This lab deals with the analysis and design of a software problem .the tool used in a lab is rational rose .this tool is used for a object oriented design of a problem . We draw a uml diagram in a rational rose which deals with the objects and classes in a system .The **Unified Modeling Language** or **UML** is is a mostly graphical modelling language that is used to express designs. It is a standardized language in which to specify the artefacts and components of a software system. It is important to understand that the UML describes a notation and not a process. It does not put forth a single method or process of design, but rather is a standardized tool that can be used in a design process.

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.1 The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

**Goals of UML**

The primary goals in the design of the UML were:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.

2. Provide extensibility and specialization mechanisms to extend the core concepts.

3. Be independent of particular programming languages and development processes.

4. Provide a formal basis for understanding the modeling language.

5. Encourage the growth of the OO tools market.

6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.

7. Integrate best practices.

**Why Use UML?**

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance.

**PRACTICAL PROJECTS FOR THE LAB**

* 1. Student Result Management System
  2. Library management system
  3. Inventory control system
  4. Accounting system for a Shop
  5. Fast food billing system for a restaurant
  6. Zakat Donation system
  7. Blood bank system
  8. Rickshaw garage system
  9. Classroom attendance system
  10. Video library management system
  11. Hotel management system
  12. Hall management system
  13. E-ticking for park
  14. Online book-selling system
  15. File management system
  16. Office resource management system
  17. Class routine management system
  18. Apartment management system
  19. Car parking management system
  20. Canteen management system
  21. Event management system
  22. Office management system
  23. Student Project monitoring system
  24. Training program management system
  25. Student bus scheduling system

**LAB SYLLABUS**

**Prepare the following documents and develop the software project startup, prototype**

**model, using software engineering methodology for at least two real time scenarios or**

**for the sample experiments.**

•Problem Analysis and Project Planning -Thorough study of the problem–Identify Project

scope, Objectives and Infrastructure.

•Software Requirement Analysis –Describe the individual Phases/modules of the project and

Identify deliverables. Identify functional and non-functional requirements.

•Data Modeling –Use work products –data dictionary.

•Software Designing -Develop use case diagrams and activity diagrams, build and test class

diagrams, sequence diagrams and add interface to class diagrams.

•Prototype model –Develop the prototype of the product.

The SRS and prototype model should be submitted for end semester examination.

**List of Sample Experiments:**

**1. Course management system (CMS)**

A course management system (CMS) is a collection of software tools providing an online environment for course interactions. A CMS typically includes a variety of online tools and environments, such as:

•An area for faculty posting of class materials such as course syllabus and handouts

•An area for student posting of papers and other assignments

•A grade book where faculty can record grades and each student can view his or her grades

•An integrated email tool allowing participants to send announcement email messages to the entire class or to a subset of the entire class

•A chat tool allowing synchronous communication among class participants

•A threaded discussion board allowing asynchronous communication among participants.

In addition, a CMS is typically integrated with other databases in the university so that students enrolled in a particular course are automatically registered in the CMS as participants in that course. The Course Management System (CMS) is a web application for department personnel, Academic Senate, and Registrar staff to view, enter, and manage course information formerly Submitted via paper. Departments can use CMS to create new course proposals, submit changes for existing courses, and track the progress of proposals as they move through the stages of online approval.

**2. Easy Leave**

This project is aimed at developing a web based Leave Management Tool, which is of importance to either an organization or a college. The Easy Leave is an Intranet based application that can be accessed throughout the Organization or a specified group/Dept. This system can be used to automate the workflow of leave applications and their approvals. The periodic crediting of leave is also automated. There are features like notifications, cancellation of leave, automatic approval of leave, report generators etc in this Tool.

Functional components of the project:

There are registered people in the system. Some are approvers. An approver can also be a Requestor. In an organization, the hierarchy could be Engineers/Managers/Business Managers/Managing Director etc. In a college, it could be Lecturer/Professor/Head of the Department/Dean/Principal etc.

Following is a list of functionalities of the system: A person should be able to

•login to the system through the first page of the application

•change the password after logging into the system

•see his/her eligibility details (like how many days of leave he/she is eligible for etc)

•query the leave balance

•see his/her leave history since the time he/she joined the company/college

•apply for leave, specifying the from and to dates, reason for taking leave, address for communication while on leave and his/her superior's email id

•see his/her current leave applications and the leave applications that are submitted to him/her for approval or cancellation

•approve/reject the leave applications that are submitted to him/her

•withdraw his/her leave application (which has not been approved yet)

•Cancel his/her leave (which has been already approved). This will need to be approved by his/her Superior

•get help about the leave system on how to use the different features of the system

•As soon as a leave application /cancellation request /withdrawal /approval /rejection /password-change is made by the person, an automatic email should be sent to the person and his superior giving details about the action

•The number of days of leave (as per the assumed leave policy) should be automatically credited to everybody and a notification regarding the same be sent to them automatically

•An automatic leave-approval facility for leave applications which are older than 2 weeks should be there. Notification about the automatic leave approval should be sent to the person as well as his superior

# Institute of Information Technology

# Jahangirnagar University

****

SUBMITTED BY

### Name:

#### Class Roll No.

#### 3rd Year 2nd Semester in ICT (Hons.) 2020-2021

**Submission Date: Submitted to:**

**Dr. Shamim Al Mamun, Associate Professor, IIT-JU**

**INDEX**

**Exercises/Assignments Page No.**

1. Exercise – 1
   1. Task Statement1… 1
   2. Task Completion procedure 2 2
   3. Conclusion, etc.

2. Exercise – 2 ……………………………………………………. ..

3. Exercise – 3 ………………………

1. etc.
2. etc.

# Institute of Information Technology

# Jahangirnagar University

****

### LAB Exercise #1: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#### Software Engineering Lab Exercise Report Submitted By:

**Name:**

**Class Roll No:**

Lab Dates: Submission Date:

**LAB EXERCISES:**

1. Software Specification document (SRS) [Word and Excel]
2. Project Management Tools [Trello board]
3. Flow chart [Draw.io https://app.diagrams.net/]
4. Data Dictionary and Class diagram
5. Use Case diagram
6. Sequence diagram
7. Activity diagram
8. Prototype modeling [FIGMA]
9. Implementation
10. Software Testing [Selenium]
11. Milestone Project Showcasing [FINAL EXAM]

**EXCERCISE NO. 1**

**AIM:** To prepare PROBLEM STATEMENT for any project.

**REQUIREMENTS:**

**Hardware Interfaces**

Pentium(R) 4 CPU 2.26 GHz, 128 MB RAM

Screen resolution of at least 800 x 600 required for proper and complete viewing of screens. Higher resolution would not be a problem.

CD ROM Driver

**Software Interfaces**

Any window-based operating system (Windows 95/98/2000/XP/NT)

WordPad or Microsoft Word

**THEORY:**

The problem statement is the initial starting point for a project. It is basically a one to three page statement that everyone on the project agrees with that describes what will be done at a high level. The problem statement is intended for a broad audience and should be written in non-technical terms. It helps the non-technical and technical personnel communicate by providing a description of a problem. It doesn't describe the solution to the problem.

The input to requirement engineering is the problem statement prepared by customer.

It may give an overview of the existing system along with broad expectations from the new system.

The first phase of requirements engineering begins with requirements elicitation i.e. gathering of information about requirements. Here, requirements are identified with the help of customer and existing system processes. So from here begins the preparation of problem statement.

So, basically a problem statement describes **what** needs to be done without describing **how**.

**Conclusion:** The problem statement was written successfully by following the steps described above.

Problem statement 5WH model to write the statement

**What**? Define the problem.

**Why**? Reason for the problem's occurrence.

**When**? When the problem began or was first noticed.

**Where**? Place of the problem's first occurrence or sighting.

**Who**? The person or thing that the problem affects.

**How**? The sequence of events that resulted in the problem.

**How To Write A Problem Statement?**

A problem statement needs to communicate the extent of an issue and the resources you require to solve the problem. Thus, it is required to be accurate and clearly written. Here are the key steps you are required to follow closely when crafting a problem statement:

1. Put the problem in the context

Before you write, you require to have a clear picture of the problem itself. Since you are addressing individuals with no background on the topic, you require to provide adequate context for them to understand the scope and urgency of the issue. In other words, you are required to describe how things should work.

At this stage, you want to discuss the what, where, when and who, regarding the problem. Think about these additional questions to help you frame an impactful problem statement:

Is the problem limited to a certain demographic or region?

Which people have attempted to solve the issue?

What do people already know about the issue?

Related: The 8D Problem-Solving Method: What It Is And How To Use It

2. Explain the relevance of the problem

The problem statement is required to address not only what the problem is but why it is important to solve. This is the part where you grab the reader's attention by specifying the seriousness of the issue. You require to communicate why it matters to them and the potential dangers if it goes unsolved. Sometimes a problem's relevance is not immediately obvious; that is why doing your due diligence in defining and describing the problem accurately is an important first step.

To clarify why your problem is significant, address the following questions:

Who would feel the consequences of the problem?

What is the financial impact of the problem?

Does the problem have any relevance to other areas of the business?

Does the problem impact the wider society?

How would solving the problem increase our understanding of the business?

Related: How To Use Deductive Reasoning

3. Backup your claims

Quantifiable data ensures people understand the relevance and scope of a problem. Using evidence to back up your problem statement would make people take the issue seriously. Knowing your numbers also boosts your credibility. It can also mobilise a faster response to the problem. For example, if a problem has diminished sales in the past few weeks, showing these statistics immediately communicates the extent of the issue.

4. Propose a solution

After doing a thorough investigation into the problem, you would have a solid grasp of how the problem occurred. Thus, you require to propose a practical solution or suggest several approaches to understanding and rectifying the issue at this stage. State your objectives by suggesting well-thought-out plans for combating the issue.

5. Explain the benefits of your proposed solution(s)

Demonstrate why the solution would work with practical examples of how it might effectively address the problem. Explain how solving the problem would benefit the organisation. Focus on the financial benefits of solving the problem and the impact on customer satisfaction.

**EXCERCISE NO. 2**

**Aim**: Understanding an SRS.

**Requirements:**

**Hardware Requirements:**

 PC with 300 megahertz or higher processor clock speed recommended; 233 MHz minimum required.

 128 megabytes (MB) of RAM or higher recommended (64 MB minimum supported)

 1.5 gigabytes (GB) of available hard disk space

 CD ROM or DVD Drive

 Keyboard and Mouse(compatible pointing device).

**Software Requirements:**

Rational Rose, Windows XP,

**Theory:**

An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It's a two-way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.

The SRS document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an eCommerce Web site, and so on) must provide, as well as states any required constraints by which the system must abide. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.

It's important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be. A well-designed, well-written SRS accomplishes four major goals:

 It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language, explained later in this article), in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.

 It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.

 It serves as an input to the design specification. As mentioned previously, the SRS serves as the parent document to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.

 It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

SRSs are typically developed during the first stages of "Requirements Development," which is the initial product development phase in which information is gathered about what requirements are needed--and not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a return-on-investment (ROI) analysis or needs analysis of the customer or client's current business environment. The actual specification, then, is written after the requirements have been gathered and analyzed.

**SRS should address the following**

The basic issues that the SRS shall address are the following:

a) **Functionality**. What is the software supposed to do?

b) **External interfaces**. How does the software interact with people, the system‘s hardware, other hardware, and other software?

c) **Performance**. What is the speed, availability, response time, recovery time of various software functions, etc.?

d) **Attributes**. What are the portability, correctness, maintainability, security, etc. considerations?

e) **Design constraints imposed on an implementation**. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.? **Chracteristics of a good SRS**

An SRS should be

a) Correct

b) Unambiguous

c) Complete

d) Consistent

e) Ranked for importance and/or stability

f) Verifiable

g) Modifiable

h) Traceable

**Correct** - This is like motherhood and apple pie. Of course you want the specification to be correct. No one writes a specification that they know is incorrect. We like to say - "Correct and Ever Correcting." The discipline is keeping the specification up to date when you find things that are not correct.

**Unambiguous -** An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. Again, easier said than done. Spending time on this area prior to releasing the SRS can be a waste of time. But as you find ambiguities - fix them.

**Complete -** A simple judge of this is that is should be all that is needed by the software designers to create the software.

**Consistent -** The SRS should be consistent within itself and consistent to its reference documents. If you call an input "Start and Stop" in one place, don't call it "Start/Stop" in another.

**Ranked for Importance -** Very often a new system has requirements that are really marketing wish lists. Some may not be achievable. It is useful provide this information in the SRS.

**Verifiable -** Don't put in requirements like - "It should provide the user a fast response." Another of my favorites is - "The system should never crash." Instead, provide a quantitative requirement like: "Every key stroke should provide a user response within 100 milliseconds."

**Modifiable -** Having the same requirement in more than one place may not be wrong - but tends to make the document not maintainable.

**Traceable -** Often, this is not important in a non-politicized environment. However, in most organizations, it is sometimes useful to connect the requirements in the SRS to a higher level document. Why do we need this requirement?

**A sample of basic SRS Outline**

**1. Introduction** 1.1 Purpose 1.2 Document conventions 1.3 Intended audience 1.4 Additional information 1.5 Contact information/SRS team members 1.6 References

**2. Overall Description** 2.1 Product perspective 2.2 Product functions 2.3 User classes and characteristics 2.4 Operating environment 2.5 User environment 2.6 Design/implementation constraints 2.7 Assumptions and dependencies

**3. External Interface Requirements** 3.1 User interfaces 3.2 Hardware interfaces 3.3 Software interfaces 3.4 Communication protocols and interfaces

**4. System Features** 4.1 System feature A 4.1.1 Description and priority 4.1.2 Action/result 4.1.3 Functional requirements 4.2 System feature B

**5. Other Nonfunctional Requirements** 5.1 Performance requirements 5.2 Safety requirements 5.3 Security requirements 5.4 Software quality attributes 5.5 Project documentation 5.6 User documentation

**6. Other Requirements** Appendix A: Terminology/Glossary/Definitions list Appendix B: To be determined

**Conclusion:** The SRS was made successfully by following the steps described above.