2 Use case/Services contracts

- **Use Case:** Add a paper.

  **Primary Actor:** Registered author or HOD.

  **Brief:** Only an author or head of department can add new papers.

  **Postconditions:**

  - The article is saved to the database.
  - All authors are notified of their new paper.
  - The changes are logged.
  - Paper is displayed.

  **Triggers:** The user invokes the Add New Paper request.

  **Basic flow:**

  1. Present the actor with a new paper form.
  2. Actor fill in the form and clicks submit.
  3. The data is sent to the server and added to the database and logs the event.
  4. Actor is presented with a success message.
  5. Paper is presented to the actor.
  6. Authors are notified.

- **Use Case:** Editing a paper.

  **Primary Actor:** Author of paper or HOD.

  **Brief:** Only an author involved with the paper or head of department can edit the paper.

  **Preconditions:**

  - Author is presented with the paper in edit mode.

**Postconditions:**

- The article is saved to the database.
- All authors are notified of the changes.
- The changes are logged.
- Paper is displayed.

**Triggers:** The user invokes the Edit Paper request.

**Basic flow:**

1. Present the actor with the paper in edit mode.
2. Actor makes all desired changes and clicks submit.
3. The data is sent to the server and added to the database and logs the event.
4. Actor is presented with a success message.
5. Paper is presented to the actor.
6. Authors are notified.

- **Use Case:** Viewing a paper.

  **Primary Actor:** Author of paper or HOD.

  **Brief:** Only an author involved with the paper or head of department can view the paper.

  **Preconditions:**

  - Author is presented with the paper in read-only mode.

  **Triggers:** The user invokes the View Paper request.

  **Basic flow:**

  1. Present the actor with the paper in read-only mode.
  2. When the actor is done viewing they simply navigate away. No changes or events has to be logged.

**Request and Results Data Structures:** See figure 1.

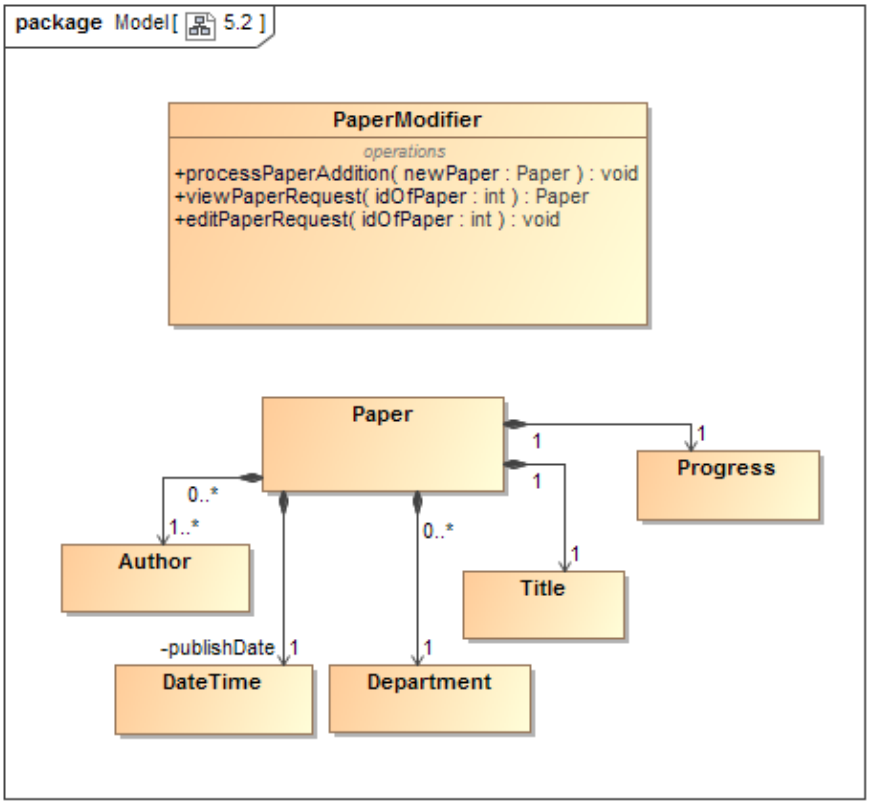


Figure 1: UML diagram for Request and Results Data Structure.

## 5.3 Required functionality

- Add a paper Use case: See figure 2

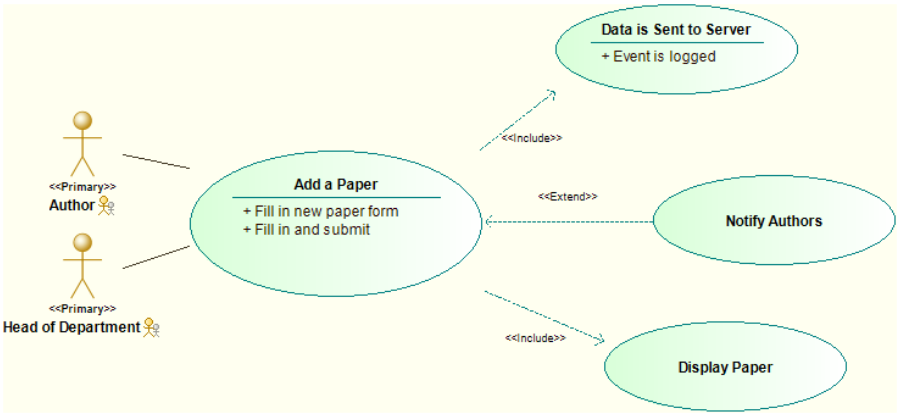


Figure 2: Use case diagram for adding a paper

- Edit a paper Use case: See figure 3

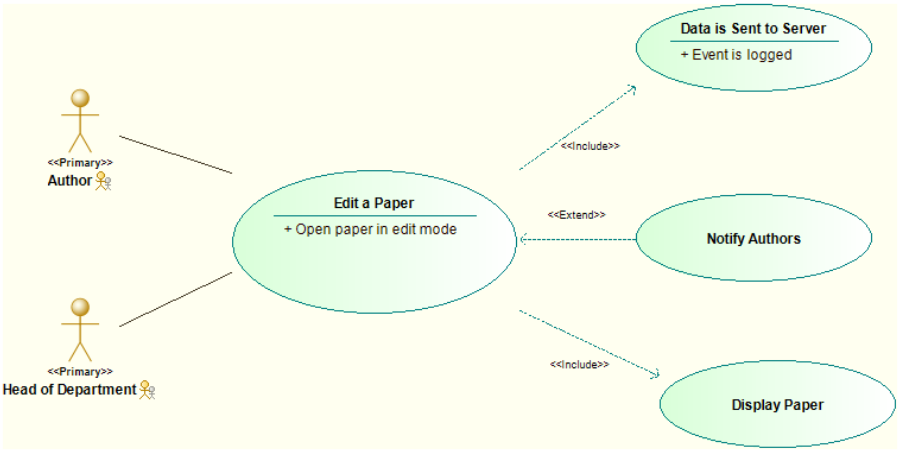


Figure 3: Use case diagram for editing a paper

- View a paper Use case: See figure 4

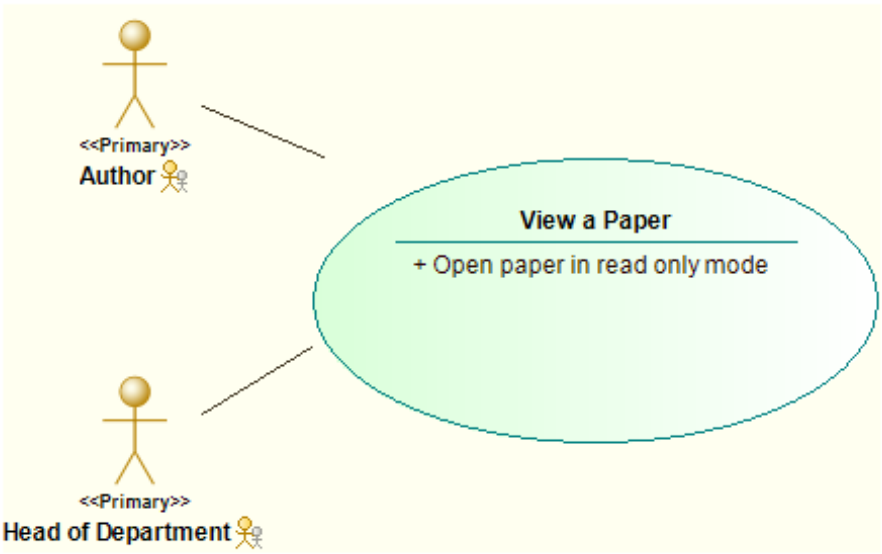


Figure 4: Use case diagram for viewing a paper

## 5.4 Process specifications

- Activity diagram to add a paper: See figure 5

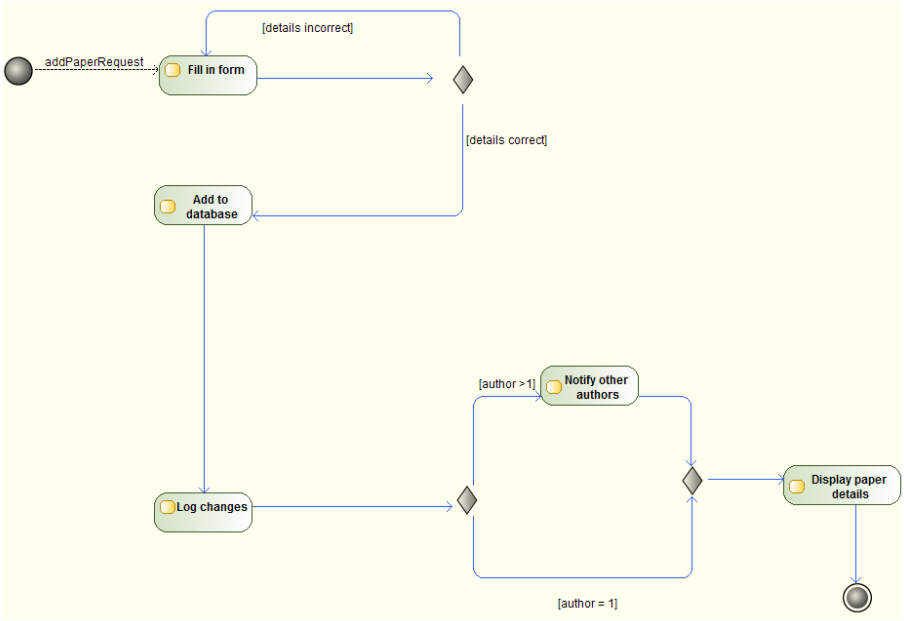


Figure 5: Adding a paper to the database

- Activity diagram to edit a paper: See figure 6
- Activity diagram to Login: See figure 7
- Activity diagram to Request document details: See figure 8

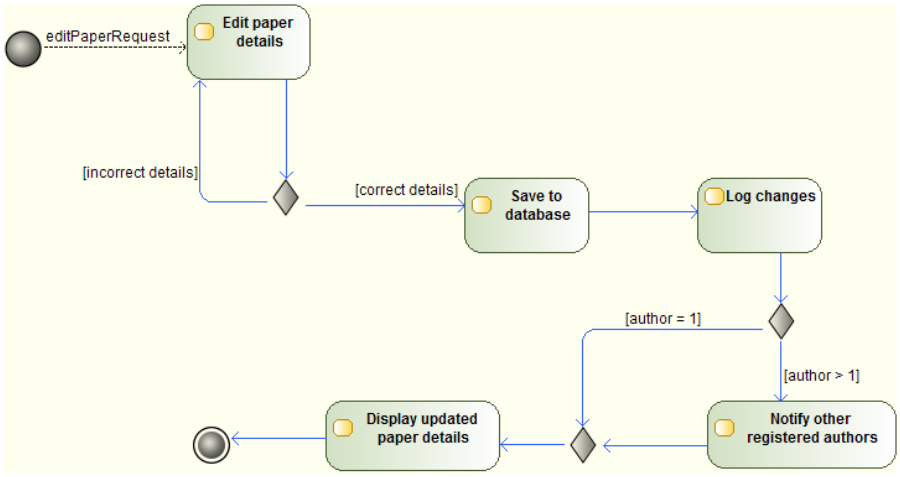


Figure 6: Editing a paper

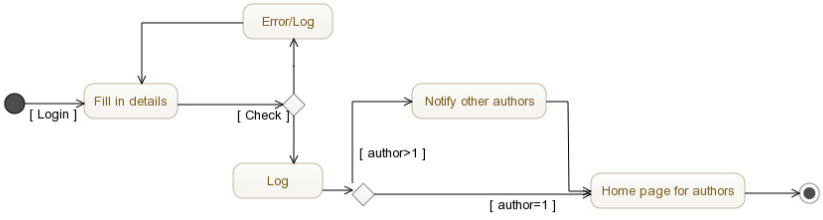


Figure 7: Login to system

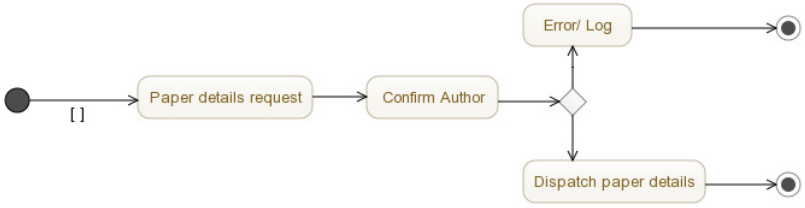


Figure 8: Request paper details

## 5.5 Domain Model

- Domain Model of the system: See figure 9

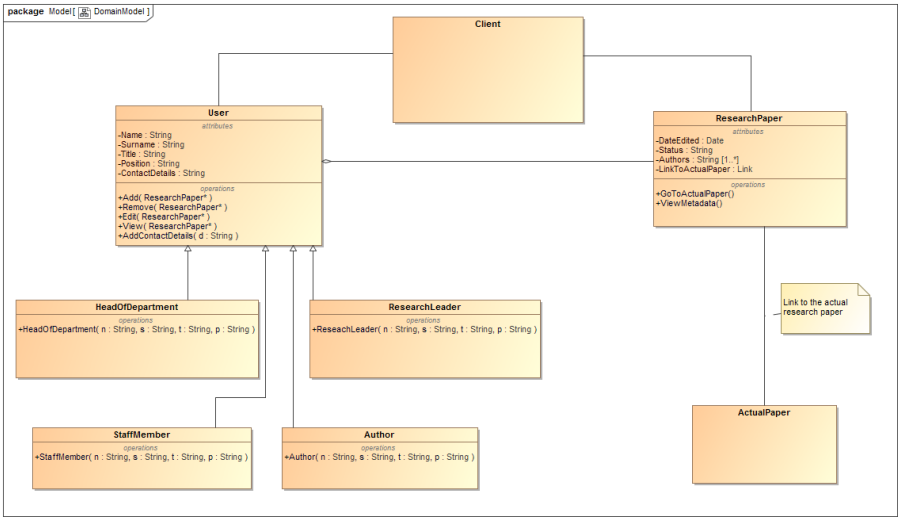


Figure 9: Domain Model

# 6 Software Architecture Documentation

## 6.1 Architecture requirements

### 6.1.1 Architectural scope

The architectural responsibilities that need to be handled by the software architecture are as follows:

- A main database that will store the papers that are added.
- A secondary database (separate from the first) that will act as the backup database where the partial and complete dumps will be stored.
- An infrastructure that will allow for notification to contributing authors that changes have been made to a paper they collaborated on, this infrastructure will also create a log of changes and who made them so that there is accountability.
- An infrastructure that will allow the users of the system to create and add papers to the database mentioned above as well as being able to edit and view them later.
- A mobile version of the above previously mentioned infrastructure must also exist.

### 6.1.2 Quality requirements

In order of importance our quality requirements are:

- **Reliability:** This is highly important as this might be the only place research could be backed up to, therefore, the system needs to be reliable. Reliability in this system is measured against the effectiveness of the backup measures in place as well as the system be available on demand.
- **Security:** The system needs to be secure from anyone who does not (or should not) have permission to use.
- **Audibility:** This is measured in terms of is every change noted and logged somewhere.
- **Scalability:** There will not be a huge amount of users making use of the system but it is important that if need be every user can make use of the system concurrently.
- **Maintainability:** Measured against how effectively the system choses to relocate data due to age.

#### 6.1.3 Integration and access channel requirements

#### 6.1.4 Architectural constraints

### 6.2 Architectural patterns or styles

### 6.3 Architectural tactics or strategies

**Performance** To better manage resource intensive tasks thread pooling will be used. There will be an object that receives all tasks and allocates them to a specified thread when a thread becomes available.

**Exception handling** Exceptions should be dealt with as soon as and as best as possible, but since it cannot be afforded that the system crashes there should be some form of "last resort" that attempts to deal with unhandled exceptions in such a way that the system as a whole keep functioning.

**Passive Redundency** There will be 2 databases that will both contain all the information. This will increase the reliability of the system by having a back up database that can take over at any point when the primary database fails. This will be transparent to the user.

**Transactions** Changes to the database will be done in transactions that will allow any changes to be rolled back to s checkpoint should an error occur. This will cause atomicity in the system and increase reliability.

**Security** Encryption to increase security https protocols will be used to transmit data and all private information (like passwords) will be hashed using the SHA512 hashing algorithm.

**Interatability** In order to increase interatability a modular approach will be taken when implementing the system by using dependency injection as far as possile.

### 6.4 Use of reference architectures and frameworks

The frameworks listed here all reduce development time and cost of the system.

The first framework to be used is Bootstrap for styling the Web interface. The reason for this is because Bootstrap makes web styling much easier and faster than just trying to use pure CSS. It also helps a lot with alignment that will help a lot with tables and follows a mobile first and responsive approach, this means that it will be a lot easier to develop a website that works both on mobile devices and desktops.

AngularJS will be used for the front end of the website along with bootstrap by using UI Bootstrap. The reason we use AngularJS is because it feaures dependency injection, along with mocking framework for unit testing among other things. AngularJS also features quick prototyping and easy testability.

PhoneGap (renamed to Apache Cordova, we will use PhoneGap in this document) will be used as a wrapper for the above website in mobile form. Because the client only needs an interface for an Android app and all major processing and storing will be done on a server, advanced native Android code is not needed. This again reduces development time because a responsive website already exists thanks to Bootstrap and UI Bootstrap. And PhoneGap makes the app cross-platform as a bonus. Finally a lot of plugins exists that can improve the functionality on top of a 'normal' webpage.

Ionic framework will be used together with AngularJS and PhoneGap in the place of Bootstrap. This is because Ionic enables much easier GUI design and a more responsive hybrid app. The reasonit will replace Bootstrap for the mobile app is because Bootstrap and Ionic do not play well because of name conflicts.

Finally Spring framework will be used on the backend for the server. Because Spring is built on Java EE it means Java can be used for the server. Some advantages this has over normal PHP is that Java is faster than PHP, especially now that the modern JVM is much faster. Java also has better integration support than PHP. And ofcourse the Spring framework makes it much easier to develop the backend and include depenency injection.

## 6.5 Access and integration channels

## 6.6 Technologies

Technologies used will be MySQL for the database , it is very light weight and will have sufficient access control for the given product. The server will be a mix of PHP and Java, both will connect to MySQL , the Java will be the host for the android connection and the PHP will handle requests from the web interface.

# 7 Open Issues

There are currently no open issues.