## STUDENT MANAGEMENT SYSTEM

#### PROJECT REPORT

**Date: 15/05/2021 Merlin Paul**

**CHELAKKARA**

### ABSTRACT

This project **“STUDENT MANAGEMENT SYSTEM**

technology has been applied throughout the entire colleges or schools, revolutionizing the document management process. With help of this , people no longer need to manually retype important documents when entering them into electronic databases The result is accurate, efficient information processing in less time.

# TABLE OF CONTENTS

1. Abstract
2. Table of content
3. List of Figure
4. Chapters
   1. Introduction
   2. Requirement Analysis

3Testing

1. Maintenance

5 Implementation

5 Design

1. Data Flow Diagrams

**LIST OF FIGURE**

1. DATA FLOW DIAGRAM
   1. Level 0
   2. Level 1

# INTRODUCTION

This project on “Student Management System” is useful for easy user interface. The system utilizes the powerful database management, data retrieval and data manipulation. This project provides more ease for managing the data than manually maintaining in the documents. The project is useful for saving valuable time and reduces the huge paper work.

It will help educational Institutions like schools and colleges will keep track of their student records like personal details, contact details, etc. The Internet is rapidly becoming a part of the everyday lives of a majority of people in the world. People perform various activities on the Internet and one of them is storing their data in data-base where they are interested in. In these data base’s they can post the queries and they can retrieve the required data. Obviously there is a need of Student Information System software for management of student’s data.

There are many departments of administration for the maintenance of college information and student databases in any institution.

All these departments provide various records regarding students. Most of these track records need to maintain information about the students. This information could be the general details like student name, address, performance, attendance etc or specific information related to departments like collection of data.

All the modules in college administration are interdependent. They are maintained manually. So they need to be automated and centralized as, Information from one module will be needed by other modules.

For example when a student needs his course completion certificate it needs to check many details about the student like his name, registration number, year of study, exams he attended and many other details. So it needs to contact all the modules that are office, department and examination and result of students.

# REQUIREMENT ANALYSIS

### Introduction

The following subsections are an overview of the entire Software Requirements Specification (SRS).

#### Purpose

This document provides the technical description of all software requirements of STUDENT MANAGEMENT SYSTEM.

The document will define the product functions, user characteristics, constraints, and specific requirements of the system.

The objective of the software is to maintain information pertaining to the students with the purpose of :-

* Planned approach towards working
* Accuracy
* Reliability
* No Redundancy
* Immediate retrieval of information
* Immediate storage of information
* Easy to Operate

#### Scope

“STUDENT MANAGEMENT SYSTEM” is a project with a mission of viewing and manipulating student information of IIMSR in a Web-based environment. Thus, the overall system will consist of a Student Database System and Web Interface.

The Student Database System will supply the fundamental database structure of the entire system whereas Web Interface will provide a secure Web interface between the users and the database.

The Software aims to create a “paperless office” rather than using a traditional record keeping system.

Although this project is presently being designed specifically for Integral Institute of Medical Sciences and Research but there exist the possibility in future to upgrade it to general level.

The software will not only help the following levels of user in viewing the information but also each user can alongside update changes within their respective access limits.

* + Administrative Level

#### Definition

Paperless Office: refers to an integrated working environment where all the data and documentation is represented in electronic format.

Student Personal Information: refers to personal records of individual students bio- data along with his performance throughout the course.

Traditional Record Keeping System: refers to a manual system where all records are kept on papers by manual in-charge.

#### Overview

This document is prepared in accordance with the IEEE Std, IEEE Recommended Practice for Software Requirements Specifications.

It also provides product perspectives, product functions, user characteristics, general constraints, and assumptions and dependencies of the system.

It will contain functional and performance requirements, design constraints, attributes and external interface requirements for the Software.

### Overall Description

This section describes the general factors that affect software and its requirements. In order to be easily understandable, this part of SRS provides a background for the requirements.

#### Product Perspective

This software is a totally self-contained system. Also it is not dependent of any larger system.

#### System Interfaces

Since this student management system is a standalone system, there is no system interface with any other system.

#### User Interfaces

The interfaces will involve check boxes, combo boxes, text boxes, and radio buttons. The combo boxes and the radio buttons will be used to prevent users from entering wrong type of information. They will also enable fast data entry. Text boxes will be controlled for avoiding invalid and inconsistent data.

Users can use “Tab” key to move cursor on screen items easily.

There will be two types of messages for constructive advice to the users: error and confirmation messages. There will be four types of error messages for application control: input, output, process and database/Web server error messages.

There will be several types of users, and each user will access the screens according to their types after entering their id and passwords. Standard screen format (fixed colors, fonts, background, the page layout, etc.) will be used throughout the interfaces.

The language of the user interfaces will be English.

#### Communication Interfaces

The default communication protocol for data transmission between server and the client is Transmission Control Protocol/ Internet Protocol (TCP/IP). At the upper level Hyper Text Transfer Protocol (HTTP, default port=80) will be used for communication between the web server and client.

#### Memory Constraints

The client computer, which runs the web browser, should have enough physical memory to run this program.

#### Product Functions

* Keep record of students information

#### User Characteristics

Administrator – The administrator will hold full access to view as well as manipulate anywhere in the software and the information.

User- The other user like staff or faculty can view only their individual report and analyze their own performance based on that.

#### Assumptions

Every user will be having the appropriate hardware and software configuration as per the necessary requirement.

### Specific Requirements

#### Functional Requirements-

The software is meant to generate a student id which will provide unique identity to individual students. It is through this student id that each students data can accessed on this platform.The requirements under Student Management System are to maintain information relevant to the following fields:

* **Students Profile**- The full information of each and every student must be maintained in System along with the facility to regularly update it from time to time at regular intervals which will be easily possible through each students unique id.
* **Recording communications with students –**It will help in providing student with updates on latest notices. It will reduce the burden on management as well as of students in manually providing notices of information.

#### Software Product Features

The software feature is to provide easy accessibility to student details on IIMSR to the management of the institute. But it requires authentication of user through login id and password at various different levels for safety of the system.

#### Performance Requirements

The performance of the software will be as smooth as possible with special consideration on the following parameters-

* Planned approach towards working
* Accuracy
* Reliability
* No Redundancy
* Immediate retrieval of information
* Immediate storage of information
* Easy to Operate

#### Design Constraints

* GUI is only in English.
* Login and password is used for identification of user and there is no facility for guest.



#### Software System Attributes

**Reliability**

The system has to operate in a reliable manner with no scope for any flaws. This is to ensure efficient working and processing of information.

#### Availability

The site should be available all the time without any issues. A backup must be available for recovery issues so that the existing is not lost in case of any issue.

#### Security

The system has an authorization mechanism for users to identify their personal profiles. Therefore, different users will have different authorization levels to access the data. Data integrity for critical variables will also be checked.

#### Maintainability

The system can meet the changing requirements easily, since the infrastructure of the system would not need major changes. The requirements of the software while evolving, will be met by just adding new sub-functions. Therefore, the maintainability of the system would not be a complex issue.

#### Portability

All of the code which will be deployed at the web server will be written in ASP.Net 4.5 and using SQL Server for database storage. So, using IIS(Internet Information Server) the software will remain portable.

# TESTING

**Software testing** is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to the process of executing a program or application with the intent of finding software bugs(errors or other defects).

Software testing can be stated as the process of validating and verifying that a computer program/application/product:

* meets the requirements that guided its design and development,
* works as expected,
* can be implemented with the same characteristics,
* and satisfies the needs of stakeholders.

#### Static vs. dynamic testing

There are many approaches to software testing. Reviews, walkthroughs, or inspections are referred to as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. Static testing is often implicit, as proofreading, plus when programming tools/text editors check source code structure or compilers (pre-compilers) check syntax and data flow as static program analysis. Dynamic testing takes place when the program itself is run. Dynamic testing may begin before the program is 100% complete in order to test particular sections of code and are applied to discrete functions or modules. Typical techniques for this are either using stubs/drivers or execution from a debugger environment.

#### The box approach

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

#### White-Box testing

**White-box testing** (also known as **clear box testing**, **glass box testing**, **transparent box testing** and **structural testing**) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in‐circuit testing (ICT).

#### Black-box testing

**Black-box testing** treats the software as a "black box", examining functionality without any knowledge of internal implementation. The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods include:equivalence partitioning, boundary value analysis, all‐pairs testing, state transition tables, decision table testing, fuzz testing, model‐based testing, use case testing,exploratory testing and specification-based testing.

**Specification-based testing** aims to test the functionality of software according to the applicable requirements. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications

and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non‐functional, though usually functional.

Ad hoc testing and exploratory testing are important methodologies for checking software integrity, because they require less preparation time to implement, while the important bugs can be found quickly. In adhoc testing, where testing takes place in an improvised, impromptu way, the ability of a test tool to visually record everything that occurs on a system becomes very important

#### Grey-box testing

**Grey-box testing** involves having knowledge of internal data structures and algorithms for purposes of designing tests, while executing those tests at the user, or black-box level. The tester is not required to have full access to the software's source code. Manipulating input data and formatting output do not qualify as grey- box, because the input and output are clearly outside of the "black box" that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for test.

# TESTING LEVELS

There are generally four recognized levels of tests: unit testing, integration testing, system testing, and acceptance testing. Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

Unit Testing

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white- box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

#### Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

#### Component interface testing

The practice of component interface testing can be used to check the handling of data passed between various units, or subsystem components, beyond full integration testing between those units. The data being passed can be considered as "message packets" and the range or data types can be checked, for data generated from one unit, and tested for validity before being passed into another unit. One option for interface testing is to keep a separate log file of data items being passed, often with a timestamp logged to allow analysis of thousands of cases of data passed between units for days or weeks.

#### System testing

System testing, or end-to-end testing, tests a completely integrated system to verify that it meets its requirements. For example, a system test might involve testing a log-on interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then log-off.

#### Acceptance testing

At last the system is delivered to the user for Acceptance testing.

#### Regression testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, as degraded or lost features, including old bugs that have come back. Such regressions occur whenever software functionality that was previously working, correctly, stops working as intended. Typically, regressions occur as an unintended consequence of program changes, when the newly developed part of the software collides with the previously existing code. Common methods of regression testing include re- running previous sets of test-cases and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the risk of the added features.

#### Alpha testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing

#### Beta testing

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

#### Functional vs non-functional testing

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work."

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or other performance, behavior under certain constraints, or security. Testing will determine the breaking point, the point at which extremes of scalability or performance leads to unstable execution. Non-functional requirements tend to be those that reflect the quality of the product, particularly in the context of the suitability perspective of its users

#### Top-down and bottom-up

**Bottom Up Testing** is an approach to integrated testing where the lowest level components (modules, procedures, and functions) are tested first, then integrated and used to facilitate the testing of higher level components. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. The process is repeated until the components at the top of the hierarchy are tested. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

**Top Down Testing** is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

In both, method stubs and drivers are used to stand-in for missing components and are replaced as the levels are completed.

# MAINTENANCE

**Software maintenance** in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes.

A common perception of maintenance is that it merely involves fixing defects. However, one study indicated that the majority, over 80%, of the maintenance effort is used for non-corrective actions. This perception is perpetuated by users submitting problem reports that in reality are functionality enhancements to the system. More recent studies put the bug-fixing proportion closer to 21%.

Software maintenance is a very broad activity that includes error correction, enhancements of capabilities, deletion of obsolete capabilities, and optimization. Because change is inevitable, mechanisms must be developed for evaluation, controlling and making modifications.

An integral part of software is the maintenance one, which requires an accurate maintenance plan to be prepared during the software development. It should specify how users will request modifications or report problems. The budget should include resource and cost estimates. A new decision should be addressed for the developing of every new system feature and its quality objectives. The software maintenance, which can last for 5–6 years (or even decades) after the development process, calls for an effective plan which can address the scope of software maintenance, the tailoring of the post delivery/deployment process, the designation of who will provide maintenance, and an estimate of the life-cycle costs. The selection of proper enforcement of standards is the challenging task right from early stage of software engineering which has not got definite importance by the concerned stakeholders.

# IMPLEMENTATION

A **product software implementation method** is a systematically structured approach to effectively integrate a software based service or component into the work flow of an organizational structure or an individual end-user.

This entry focuses on the process modeling (Process Modeling) side of the implementation of “large” (explained in complexity differences) product software, using the implementation of Enterprise Resource Planning systems as the main example to elaborate on.

A product software implementation method is a blueprint to get users and/or organizations running with a specific software product.

The method is a set of rules and views to cope with the most common issues that occur when implementing a software product: business alignment from the organizational view and acceptance from human view.

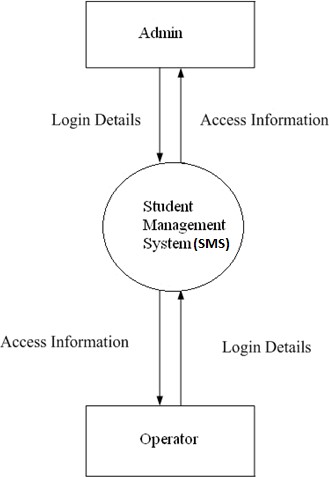
The implementation of product software, as the final link in the deployment chain of software production, is in a financial perspective of a major issue.

It is stated that the implementation of (product) software consumes up to 1/3 of the budget of a software purchase (more than hardware and software requirements together).

# DESIGN

### DATA FLOW DIAGRAM(DFD):

**LEVEL 0 DFD:**



**LEVEL 1 DFD:**

