# Software Design and Arhitecture

Taylor Letsoaka
Nando Bingani
Nkavelo Nxumalo
Lesetsa Mafisa
Ziphozonke Mbatha

September 2018

# Contents

# 1 Introduction

## 1.1 Purpose

This document serves to present an architectural and design overview of the Examiners' Board Review System (EBRS). The primary purpose of the Examiners' Board Review System is to simplify and digitise the Examiners' Board Review meeting where marks and students' progressed is assessed and amended at the discretion of Senate. This SDD defines and describes the use of each view, the architectural constraints of the system, the functional requirements with a significant impact on the architecture.

## 1.2 Scope

This software system will be a Web-Application System for the Faculty of Science. This system will be designed to maximize the productivity by providing tools to assist in automating the voting system review and mark publishing process, which would otherwise have to be performed manually. By maximizing the faculty registers work efficiency and production the system will meet the faculty's needs while remaining easy to understand and use. More specifically, this system is designed to allow the faculty to manage and communicate with a group of Administrators, Dean and Lecturers to vote on the website. The software will facilitate voting between Administrators, Dean and Lecturers using the web-application. This is a Java web application that will allow members of the Science Faculty at Wits University to approve the promotion, failure or exclusion status of each student. This application enables each student's marks to be viewed on any device used by the participants of the meeting.

## 1.3 Definitions, Acronyms and Abbreviations

1. **Process** -set of interrelated or interacting activities which transform inputs into outputs [ISO/IEC 12207]. .
2. **User** - A member of faculty that is part of the administration or the mark review committee(The Dean, Head Of School and Lecturers).
3. **Software Requirements Specification** - A document that completely all of the functions of a proposed system and the constraint under which it must operate. For example this document.
4. **Faculty of Science** - A group of university scientific schools.
5. **School** - A department concerned with a specific discipline e.g School Of Mathematics.
6. **Member** - An employee of the faculty of Science who is part of the mark review committee e.g The Dean Of Faculty.
7. **Student** - A registered individual for a specific degree in the faculty.
8. **Vaadin** - An open-source platform for web application development.
9. **MySQL** - an open-source relational database management system.
10. **SDD**- Software Design Document

## 1.4 References

IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications (Revision of IEEE Std 830-1993), IEEE Computer Society, 1998.

The forthcoming section i.e. the Overall Description, will give a more broad explanation of how the product is intended to work. It will delve into things like the product functions, design constraints, assumptions etc. The Specific Requirements section is a derivation of the preceding section written specifically for developers hence it uses a lot of technical jargon.

## 2 Architectural Goals and Constraints

### 2.1 Architectural Goals Constraints and Representation

This section describes the software requirements and objectives that may have some significant influence on the architecture

Audience: all the stakeholders of the system, including the end-users.
Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system.
Related Artifacts : Use-Case Model, Use-Case documents.

**Technical Platform**
Examiners Board application will be deployed onto a J2EE application server.

**Transaction**
J2EE platform already has built in transaction capabilities, they will be used.

**Security**
The system must be secured, so that a customer can make online payments (Premium Membership) Basic security behaviors:

- Authentication: Log in using at least a user name and a password
- Authorization: according to their profile, online user must be granted or not allowed to receive some specific services (Automatic match finding, Ride Suggestion, etc...) For internet access, the following requirements are mandatory
- Confidentiality: sensitive data must be encrypted if any (student and staff credentials) .
- Safety: Staff login details must not be kept at a local database.
- Data integrity : Data sent across the network cannot be modified by a tier
- Auditing: Every sensitive action can be logged
- Non-repudiation : gives evidence a specific action occurred

J2EE security model will be reused

**Persistence** Data persistence will be addressed using a relational database and J2EE s Object Relational Mapping capability will be reused.

**Reliability/Availability**

High availability is required since there are time constraint issues related to the systems availability. Board members (users) should not be disappointed. The system's high availability will also ensure customer (faculty of science) satisfaction and loyalty. J2EE solutions will be used.
Targeted availability: 4 hours a day, 3 days a week (Maintenance at night)

**Performance**
Search queries should return Voting shouldn't take more than 10 sec per user.

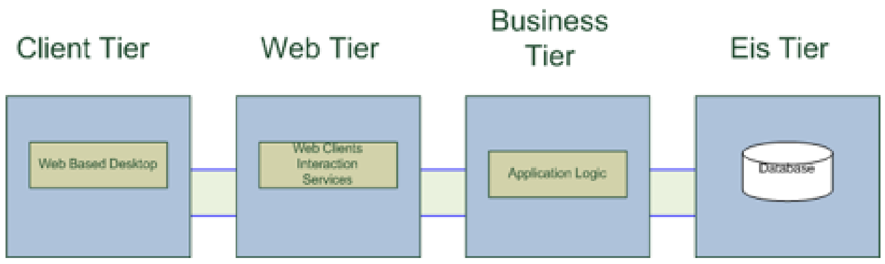
**Internationalization**
Initially the system will support Turkish and English. It should be easily modifiable to extend language support.

#### 2.1.1 Architectural Views

### Logical View

Overview

The Examiners' Board is divided into layers based on the N-Tier architecture



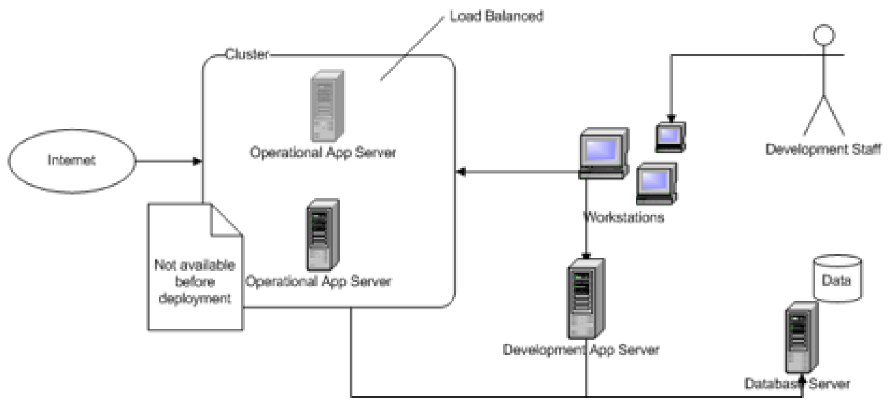
This approach is most commonly adopted for corporate enterprise applications, which demand scalability among other things , modularity and easy maintenance.

- The **web tier** is responsible for the presentation of logic and pages rendering.
- The **business tier** is responsible for the core functionalities of the system
- The **EIS tier** is responsible for storing student data and mark updates.

**Deployment View**

Logical Structure

Physical Structure



**Details**

- Runtime Pattern is applied.
- Development is done on single application server.
- Application is deployed into a single application cloud server.
- Application server is Appache running Application Server
- Database server is Myphp server running MySQL server.
- All application servers and Database server have redundancy using mirroring RAID mode.
- Workstations run Linux and use Vadiin/Eclipse platform for development

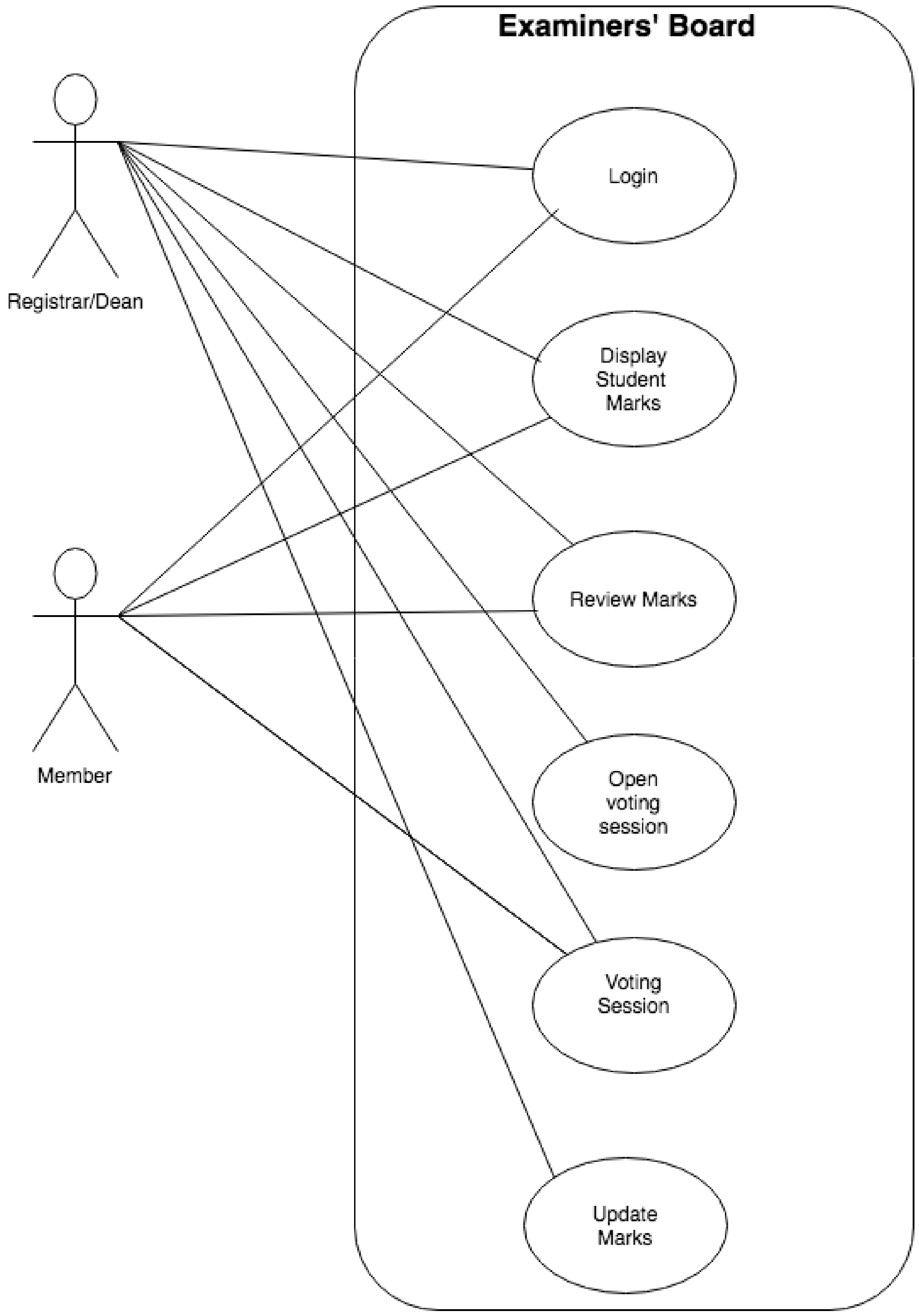
#### 2.1.2 Architectural Design Pattern

#### 2.1.3 Architectural Style

#### 2.1.4 Architectural Process

# 3 Architectural View Decomposition

## 3.1 Use-Case View



#### 3.1.1 Use-Case Realisations

Not the scope of this phase

# 4 Design description organisation

## 4.1 Introduction

The Software Design Description (SDD) details the chosen software architecture and the justification for selecting that architecture. In this project the team was tasked with architecting and implementing an Examiners' Board Meeting Software System.

## 4.2 Design Views

Entity attribute information may be organized in several ways to reveal all of the important aspects of a design. In so doing, the user is able to focus on design details from a different perspective or viewpoint. A design view is a subset of design entity attribute information that is specifically suited to the needs of a software project activity. Each design view represents a separate concern about a software system. Together, these views provide a comprehensive description of the design in a concise and usable form that simplifies information access and assimilation.

### 4.2.1 Decomposition description

#### Scope

Firstly , we need to understand the organisation's business functions before beginning developing information systems. The decomposition descriptions are done in order to plan business functions, processes and sub-processes within "Examiner's Board Meeting" Project

#### Use

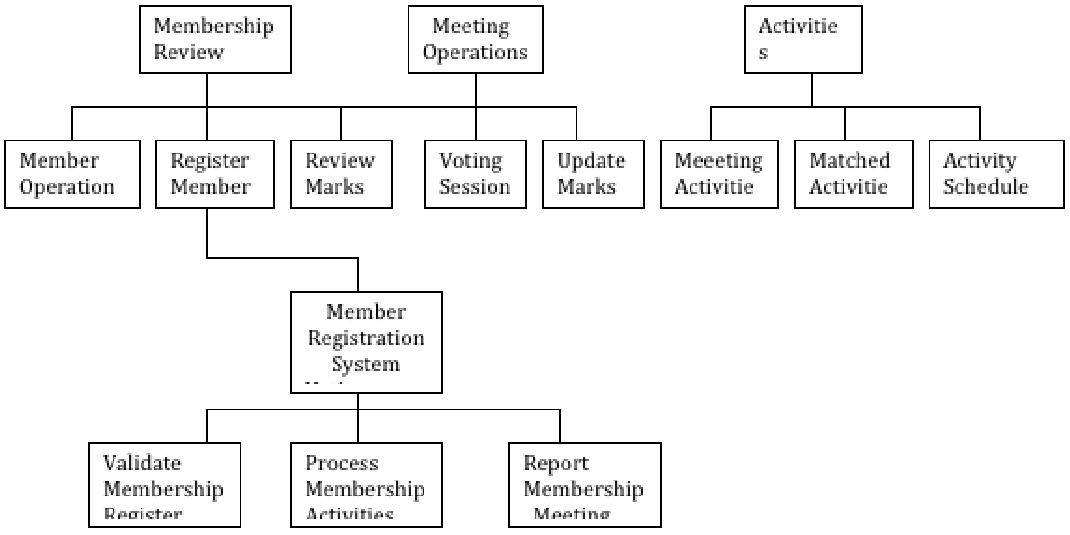
Our project will perform a number of funnctions. Before we plan what system to build for the organisation, it is helpful to first breakdown the functionalities of "Examiners' Board" Project needs to perform.Then it is much easier to identify processes that occur within the business functions, and ultimately the systems that will support those processes. This is a top-down approach to systems development. (As we mentioned we are going to use a Top- Down waterfall like design in earlier stages.)

The process of starting at a high level and moving into smaller and smaller subsystems is called decomposition. The functional decomposition diagram (FDD) is a planning tool for identifying business functions and the processes that comprise them. The diagram is the starting point for more detailed process diagrams, such as data flow diagrams

#### Objectives

- Understand the rules and style guidelines for functional decomposition diagrams(FDDs)
- Understand the process used to create FDDs
  -Be able to create a Functional Decomposition Diagram.

**Representation**



## 4.3 Dependency Description

### 4.3.1 Scope

In our Project many methods are available for visualization and measurement of structure during the design of a programming solution.

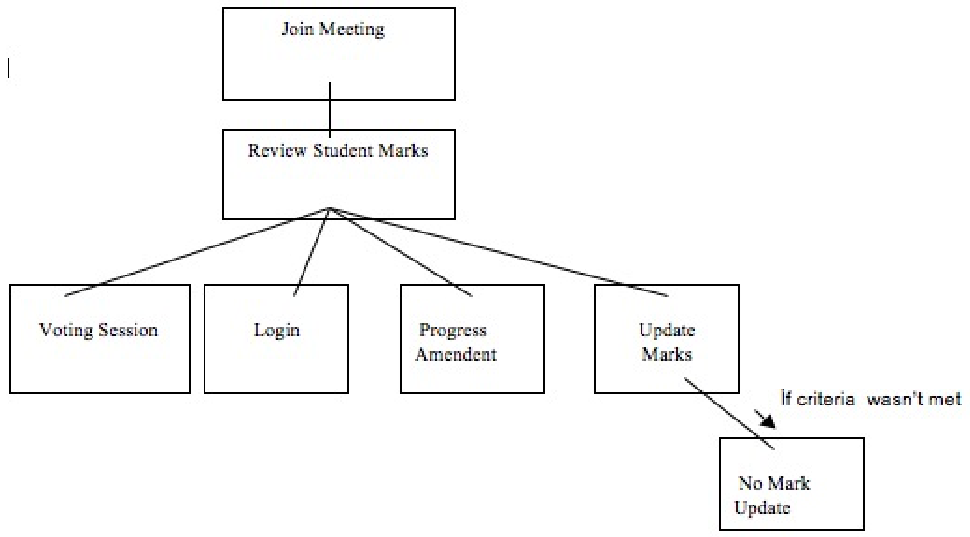
### 4.3.2 Use

In order to avoid the possible problems that may occur due to visualizing the program structure we draw as many diagrams as possible since the diagrams are more clear and easily understandable under any case. As we are working on, adding, moving and deleting pieces as the program developed and problems with the translation from design through pseudo-code to high-level language code were overcome. Over time, we decided that it was most helpful to make the diagram show the relationships between sections of the program.

A decomposition diagram shows a hierarchy between modules but does not show that some modules are dependent on other modules. When one module cannot occur until another module is completed, there is a dependency between the two modules. These dependencies are illustrated on a Dependency Diagram. Just as a decomposition diagram does not show dependencies, a Dependency Diagram does not show program structure.

### 4.3.3 Representation

## Structure Chart



Deployment Model

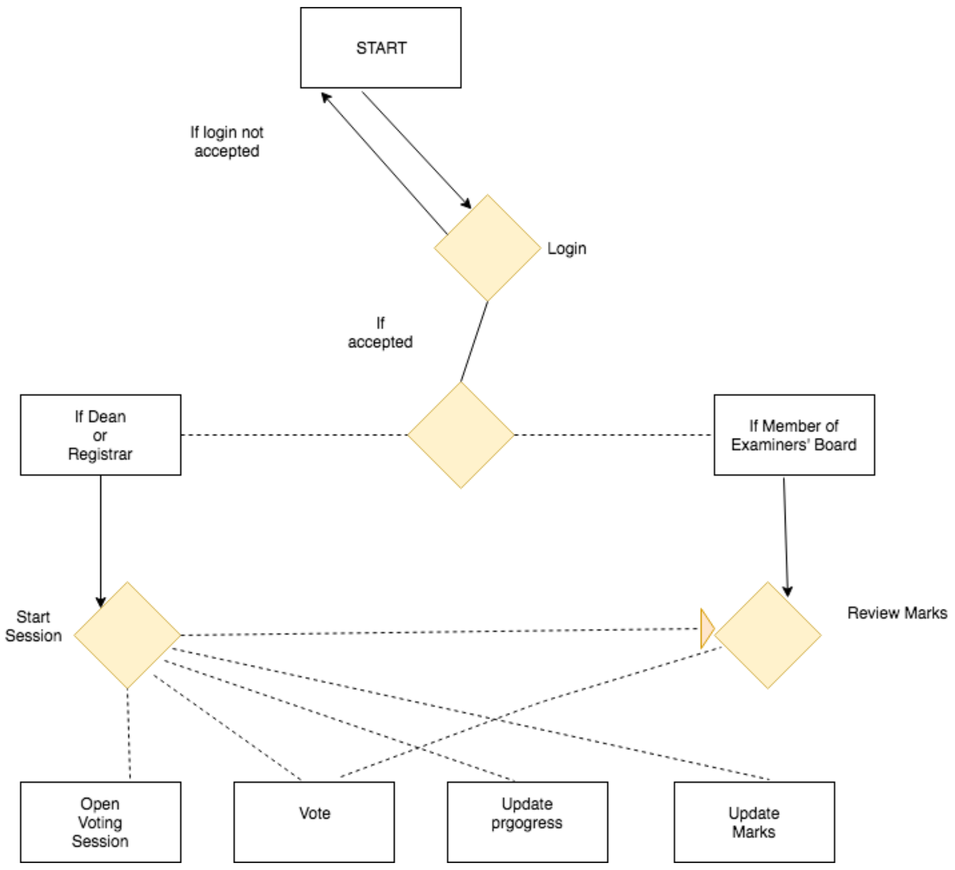


Diagram.png

## 4.4 Interface description

### 4.4.1 Scope

Since our project is going to work through a web site the main screen we met with user is very important.Because if we can attract the user to stay in our site this means the user may be satisfied. The graphic user interface (GUI) of a computer system comprises the interaction metaphors, images, and concepts used to convey function and meaning on the computer screen.

It also includes the detailed visual characteristics of every component of the graphic interface and the functional sequence of interactions over time that produce the characteristic look and feel of Web pages and hypertext linked relations. Graphic design and visual "signature" graphics are not used simply to enliven Web pages — graphics are integral to the user's experience with your site. In interactive documents graphic design cannot be separated from issues of interface design. There are some criteria that we are going to apply to our web site in order to have effective, efficient, easily understandable and usable interface. These are;

**Visibility** of system status
The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
**Match between system and the real world**
The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
**User control and freedom**
Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
**Consistency and standards**
Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
**Error prevention**
Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
**Recognition rather than recall**
Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
**Flexibility and efficiency of use**
Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

**Aesthetic and minimalist design**
Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
**Help users recognize, diagnose, and recover from errors**
Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
**Help and documentation**
Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task , list concrete steps to be carried out , and not to be too large.