D Y PATIL
RAMRAO ADIK INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

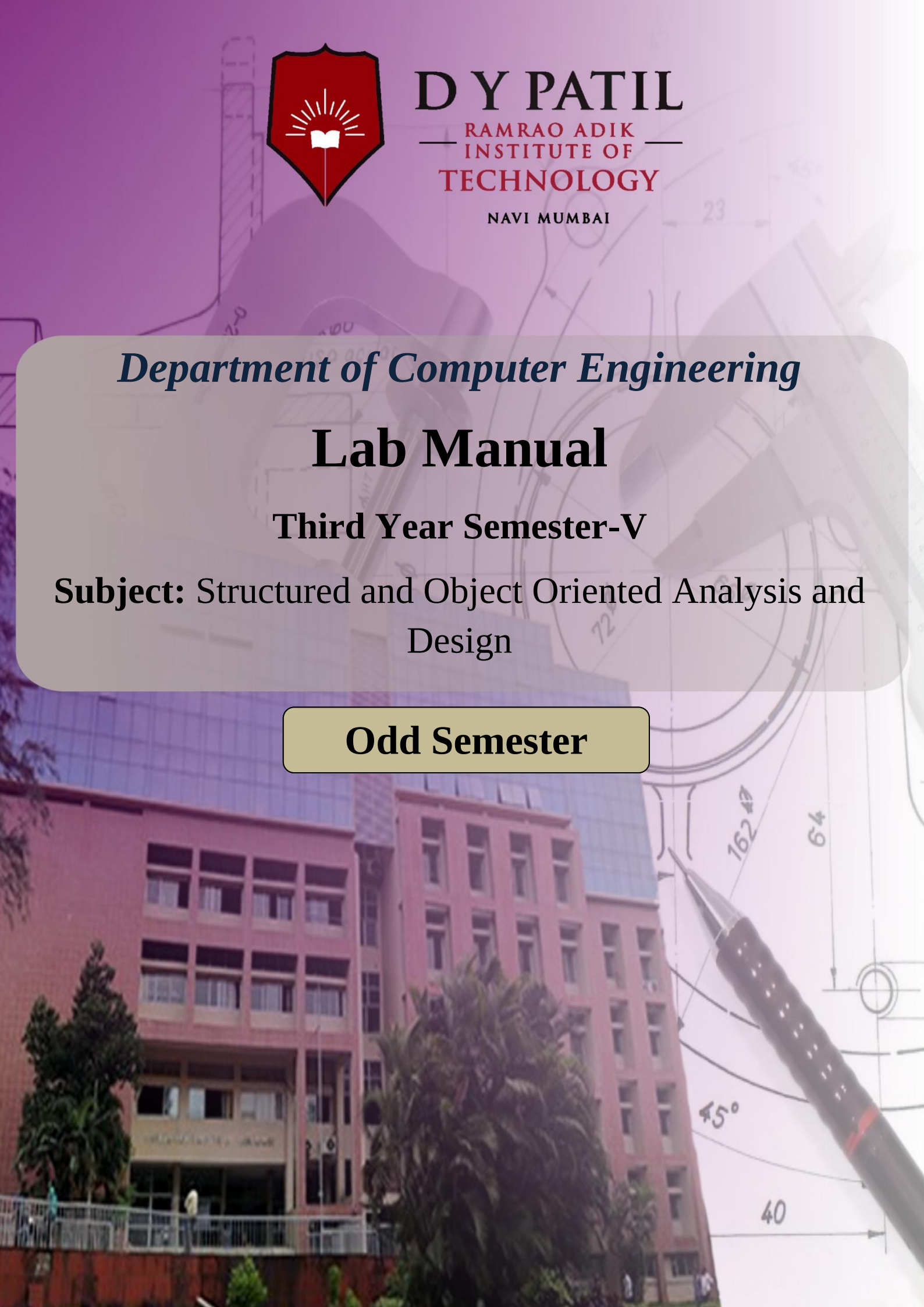
Department of Computer Engineering

Lab Manual

Third Year Semester-V

Subject: Structured and Object Oriented Analysis and Design

Odd Semester



Institutional Vision, Mission and Quality Policy

Our Vision

To foster and permeate higher and quality education with value added engineering, technology programs, providing all facilities in terms of technology and platforms for all round development with societal awareness and nurture the youth with international competencies and exemplary level of employability even under highly competitive environment so that they are innovative adaptable and capable of handling problems faced by our country and world at large.

RAIT's firm belief in new form of engineering education that lays equal stress on academics and leadership building extracurricular skills has been a major contribution to the success of RAIT as one of the most reputed institution of higher learning. The challenges faced by our country and world in the 21 Century needs a whole new range of thought and action leaders, which a conventional educational system in engineering disciplines are ill equipped to produce. Our reputation in providing good engineering education with additional life skills ensure that high grade and highly motivated students join us. Our laboratories and practical sessions reflect the latest that is being followed in the Industry. The project works and summer projects make our students adept at handling the real life problems and be Industry ready. Our students are well placed in the Industry and their performance makes reputed companies visit us with renewed demands and vigour.

Our Mission

The Institution is committed to mobilize the resources and equip itself with men and materials of excellence thereby ensuring that the Institution becomes pivotal center of service to Industry, academia, and society with the latest technology. RAIT engages different platforms such as technology enhancing Student Technical Societies, Cultural platforms, Sports excellence centers, Entrepreneurial Development Center and Societal Interaction Cell. To develop the college to become an autonomous Institution & deemed university at the earliest with facilities for advanced research and development programs on par with international standards. To invite international and reputed national Institutions and Universities to collaborate with our institution on the issues of common interest of teaching and learning sophistication.

RAIT's Mission is to produce engineering and technology professionals who are innovative and inspiring thought leaders, adept at solving problems faced by our nation and world by providing quality education.

The Institute is working closely with all stake holders like industry, academia to foster knowledge generation, acquisition, dissemination using best available resources to address the great challenges being faced by our country and World. RAIT is fully dedicated to provide its students skills that make them leaders and solution providers and are Industry ready when they graduate from the Institution.

We at RAIT assure our main stakeholders of students 100% quality for the programmes we deliver. This quality assurance stems from the teaching and learning processes we have at work at our campus and the teachers who are handpicked from reputed institutions IIT/NIT/MU, etc. and they inspire the students to be innovative in thinking and practical in approach. We have installed internal procedures to better skills set of instructors by sending them to training courses, workshops, seminars and conferences. We have also a full fledged course curriculum and deliveries planned in advance for a structured semester long programme. We have well developed feedback system employers, alumni, students and parents from to fine tune Learning and Teaching processes. These tools help us to ensure same quality of teaching independent of any individual instructor. Each classroom is equipped with Internet and other digital learning resources.

The effective learning process in the campus comprises a clean and stimulating classroom environment and availability of lecture notes and digital resources prepared by instructor from the comfort of home. In addition student is provided with good number of assignments that would trigger his thinking process. The testing process involves an objective test paper that would gauge the understanding of concepts by the students. The quality assurance process also ensures that the learning process is effective. The summer internships and project work based training ensure learning process to include practical and industry relevant aspects. Various technical events, seminars and conferences make the student learning complete.

Our Quality Policy

ज्ञानधीनं जगत् सर्वम।

Knowledge is supreme.

Our Quality Policy

It is our earnest endeavour to produce high quality engineering professionals who are innovative and inspiring, thought and action leaders, competent to solve problems faced by society, nation and world at large by striving towards very high standards in learning, teaching and training methodologies.

Our Motto: If it is not of quality, it is NOT RAIT!

Departmental Vision, Mission

Vision

To impart higher and quality education in computer science with value added engineering and technology programs to prepare technically sound, ethically strong engineers with social awareness. To extend the facilities, to meet the fast changing requirements and nurture the youths with international competencies and exemplary level of employability and research under highly competitive environments.

Mission

To mobilize the resources and equip the institution with men and materials of excellence to provide knowledge and develop technologies in the thrust areas of computer science and Engineering. To provide the diverse platforms of sports, technical, cocurricular and extracurricular activities for the overall development of student with ethical attitude. To prepare the students to sustain the impact of computer education for social needs encompassing industry, educational institutions and public service. To collaborate with IITs, reputed universities and industries for the technical and overall upliftment of students for continuing learning and entrepreneurship.

Departmental Program Specific Objectives (PSOs)

1. To build competencies towards problem solving with an ability to understand, identify, analyze and design the problem, implement and validate the solution including both hardware and software.
2. To build appreciation and knowledge acquiring of current computer techniques with an ability to use skills and tools necessary for computing practice.
3. To be able to match the industry requirements in the area of computer science and engineering. To equip skills to adopt and imbibe new technologies.
4. **Techno-leader**

   To provide exposure to emerging cutting edge technologies, adequate training & opportunities to work as teams on multidisciplinary projects with effective communication skills and leadership qualities.

5. **Practice citizenship**

   To provide knowledge of professional and ethical responsibility and to contribute to society through active engagement with professional societies, schools, civic organizations or other community activities.

6. **Clarify Purpose and Perspective**

   To provide strong in-depth education through electives and to promote student awareness on the life-long learning to adapt to innovation and change, and to be successful in their professional work or graduate studies.

Departmental Program Outcomes (POs)

Pa Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Pb**.** Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

Pc**.** Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

Pd. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

Pe. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

Pf**.** The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

Pg**.** Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

Ph. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Pi. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Pj. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Pk. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Pl. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Index

List of Experiments

Sr. No.	Experiments Name
1	To Study the phases of the System Development Life Cycle(SDLC).
2	To develop Requirement specification document of the selected / allotted project.
3	To develop ERD and DFD model (level-0, level-1 DFD and Data dictionary) of selected / allotted project.
4	To study and draw Class diagram for selected / allotted project.
5	To study and draw Use case diagram for selected / allotted project.
6	To study and draw Activity diagram for selected / allotted project.
7	To study and draw State Transition diagram for selected / allotted project.
8	To study and draw Collaboration diagram / Communication Diagram for selected / allotted project.
9	To study and draw Sequence diagram for selected / allotted project.
10	To study and draw Component diagram for selected / allotted project.
11	To study and draw Deployment diagram for selected / allotted project
12	To develop the prototype of project selected / allotted project.

Course Objectives, Course Outcome & Experiment Plan

Course Objectives:

1.	Give balanced exposure to both traditional and object oriented approaches to system analysis & design.
2.	To explore an object-oriented analysis and design (OOAD) methodology to the requirements, analysis and design phases of a software lifecycle.
3.	Become conversant with the Unified Language (UML) notation and symbols.
4.	To familiarize with the benefits and liabilities of a small subset of architectural and design patterns.

Course Outcomes:

CO1	To familiarize with the benefits and liabilities of a small subset of architectural and design patterns.
CO2	Understand and apply techniques to get the system requirements and present it in standard format.
CO3	Construct the candidate system following design methodology.
CO4	Apply key concepts to both the traditional structured approach and the Object Oriented approach.
CO5	Apply UML notation to construct and graphically present various diagrams for Object-Oriented systems analysis.
CO6	Understanding of Architecture Strategies for information System.

Module No.	Week No.	Experiments Name	Course Outcome	Weightage
1	W1	To Study the phases of the software development life cycle.	CO1	5
2	W2	Develop Requirement specification document of the selected / allotted project.	CO2	10
3	W3	Develop ERD and DFD and model (level-0, level-1 DFD and Data dictionary) of selected / allotted project.	CO3	10
4	W4	Develop Class diagram selected / allotted project.	CO5	4
5	W5	Develop Use case diagram for selected / allotted project.	CO4	4
6	W6	Develop Activity diagram for selected / allotted project	CO4	3
7	W7	Develop State Transition diagram for selected / allotted project	CO4	3
8	W8	Develop Collaboration diagram / Communication Diagram for selected / allotted project	CO5	3
9	W9	Develop Sequence diagram for selected / allotted project.	CO5	3
10	W10	Draw Component diagram for selected / allotted project.	CO6	5
11	W11	Draw Deployment diagram for selected / allotted project	CO6	5
12	W12	To study the prototype of project selected / allotted project.	CO1	5

Mapping Course Outcomes (CO) - Program Outcomes (PO)

	Course Outcomes	Pa	Pb	Pc	Pd	Pe	Pf	Pg	Ph	Pi	Pj	Pk	Pl
PRATICAL /ORAL + TERM WORK 40%	CO1:Understanding concepts of system overview, terms concerning software lifecycles and phase of the system development.	2	1		2		1	1		1			2
	CO2:Understand and apply techniques to get the system requirements and present it in standard format.	1	1	2	1	3	1				1		
	CO3:Construct the candidate system following design methodology.	1	1	3	1	1		1			1		1
	CO4:Apply key modelling concepts to both the traditional structured approach and the Object Oriented approach.			3		1	1	1	1		1	1	1
	CO5:Apply UML notation to construct and graphically present various diagrams for Object- Oriented systems analysis.	1		2		3			1	1		1	1
	CO6:Understanding of architecture strategies for information System.			1	2		2			1	1	2	1

Study and Evaluation Scheme

Course Code	Course Name	Teaching Scheme			Credits Assigned			
		Theory	Practical	Tutorial	Theory	Practical	Tutorial	Total
CPC503	Structured and Object Oriented Analysis and Design	4	2	--	4	01	--	05

Course Code	Course Name	Examination Scheme		
		Term Work	Oral	Total
CPC503	Structured and Object Oriented Analysis and Design	25	25	50

Term Work:

1. Term work consists of at least 10 experiments.
2. The final certification and acceptance of term work ensures satisfactory performance of laboratory work and minimum passing marks in term work.

Practical & Oral:

1. Oral exam will be based on the entire syllabus of Structured and Object Oriented Analysis and design.

Structured and Object Oriented Analysis and Design

Experiment No. : 1

System Development Life Cycle (SDLC)

Experiment No. 1

1. **Aim:** To Study the phases of System Development Life Cycle (SDLC).

2. **Objectives:** From this experiment, the student will be able to
   - Get balanced exposure to both traditional and object oriented approaches to system analysis & design.
   - To analyze the need for the experiment.
   - To understand the importance of this experiment from application point of view

3. **Outcomes:** The learner will be able to
   - To understand, identify, analyze and design the real world problem.
   - Validate the solution using software.
   - Formulate and solve the engineering problems.

4. **Hardware / Software Required:** Rational Software Architect tool.

5. **Theory:**

   A software development process is a structure imposed on the development of a software product. Synonyms include software life cycle and software process. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

   System Development Life Cycle Model

   A system life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes or for building empirically grounded prescriptive models.

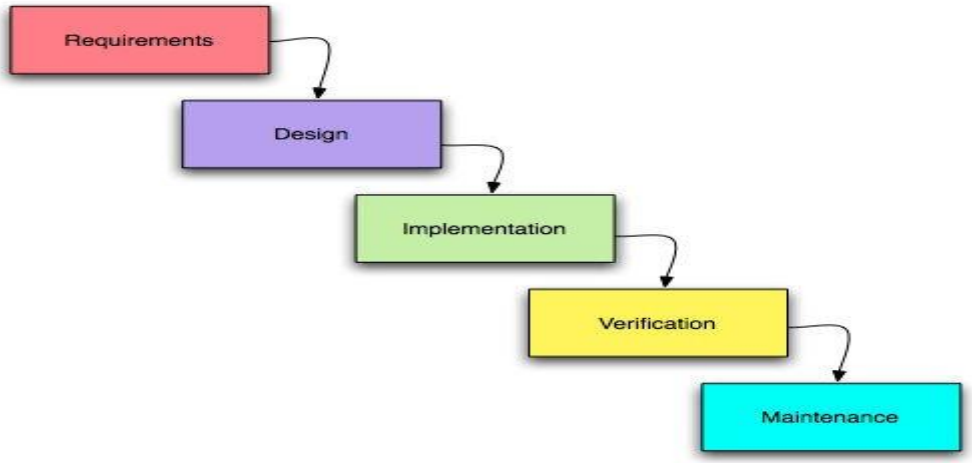


Figure: System Development Life Cycle

Requirement Analysis

The important task in creating a software product is extracting the requirements or requirements analysis. Customers typically have an abstract idea of what they want as an end result, but not what software should do. Incomplete, ambiguous, or even contradictory requirements are recognized by skilled and experienced software engineers at this point. Frequently demonstrating live code may help reduce the risk that the requirements are incorrect.

Once the general requirements are gleaned from the client, an analysis of the scope of the development should be determined and clearly stated. This is often called a scope document.

Design

The software system design is produced from the results of the requirements phase. Architects have the ball in their court during this phase and this is the phase in which their focus lies. This is where the details on how the system will work is produced. Architecture, including hardware and software, communication, software design are all part of the deliverables of a design phase.

Implementation & Testing

Implementation is the part of the process where software engineers actually program the code for the project. Software testing is an integral and important part of the software development process. This part of the process ensures that bugs are recognized as early as possible.

Deployment and Maintenance

Deployment starts after the code is appropriately tested, is approved for release and sold or otherwise distributed into a production environment. Maintenance and enhancing software to cope with newly discovered problems or new requirements can

take far more time than the initial development of the software. It may be necessary to add code that does not fit the original design to correct an unforeseen problem or it may be that a customer is requesting more functionality and code can be added to accommodate their requests.

6. Conclusion:

To develop any software project, we have to go through all phases of system development life cycle model, starting from software requirement analysis phase to maintenance phase. There are various types of system development models and these models are used according to software project requirements.

7. Viva Questions:

- What is SDLC?
- Write SDLC for College Management System?
- Why SDLC is required?

8. References:

1. System Analysis & Design by Satzinger, Jackson and Burd, Cengage Learning, 2007.
2. System Analysis and Design Methods by Jeffery l. hitten, Lonnie D Bentley, McGraw Hill, 7 th edition.
3. System Analysis and Design by Alan Dennis, Barbara H. Wixom, Roberta M. Roth, Wiley India 4th edition.

Structured and Object Oriented Analysis and Design

Experiment No. : 2

Requirement specification document of the selected / allotted project

Experiment No. 2

1. **Aim:** To develop Requirement specification document of the selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to
   - Explore an object-oriented analysis and design (OOAD) methodology to the requirements.
   - Learn the efficient way to implement the system.
   - Understand the importance of this experiment from application point of view.

3. **Outcomes:** The learner will be able to
   - Understand and apply techniques to get the system requirements and present it in standard format.
   - Apply knowledge of computing, and fundamental engineering concepts appropriate to the discipline.
   - Recognition of the need for and an ability to engage in continuing professional development.

4. **Hardware / Software Required:** Rational Software Architect tool.

5. **Theory:**

   A Software requirements specification (SRS), a requirements specification for a software system, is a description of the behaviour of a system to be developed and may include a set of use cases that describe interactions the users will have with the software. In addition it also contains non-functional requirements. Non-functional requirements impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints). The software requirements specification document enlists enough and necessary requirements that are required for the project development. To derive the requirements we need to have clear and thorough understanding of the products to be developed or being developed.

1. Introduction

The introduction to the Software Requirement Specification (SRS) document should provide an overview of the complete SRS document. While writing this document please remember that this document should contain all of the information needed by a software engineer to adequately design and implement the software product described by the requirements listed in this document.

Typically software designers use IEEE STD 830-1998 as the basis for the entire software specifications the standard template for writing SRS is as given below.

Document Title
Author(s)
Affiliation
Address Date
Document version

Purpose

The purpose of this SRS and the (intended) audience for which it is written. The document gives detailed functional and non functional requirement for application provided. The purpose of document is that the requirements mentioned in it should be utilized by software developer to implement the system.

Scope

This subsection should:

(1) Identify the software product(s) to be produced by name; for example Host DBMS, Report Generator, etc
(2) Explain what the software product(s) will, and, if necessary, will not do.
(3) Describe the application of the software being specified. As a portion of this, it should:
(4) Describe all relevant benefits, objectives, and goals as precisely as possible. For example, to say that one goal is to provide effective reporting capabilities.

Overview

Provides the brief overview of the product defined as a result of the requirements elicitation process. The system provides an easy solution.

2. GENERAL DESCRIPTION

Describes the general functionality of the product such as similar system information, user characteristics, user objectives, general constraints placed on design team. Describe the features of the user community, including their expected expertise with software systems and the application domain.

2.1User Manual

The system should provide help option in which how to operate the system should be explained. Also hard copy of this document should be given to the user in a booklet form.

3. FUNCTIONAL REQUIREMENTS

This section lists the functional requirements in ranked order. A functional requirement describes the possible effects of a software system, in other words, what the system must accomplish .each functional requirement should be specified in following manner.

Short, imperative sentence stating ranked functional requirement.

3.1Description
A full description of the requirement

• Criticality
Describes how essential this requirement is to the overall system.

3.3Technical Issues
Describes any design or implementation issues involved in satisfying this requirement.

• Cost and schedule
Describes the relative or absolute costs of the system.

• Risks
Describes the circumstances under which this requirement might not able to be satisfied.

3.2Dependencies with other requirements
Describes interaction with other requirements.

4. INTERFACE REQUIREMENTS

This section describes how the software interfaces withj other software products or users for input or output. Examples of such interfaces include library routines , token streams , shared memory, data streams, and sop forth.

4.1GUI user Interface

Describes how this product interfaces with the user. It describe the graphical user interface if present this section should include a set of screen dumps to illustrate user interface features.

• CLI

Describes the command- line interface if present. For each command, a description of all arguments and example values and invocations should be provided.

• API

Describes the application programming interface, if present.

4.2 Hardware interface

Describes interfaces to hardware devices.

Hardware interface 1: The system should be embedded in the laptops.

Hardware interface 2: The laptop should use wireless Ethernet departmental database server card to send e-mails

4.3 software interface

Describes any remaining software interfaces not included above.

Software interface 1: maintenance system.

Software interface 2: e-mail message generator which generates standard.

Software interface 3: report generators

5. PERFORMANCE REQUIREMENTS

This system should work concurrently on multiple processors. Specifies speed and memory requirements.

6. DESIGN CONSTARINTS

Specifies any constraints for the design team such as software or hardware limitations.

7. OTHER NON FUNCTIONAL ATTRIBUTES.

Specifies any constraints for the design team such as software or hardware limitations.

- Security
- Reliability
- Availability
- Maintainability
- Reusability
- Portability

8. OPERATIONAL SCENARIOS

This section should describe a set of scenarios that illustrate, from the users perspective, what will be experienced when utilizing the system under various situations.

9. PRELIMINARY SCHEDULE

This section provides an initial version of project plan, including the major tasks to be accomplished, their interdependencies, and their tentative start / stop dates.

6. Conclusion:

Software Requirement Specification is written for the allotted Project.(write Project name). The document gives detailed functional and non functional requirement for application provided. It provides the brief overview of the product defined as a result of the requirements elicitation process.

7. Viva Questions:

- What is Requirement Specification?
- Why it is required?

8. References:

- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- "Modern Systems Analysis and Design by Jeffrey A. Hoffer, Joey F. George, Joseph S.Valacich, Prabin K. Panigrahi, Pearson Education Publication, 4th Edition".

Structured and Object Oriented Analysis and Design

Experiment No. : 3

ERD and DFD and model (Level 0, level 1 DFD and Data dictionary) of selected / allotted project

Experiment No. 3

1. **Aim:** To develop ERD and DFD and model (level 0, level 1 DFD and Data dictionary) of Selected / allotted project. (Write the name of your project)

2. **Objectives:** From this experiment, the student will be able to

   - Get balanced exposure to both traditional and object oriented approaches to system analysis & design.
   - To learn the efficient tools to implement the method.
   - To understand the importance of this experiment from application point of view.

3. **Outcomes:** The learner will be able to

   - To construct the candidate system following design methodology.
   - Have leadership and management skills to accomplish a common goal.
   - Communicate effectively in both graphical and written form.

4. **Hardware / Software Required:** Rational Software Architecture tool.

5. **Theory:**

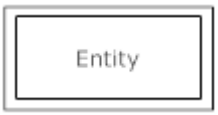
Entity Relationship Diagram

An entity relationship diagram (ERD) is one means of representing the objects and their relationships in the data model for a software product. Entity Relationship diagram notation:

Entity Relationship Diagram Notations

Entity: An entity is an object or concept about which you want to store information.

Weak Entity: A weak entity is an entity that must define by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.



Key attribute: A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.

Relationships: Relationships illustrate how two entities share information in the database structure.

Cardinality: Cardinality specifies how many instances of an entity relate to one instance of another entity.

Documenting Entity Relationship Diagram: (Note for students: Do not copy the content given below. You have to describe each entity of your system in the way given in the table. There will be such table for all the entities that are there in your system (if there are 5 entities then there will be 5 such tables describing each entity)

Entity 1

Entity Name	Unique Name of Entity and its type (like weak etc)
Attributes	List of attributes for each class
Types of attribute	Mention the key attribute, derived and multivalued attribute
Cardinality	One to many or many to one etc
Description (optional)	Description of Entity if needed

Data Flow Diagram

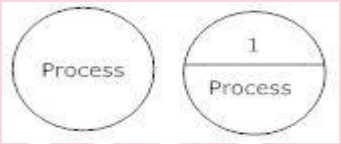
A data flow data diagram is one means of representing the functional model of a software product. DFDs do not represent program logic like flow transitions do. The DFD can be created for different levels. The context level DFD (also considered as level 0) shows the entire system as a single process, and gives no clues as to its internal organization. This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD

shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole.

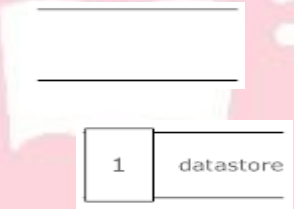
Data Flow Diagram Notations:

External Entity: External entities are objects outside the system, with which the system communicates. External entities are sources and destinations of the system's inputs and outputs.

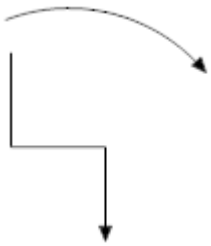
Process: A process transforms incoming data flow into outgoing data flow.



Datastore Notations
DataStore: Datastores are repositories of data in the system. They are sometimes also referred to as files.



Dataflow: Dataflow are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it



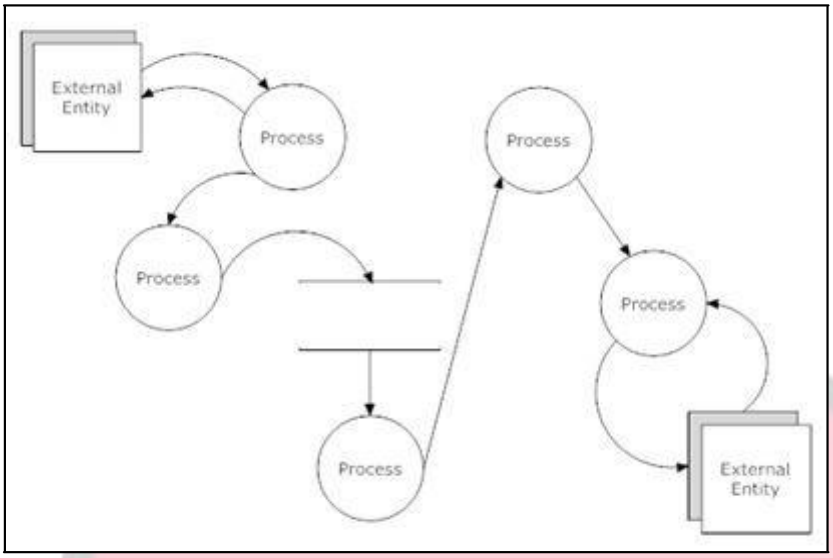


Fig: Data Flow Diagram

Documenting Data Flow Diagram:

Level 0 DFD Documentation: Process :

Process Name	Name of the process
Entities	List of entities related with this process
Data Flow	Name of the flow
Data store	List all the related datastores to this process
Description (optional)	Description of process if needed

LEVEL 1 DFD

Process Name	Name of the process
Entities	List of entities related with this process
Data Flow	Name of the flow
Data store	List all the related datastores to this process
Description (optional)	Description of process if needed

Process 2:

Process Name	Name of the process
Entities	List of entities related with this process
Data Flow	Name of the flow
Data store	List all the related datastores to this process
Description (optional)	Description of process if needed

|
|
Process n

6. Conclusion:

ERD is a data technique used in software engineering to produce a conceptual data model of an information system and DFD is a Graphical representation of functional of an information system ERD and DFD.

7. Viva Questions:

- What is the need of ER diagram and DFD?
- What is the difference between ER diagram and DFD?
- What do you mean by functional ?

8. References:

- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- "Modern Systems Analysis and Design by Jeffrey A. Hoffer, Joey F. George, Joseph S.Valacich, Prabin K. Panigrahi, Pearson Education Publication, 4th Edition."

Structured and Object Oriented Analysis and Design

Experiment No. : 4

Class diagram for selected / allotted project

Experiment No. 4

1. **Aim:** To study and draw Class diagram for selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Become conversant with the Unified Language (UML) notation and symbols.
   - Learn the efficient tools to implement the method.
   - Understand the importance of this experiment from application point of view.

3. **Outcomes:** The learner will be able to

   - Understand architecture strategies for information System.
   - Match the industry requirements in the domains of Database management, Programming and Networking with the required management skills.
   - Analyze the local and global impact of computing on organizations, and society.

4. **Hardware / Software Required:** Rational Software Architect tool.

5. **Theory:**

   Object diagram provide a formal graphic notation for object, classes and their relationship to one another. Object diagrams are useful for both abstract modelling and for designing actual programs. There are two types of object diagram: class diagrams and instance diagrams.
   Class diagram is a schema, pattern or template for describing many possible instances of data. A class diagram describes object classes. An instance diagram describes how a particular set of objects relates to each other. An instance diagram describes object instances. Instance diagram are useful for documenting test cases (especially scenarios) and discussing example. A given class diagram describes infinite set of instance diagrams

   Need of Class Diagram:
   An object model captures a static structure of a system by showing the objects in the system, relationship between the objects, and the attributes and operation that characterize each class of objects. Object models provide an intuitive graphic representation of a system and are valuable for communicating with customers and documenting the structure of the system.

Steps for Constructing Object Model (Class Diagram):

The first step in analyzing the requirements is to construct an object model. The object model shows the static data structure of the real world system and organizes it into workable pieces. The object model describes real world object classes and their relationships to each other.

The following steps are performed in constructing an object model:

(1) Identify objects and classes
(2) Prepare a data dictionary
(3) Identify associations (including aggregations) between objects.
(4) Identify attributes of objects and links.
(5) Organize and simplify object classes using inheritance.
(6) Identify operations to be included in a class.
(7) Verify that access paths exist for likely queries.
(8) Iterate and refine the model
(9) Group classes into modules

Elements of Class Diagram:

Class: Classes are composed of three things: a name, attributes, and operations. Below is an example of a class:

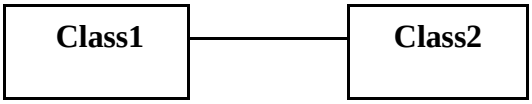
Class-Name
Attribute Name - 1 : datatype 1 = default value 1 Attribute Name - 2 : datatype 2 = default value 2
Operation Name 1 (argument List 1) : result Type 1 Operation Name 2 (argument List 2) : result Type 2

Attributes: An attribute is data value held by the objects in a class. An attribute should be a pure data value, not an object. Unlike objects, pure data values do not have identity.

Operations and Methods: An operation is a function or transformation that may be applied to or by objects in a class. Operations are listed in the lower third of the class box.

Links and Association: Links and association are the means for establishing relationships among objects and classes. A link is a physical or conceptual connection between object instances. An association describes a group of links with common structure and common semantics.

An association describes a set of potential links in the same way that a class describes a set of potential objects. Associations are inherently bi-directional.



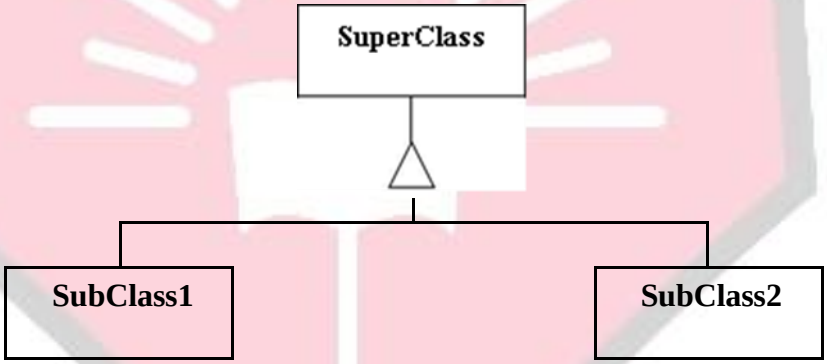
Association

Multiplicity: Multiplicity specifies how many instances of one class may relate to a single instance of an associated class. Multiplicity constrains the number of related objects. Multiplicity is often described as being " one" or "many" but more generally it is subset of non negative integers.

Object diagram indicate multiplicity with special symbols at the ends of association lines. Multiplicity can be specified with a number or set of intervals, such as "1", "1+"(1 or more), "3-5"(3 to 5, inclusive), and "2, 4,18" (2,4 or 18)

Generalization and Inheritance: Generalization and Inheritance are powerful abstractions for sharing similarities among classes while preserving their differences.

Generalization is the relationship between a class and one or more refined versions of it. The class being refined is called the superclass and each refined version is called subclass. Attributes and operations common to a group of subclasses are attached to the superclass and shared by each subclass. Each subclass is said to inherit the features of its superclass. Generalization is sometimes called the "is-a" relationship because each instance of a subclass is an instance of the superclass as well.



Generalization

Aggregation: Aggregation is the "part- whole" or "a-part-of" relationship is which objects representing the components of something are associated with an object representing the entire assembly. Aggregation is a tightly coupled form of association with some extra semantics. The most significant property of aggregation is transitivity that is if A is part of B, and B is part of C then A is part of C. Aggregation is also antisymmetric, that is if A is part of B then B is not part of A. Finally, some properties of the assembly propagate to the components as well possibly with some local modifications.

Documenting Class Diagram:

CLASS 1

Class Id	Unique Id of Class
Class Name	Name of Class
Attributes	List of attributes for each class
Methods	Functions carried out by class
Associations	Relationship between different classes
Inheritance	Classes sharing similarities
Multiplicity	Identify how many instances of one class may relate to single instance of an associated class
Description	Description of Diagram

CLASS 2
….
CLASS N

6. Conclusion:

Class diagram is useful for abstract modelling and for designing actual programs. Class Diagram matches the industry requirements in the domains of Database management, Programming and Networking which gives the complete knowledge and understanding of the system .It also helps in analyzing the local and global impact of computing on system and organizations.

7. Viva Questions:

- Write different notations for class diagram?
- Draw class diagram for hospital management system?
- What is the difference between Aggregation and Generalization?

8. References:

- The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication.
- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- "Modern Systems Analysis and Design by Jeffrey A. Hoffer, Joey F. George, Joseph S.Valacich, Prabin K. Panigrahi, Pearson Education Publication, 4th Edition."
- https://www.youtube.com/watch?v=TL4ABTx_RtE
- home.iitk.ac.in/~blohani/Limulator/publication/Rakesh_Indore_SE.pdf

Structured and Object Oriented Analysis and Design

Experiment No. : 5

Use case diagram for selected / allotted project

Experiment No. 5

1. **Aim:** To study and draw Use case diagram for selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Apply key modelling concepts to the Object Oriented approach.
   - Learn the document the system.
   - Learn the efficient tools to design use case diagram.

3. **Outcomes:** The learner will be able to

   - Apply various tools and techniques for key modelling to both the traditional structured approach and the Object Oriented approach.
   - Recognize the need of use case diagram to analyse the system.
   - Use current techniques, skills, and tools necessary to design use case diagram.

4. **Hardware / Software Required:** Rational Software Architect tool.

5. **Theory:**

The Object Oriented analysis phase of the Unified approach uses Use Case diagram to describe the system from user's perspective.

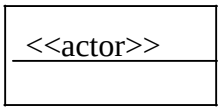
Need of Use Case:

Use Case describes the behaviour of a system from a user's standpoint, Provides functional description of a system and its major processes, Provides graphic description of the users of a system and what kinds of inheritance to expect within that system, Displays the details of the processes that occur within the application area, Used to design the test cases for testing the functionality of the system.

Elements of Use Case:

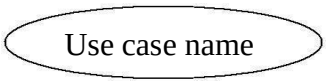
A Use Case diagram is quite simple in nature and depicts two types of elements: one representing the business roles and the other representing the business processes. Elements of Use Case diagrams are:

Actors: An actor portrays any entity (or entities) that perform that perform certain roles in a given system. An actor in a use case diagram interacts with a use case and is depicted "outside" the system boundary.



Actor Name

Use case: A use case in a Use Case diagram is a visual representation of distinct business functionality in a system. Each use case is a sequence of transactions performed by the system that produces a major suit for the actor.



System Boundary: A system boundary defines the scope of what a system will be. A system cannot have infinite functionality.

Relationships in Use Case:

1. Include: Include is used when two or more use cases share common portion in the flow of events. The stereotype <<include>> identifies the relationship include.

2. Extend: In an extend relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case.

3. Generalization: A generalization relationship is also a parent-child relationship between use cases. The child use case in the generalization relationship has the underlying business process meaning, but is an enhancement of the parent use case.

DOCUMENTING USE CASE DIAGRAM:

USE CASE 1:

Use case name	Name of use case
Use case Id	Unique identifier of the use case
Super use case	If the use case inherits a parent use case
Actor	The actor which are participating in the execution of use case.
Brief Description	Description of scope of use case and
Preconditions	Constraints that must be satisfied before
Post conditions	Condition will be established after the use case.
Priority	Development priority from the view of development team
Flow of events	Ste by step description of the interaction
Alternative flows and exceptions	Alternatives or exceptions that may occur.

Non-behavioral requirements	Non-functional requirements like h/w and s/w requirements.
Assumptions	All the assumptions made about the use
Issues	All outstanding issues related to the use case need to be resolved.
Source	Includes references and materials used in developing use case

6. Conclusion:

Use Case diagram helps to understand the different processes and entities involved in a system and help the analyst to understand and document the system.

7. Viva Questions:

- Are use cases the same as functional requirements or functional requirements are different from use cases?
- What is use case diagram?
- What is actor in use case diagrams?
- What is the difference between use case diagram and use case?

8. References:

- The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication.
- https://www.youtube.com/watch?v=tLJXJLfLCCM&list=PL05DE2D2EDDEA8D68
- www.sqa.org.uk/e-learning/SDM01CD/page_05.htm

Structured and Object Oriented Analysis and Design

Experiment No. : 6

Activity diagram for selected / allotted project

Experiment No. 6

1. **Aim:** To study and draw Activity diagram for selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Apply key modelling concepts to both the traditional structured approach and the Object Oriented approach.
   - Captures the dynamic behaviour of the system.
   - Learn the efficient tools to design Activity diagram.

3. **Outcomes:** The learner will be able to

   - Apply various tools and techniques for key modelling to both the traditional structured approach and the Object Oriented approach.
   - Model the activities which are nothing but business requirements.
   - Use current techniques, skills, and tools necessary to design activity diagram.

4. **Hardware / Software Required:** Rational software architect tool.

5. **Theory:**

Activity diagram is used for business process modelling, for modelling the logic captured by a single use case or usage scenario, or for modelling the detailed logic of a business rule. Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state. The easiest way to visualize an activity diagram is to think of a flow transition and data flow diagrams (DFDs).

Need of an Activity Diagram:

The general purpose of activity diagrams is to focus on flows driven by internal processing vs. external events. Activity diagrams are also useful for: analysing a use case by describing what actions needs to take place and when they should occur; describing a complicated sequential algorithm; and modelling applications with parallel processes.

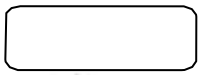
Elements of an Activity Diagram

An Activity diagram consists of the following behavioural elements:

Initial Activity: This shows the starting point or first activity of the flow and denoted by a solid circle. There can only be one initial state on a diagram.

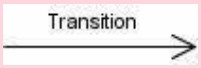
Initial Activity

Activity: Represented by a rectangle with rounded (almost oval) edges.



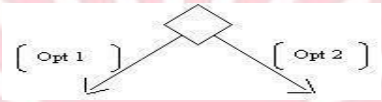
Activity

Transition: When an activity state is completed, processing moves to another activity state. Transitions are used to mark this movement. Transitions are modelled using arrows.



Transition

Decisions: Similar to flowcharts, a logic where a decision is to be made is depicted by a diamond, with the options written on either side of the arrows emerging from the diamond, within box brackets.



Decisions

Synchronization Bar: Activities often can be done in parallel. To split processing ("fork"), or to resume processing when multiple activities have been completed ("join"), Synchronization Bars are used. These are modelled as solid rectangles, with multiple transitions going in and/or out. Fork denotes the beginning of parallel activity. Join denotes the end of parallel processing.

Synchronization Bar

Final Activity: The end of the Activity diagram is shown by a bull's eye symbol, also called as a final activity. An activity diagram can have zero or more activity final nodes.

Final Activity

Swim Lanes

Activity diagrams provide another ability, to clarify which actor performs which activity. If you wish to distinguish in an activity diagram the activities carried out by individual actors, vertical columns are first made, separated by thick vertical black lines, termed swim lanes and name each of these columns with the name of the actor involved. You place each of the activities below the actor performing these activities and then show how these activities are connected.

Documenting Activity Diagram:

Activity 1:

Activity name	Name of the activity
Description	Description about activity
Events	Events occurred during activity
Actor	Actor performing activity
Join synchronization bar	To show multiple activities occurring simultaneously
Fork synchronization bar	To resume processing when multiple activities have been completed
Activity Diagram Id	ID that identifies diagram uniquely
Name	Name o the diagram
Preconditions	Conditions before starting the activities
Post Conditions	Conditions after the activities are over

6. Conclusion:

It can be concluded saying activity diagrams focus on flows driven by internal processing vs. external events and used to model a logic captured by single scenario.

7. Viva Questions:

- What is an Activity Diagram and what is its purpose?
- What is swim lanes?

8. References:

- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- Object-Oriented and Design with UML by Michael Blaha, James Rumbaugh, Pearson Education Publication, 2nd Edition.
- http://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_paradigm.htm

Structured and Object Oriented Analysis and Design

Experiment No. : 7

State Transition diagram for selected / allotted project

Experiment No. 7

1. **Aim:** To study and draw State Transition diagram for selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Apply key modelling concepts to both the traditional structured approach and the Object Oriented approach.
   - Model life time of a reactive system.
   - Learn the efficient tools to design state transition diagram.

3. **Outcomes:** The learner will be able to

   - Define states it is used to model lifetime of an object.
   - Describe the behaviour of the system is analysed and represented as a series of events that can occur in one or more possible states.
   - Use current techniques, skills, and tools necessary to design state transition diagram.

4. **Hardware / Software Required:** Rational software architect tool.

5. **Theory:**

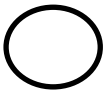
State transition diagrams model the dynamic behaviour of individual classes or any other kind of object. They show the sequences of states that an object goes through, the events that cause a transition from one state to another and the actions that result from a state change. State transition diagrams are closely related to activity diagrams. The main difference between the two diagrams is state transition diagrams are state centric, while activity diagrams are activity centric.

Need of State Transition Diagram:

A state transition diagram is typically used to model the discrete stages of an object's lifetime, whereas an activity diagram is better suited to model the sequence of activities in a process. Each state represents a named condition during the life of an object during which it satisfies some condition or waits for some event. A state transition diagram typically contains one start state and multiple end states. Transitions connect the various states on the diagram. As with activity diagrams, decisions, synchronizations, and activities may also appear on state transition diagrams.

A State chart diagram consists of the following behavioural elements:

Start State: The start state icon is a small, filled circle that may contain a name (Begin Process).



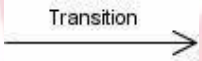
Initial State

State: The state icon appears as a rectangle with rounded corners and a name. The name of a state should be unique to its enclosing class, or if nested, within the state. All state icons with the same name in a given diagram represent the same state.

`

State

Transition: When an activity state is completed, processing moves to another activity state. Transitions are used to mark this movement. Transitions are modelled using arrows.



Transition

Actions: Actions on states can occur at one of four times:

- On entry
- On exit
- do
- On event.

An on event action is similar to a state transition label with the following syntax: event (args)[condition] : the Action

End State: The end state icon is a filled circle inside a slightly larger unfilled circle that may contain a name (End Process):

Documenting State Chart Diagram:

STATE 1:

State Name	Task performed by that object
State Id	Unique id of that state
Activity	Activity performed.

Transitions	Identify transitions to move from one activity to another activity

STATE 2:

.

.

.

STATE n:

6. Conclusion:

It can be concluded saying state chart diagrams model the discrete stages of an object's lifetime.

7. Viva Questions:

- What is difference between state chart and sequence diagram?
- What is history state, concurrent state, start state?

8. References:

- Analysis and Design of Information Systems by James a. Senn, 2nd Edition,
- McGrawHill.
- "The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication."
- www.tutorialspoint.com/object.../ooad_uml_behavioural_diagrams.htm
- https://youtu.be/qiyMyyYqZVY

Structured and Object Oriented Analysis and Design

Experiment No. : 8

Collaboration diagram for selected / allotted project

Experiment No. 8

1. **Aim:** To study and draw Collaboration diagram / Communication Diagram for selected / allotted project.

2. **Objectives:** The learner will be able to

   - Apply UML notation to construct and graphically present communication diagram for Object- Oriented systems analysis.
   - Illustrate the relationships and interactions among software objects in the Unified Modelling Language (UML).
   - Learn the efficient tools to design communication diagram.

3. **Outcomes:** The learner will be able to

   - Identify the roles, functionality and behaviour of individual objects as well as the overall operation of the system in real time.
   - Show message flow between objects in an object oriented application.

4. **Hardware / Software Required:** Rational software architect tool.

5. **Theory:**

   A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages.
   We form a collaboration diagram by first placing the objects that participate in the interaction as the vertices in a graph, and then add the links that connect these objects as the arcs of this graph. Finally, adorn these links with the messages that objects sends and receive with sequence numbers.

   Need of Collaboration Diagram:

   We use collaboration diagram to describe a specific scenario. Numbered arrows show the movement of messages during the course of a scenario. A distinguishing feature of a Collaboration diagram is that it shows the objects and their association with other objects in the system apart from how they interact with each other. The association between objects is not represented in a Sequence diagram.

   Elements of Collaboration Diagram:

A sophisticated tool can easily convert a collaboration diagram into a sequence diagram and the vice versa. Hence, the elements of a Collaboration diagram are essentially the same as that of a Sequence diagram.

Object : The objects interacting with each other in the system. Depicted by a rectangle with the name of the object in it, preceded by a colon and underlined.

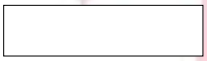
Object

Relation/Association: A link connecting the associated objects. Qualifiers can be placed on either end of the association to depict cardinality.

* 1..*

Association

Messages: An arrow pointing from the commencing object to the destination object shows the interaction between the objects. The number represents the order/sequence of this interaction.



Message

Documenting Collaboration Diagram:

Scenario Id Scenario Name
Objects Participating in Collaboration
Association between Objects
Sequence of Operations in Scenario

6. Conclusion:

Thus Collaboration diagram helps to model different objects in a system and to represent the associations between the objects as links.

7. Viva Questions:

- What is difference between collaboration and sequence diagram?
- What are the diagram notations for collaboration and sequence diagram?

8. References:

- The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication.
- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- https://www.youtube.com/watch?v=TL4ABTx_RtE
- home.iitk.ac.in/~blohani/Limulator/publication/Rakesh_Indore_SE.pdf

Structured and Object Oriented Analysis and Design

Experiment No. : 9

Sequence diagram for selected / allotted project

Experiment No. 9

1. **Aim:** To study and draw Sequence diagram for selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Apply UML notation to construct and graphically present communication diagram for Object- Oriented systems analysis.
   - Show objects interactions arranged in time sequence and messages exchanged.
   - Learn the efficient tools to design sequence diagram.

3. **Outcomes:** The learner will be able to

   - Identify the roles, functionality and behaviour of individual objects as well as the overall operation of the system in real time.
   - Show object interactions arranged in time sequence.

4. **Hardware / Software Required:** Rational software architect tool.

5. **Theory:**

   A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence. Sequence diagrams are closely related to collaboration diagrams and both are alternate representations of an interaction. There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other.
   A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

   Need of Sequence Diagram:
   Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces. This type of diagram is best used during early analysis phases in design because they are simple and easy to comprehend. Sequence diagrams are normally associated with use cases.

   Elements of Sequence Diagram:
   The following tools located on the sequence diagram toolbox enable to model sequence diagrams:

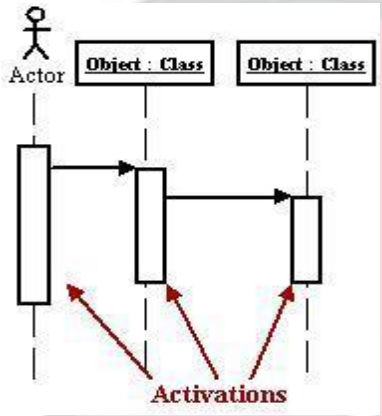
   Class roles

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Object : Class

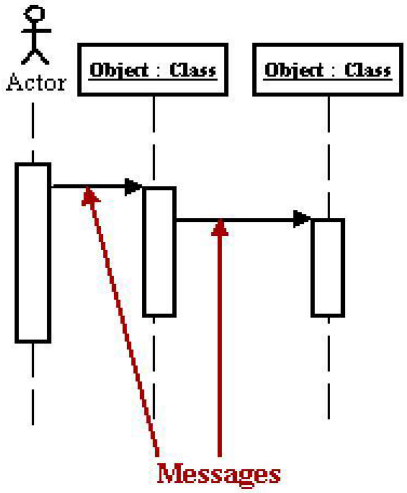
Activation

Activation boxes represent the time an object needs to complete task.



Messages

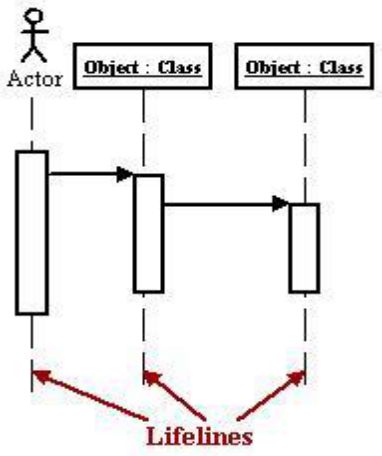
Messages are arrows that represent communication between objects. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.



Arrow	Message type
→	Simple
→	Synchronous
→	Asynchronous
↩	Balking
⏱→	Time out

Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



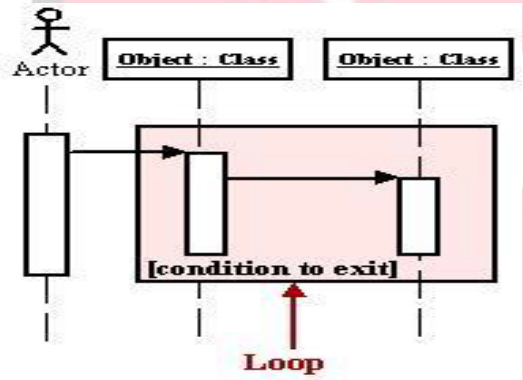
Destroying Objects:

Objects can be terminated early using an arrow labeled "< < destroy > >" that points to an X.

< < destroy > >

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].



Documenting Sequence Diagram:

Scenario Id
Scenario Name
Objects Participating in Sequence
Sequence of Operations in Scenario

6. Conclusion:

Sequence diagram helps to model different objects in a system and to represent the sequence of operations in scenario.

7. Viva Questions:

- How to model exception handling in sequence diagram?
- Do we show private or protected methods (messages) on sequence diagrams?
- Can a sequence diagram be replaced by communication diagram?

8. References:

- The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication.
- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- https://www.youtube.com/watch?v=4WDbte6cPa8
- home.iitk.ac.in/~blohani/Limulator/publication/Rakesh_Indore_SE.pdf

Structured and Object Oriented Analysis and Design

Experiment No. : 10

Component diagram for selected / allotted project

Experiment No. 10

1. **Aim:** To study and draw Component diagram for selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Understand architecture strategies for information system.
   - Show objects interactions arranged in time sequence and messages exchanged.
   - Learn the efficient tools to design sequence diagram.

3. **Outcomes:** The learner will be able to

   - Identify the roles, functionality and behaviour of individual objects as well as the overall operation of the system in real time.
   - Show object interactions arranged in time sequence.

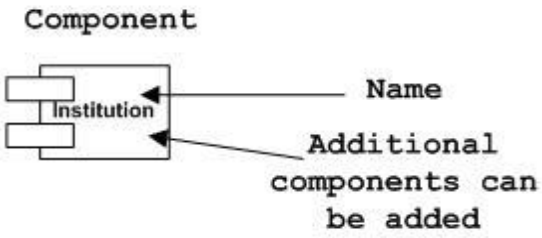
4. **Hardware / Software Required:** Rational software architect tool.

5. **Theory:**

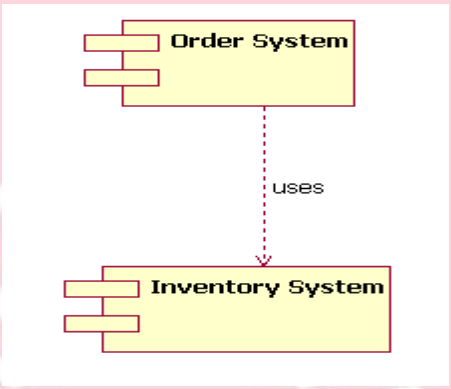
   Component diagrams are used to model physical aspects of a system. Physical aspects are the elements like executable, libraries, files, documents etc which resides in a node. So component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems. Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities. Component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files etc. Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment. A single component diagram cannot represent the entire system but a collection of diagrams are used to represent the whole.
   So the purpose of the component diagram can be summarized as:

   - Visualize the components of a system.
   - Construct executables by using forward and reverse engineering.
   - Describe the organization and relationships of the components.

   The notation A component in UML is shown as below with a name inside. Additional elements can be added wherever required.



Component is used to represent any part of a system for which UML diagrams are made. The component diagram notation set now makes it one of the easiest UML diagrams to draw. Figure below shows a simple component diagram using the former UML notation; the example shows a relationship between two components: an Order System component that uses the Inventory System component. As you can see, a component in UML was drawn as a rectangle with two smaller rectangles protruding from its left side.



This simple component diagram shows the Order System's general dependency

Documenting Component Diagram:

Scenario Id
Scenario Name
Objects Participating in component diagrams

6. Conclusion:

A component diagram describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

7. Viva Questions:

- What is component diagram?
- What's the Difference Between a Package Diagram and a Component Diagram?

8. References:

- The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication.
- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- "Modern Systems Analysis and Design by Jeffrey A. Hoffer, Joey F. George, Joseph S.Valacich, Prabin K. Panigrahi, Pearson Education Publication, 4th Edition."
- https://www.smartdraw.com/component-diagram

Structured and Object Oriented Analysis and Design

Experiment No. : 11

Deployment diagram for selected / allotted project

Experiment No. 11

1. **Aim:** To study and draw Deployment diagram for selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Familiarize with the benefits and liabilities of a small subset of architectural and design patterns.
   - To learn the efficient tool to draw the deployment diagram.
   - To understand the importance of this experiment from application point of view.

3. **Outcomes:** The learner will be able to

   - Understand architecture strategies for information System.
   - Match the industry requirements in the domains of Database management and Networking with the required management skills.
   - Analyze the local and global impact of computing on organizations, and society.

4. **Hardware / Software Required:** Rational software architect tool.

5. **Theory:**

   Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed. So deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

   The name Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components where software components are deployed. Component diagrams and deployment diagrams are closely related. Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

   UML is mainly designed to focus on software artifacts of a system. But these two diagrams are special diagrams used to focus on software components and hardware components.

   So most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on hardware topology of a system. Deployment diagrams are used by the system engineers.

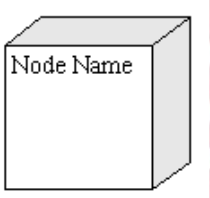
The purpose of deployment diagrams can be described as:

- Visualize hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe runtime processing nodes.

Deployment diagrams are used to model the configuration of run-time processing elements and the software components, processes, and objects that live on them. In the deployment diagram, you model the physical nodes and the communication associations that exist between them. Each node can contain run-time component instances, indicating that the component lives or runs on the node. You may optionally model the objects that are contained within the component. Deployment diagrams are used to model only components that exist as run-time entities; they are not used to model compile-time only or link-time only components.

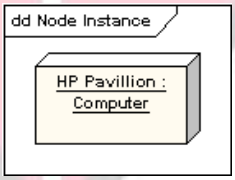
Notations Component

A node is a physical resource that executes code components.



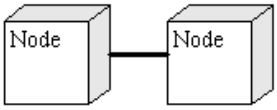
Node Instance

A node instance can be shown on a diagram. An instance can be distinguished from a node by the fact that its name is underlined and has a colon before its base node type. An instance may or may not have a name before the colon. The following diagram shows a named instance of a computer.



Association

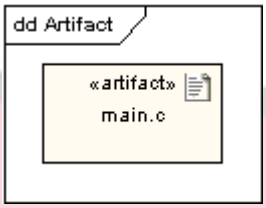
Association refers to a physical connection between nodes, such as Ethernet.



Artifact

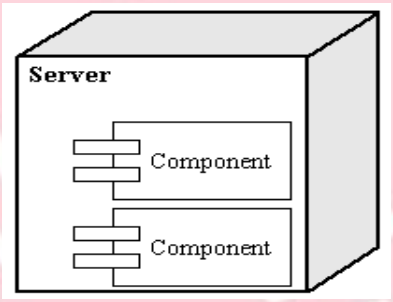
An artifact is a product of the software development process. That may include process models (e.g. use case models, design models etc), source files, executables, design documents, test reports, prototypes, user manuals, etc.

An artifact is denoted by a rectangle showing the artifact name, the «artifact» keyword and a document icon, as shown below.



Components and Nodes

Place components inside the node that deploys them.



Documenting Deployment Diagram:

Scenario Id

Scenario Name

Objects Participating in Deployment diagrams

6. Conclusion:

We have successfully drawn the Deployment diagram for our system (System Name). Deployment diagram are used to represent the systems data base in effective way. We have understood the different components of Deployment Diagram and their uses.

7. Viva Questions:

- Why we use Deployment Diagram?
- What are the objects participated in Deployment Diagram?
- Draw the deployment diagram for given system?

8. References:

- The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication.
- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- "Modern Systems Analysis and Design by Jeffrey A. Hoffer, Joey F. George, Joseph S.Valacich, Prabin K. Panigrahi, Pearson Education Publication, 4th Edition."

Structured and Object Oriented Analysis and Design

Experiment No. : 12

Prototype of project selected / allotted project

Experiment No. 12

1. **Aim:** To develop the prototype of project selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Get balanced exposure to both traditional and object oriented approaches to system analysis & design.
   - to analyze the need for the experiment.
   - to understand the importance of this experiment from application point of view.

3. **Outcomes:** The learner will be able to

   - To understand, identify, analyze and design the problem.
   - Validate the solution using software.
   - Match the industry requirements in the domains of database management, programming and networking with the required management skills.

4. **Hardware / Software Required:** Rational software architect tool.

5. **Theory:**

Prototypes are typically a feature of an online application, since they allow a client to visualize what the final solution could look like. The prototype is a shell that contains the online screens. There would be very little programming of the core business processes. The user would have a screen that they can use to input information. The logic would then go to the next screen(s), passing whatever information made sense. User input could just be ignored, and screen values might be hard-coded, rather than require a database call to a table that may or may not exist yet.

The process of prototyping involves the following steps

1. Identify basic requirements

Determine basic requirements including the input and output information desired. Details, such as security, can typically be ignored.

2. Develop Initial Prototype

The initial prototype is developed that includes only user interfaces.

3. Review

The customers, including end-users, examine the prototype and provide feedback on additions or changes.

4. Revise and Enhance the Prototype

Using the feedback both the specifications and the prototype can be improved. Negotiation about what is within the scope of the contract/product may be necessary. If changes are introduced then a repeat of steps #3 and #4 may be needed.

Horizontal Prototype

A common term for a user interface prototype is the horizontal prototype. It provides a broad view of an entire system or subsystem, focusing on user interaction more than low-level system functionality, such as database access. Horizontal prototypes are useful for:

- Confirmation of user interface requirements and system scope
- Demonstration version of the system to obtain buy-in from the business
- Develop preliminary estimates of development time, cost and effort.

Vertical Prototype

A vertical prototype is a more complete elaboration of a single subsystem or function. It is useful for obtaining detailed requirements for a given function, with the following benefits:

- Refinement database design
- Obtain information on data volumes and system interface needs, for network sizing and performance engineering
- Clarifies complex requirements by drilling down to actual system functionality

Advantages of prototyping

There are many advantages to using prototyping in software development like:

Reduced time and costs: Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of what the user really wants can result in faster and less expensive software.

Improved and increased user involvement: Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications. The presence of the prototype being examined by the user prevents many misunderstandings and miscommunications that occur when each side believe the other understands what they said. Since users know the problem domain better than anyone on the development team does, increased interaction can result in final product that has greater tangible and intangible quality. The final product is more likely to satisfy the user's desire for look, feel and performance.

Disadvantages of prototyping

Using, or perhaps misusing, prototyping can also have disadvantages:

Insufficient analysis: The focus on a limited prototype can distract developers from properly analyzing the complete project. This can lead to overlooking better

solutions, preparation of incomplete specifications or the conversion of limited prototypes into poorly engineered final projects that are hard to maintain. Further, since a prototype is limited in functionality it may not scale well if the prototype is used as the basis of a final deliverable, which may not be noticed if developers are too focused on building a prototype as a model.

User confusion of prototype and finished system: Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished. (They are, for example, often unaware of the effort needed to add error-checking and security features which a prototype may not have.) This can lead them to expect the prototype to accurately model the performance of the final system when this is not the intent of the developers. Users can also become attached to features that were included in a prototype for consideration and then removed from the specification for a final system. If users are able to require all proposed features be included in the final system this can lead to conflict.

Developer misunderstanding of user objectives: Developers may assume that users share their objectives (e.g. to deliver core functionality on time and within budget), without understanding wider commercial issues. For example, user representatives attending Enterprise software (e.g. PeopleSoft) events may have seen demonstrations of "transaction auditing" (where changes are logged and displayed in a difference grid view) without being told that this feature demands additional coding and often requires more hardware to handle extra database accesses. Users might believe they can demand auditing on every field, whereas developers might think this is feature creep because they have made assumptions about the extent of user requirements. If the developer has committed delivery before the user requirements were reviewed, developers are between a rock and a hard place, particularly if user management derives some advantage from their failure to implement requirements.

Developer attachment to prototype: Developers can also become attached to prototypes they have spent a great deal of effort producing; this can lead to problems like attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture.

Excessive development time of the prototype: A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex. When the prototype is thrown away the precisely developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. Users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product.

Expense of implementing prototyping: the start up costs for building a development team focused on prototyping may be high. Many companies have development

methodologies in place, and changing them can mean retraining, retooling, or both. Many companies tend to just jump into the prototyping without bothering to retrain their workers as much as they should.
A common problem with adopting prototyping technology is high expectations for productivity with insufficient effort behind the learning curve. In addition to training for the use of a prototyping technique, there is an often overlooked need for developing corporate and project specific underlying structure to support the technology. When this underlying structure is omitted, lower productivity can often result

Types of prototyping

Throwaway prototyping
Also called as close-ended prototyping. Throwaway or Rapid Prototyping refers to the creation of a model that will eventually be discarded rather than becoming part of the final delivered software. After preliminary requirements gathering is accomplished, a simple working model of the system is constructed to visually show the users what their requirements may look like when they are implemented into a finished system.
Rapid Prototyping involved creating a working model of various parts of the system at a very early stage, after a relatively short investigation. The method used in building it is usually quite informal, the most important factor being the speed with which the model is provided. The model then becomes the starting point from which users can re-examine their expectations and clarify their requirements. When this has been achieved, the prototype model is 'thrown away', and the system is formally developed based on the identified requirements.
In this approach the prototype is constructed with the idea that it will be discarded and the final system will be built from scratch. The steps in this approach are:

1. Write preliminary requirements
2. Design the prototype
3. User experiences/uses the prototype, specifies new requirements

4. Repeat if necessary
5. Write the final requirements

Evolutionary prototyping
Evolutionary Prototyping (also known as breadboard prototyping) is quite different from Throwaway Prototyping. The main goal when using Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built.
When developing a system using Evolutionary Prototyping, the system is continually refined and rebuilt.

For a system to be useful, it must evolve through use in its intended operational environment. A product is never "done;" it is always maturing as the usage environment changes…we often try to define a system using our most familiar frame of reference---where we are now. We make assumptions about the way business will be conducted and the technology base on which the business will be implemented. A plan is enacted to develop the capability, and, sooner or later, something resembling the envisioned system is delivered. Evolutionary Prototypes have an advantage over Throwaway Prototypes in that they are functional systems. Although they may not have all the features the users have planned, they may be used on an interim basis until the final system is delivered.

In Evolutionary Prototyping, developers can focus themselves to develop parts of the system that they understand instead of working on developing a whole system.

To minimize risk, the developer does not implement poorly understood features. The partial system is sent to customer sites. As users work with the system, they detect opportunities for new features and give requests for these features to developers. Developers then take these enhancement requests along with their own and use sound configuration-management practices to change the software-requirements specification, update the design, recode and retest.

Incremental prototyping

The final product is built as separate prototypes. At the end the separate prototypes are merged in an overall design. By the help of incremental prototyping we can reduce the time gap between user and software developer.

Extreme prototyping

Extreme Prototyping as a development process is used especially for developing web applications. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase, the services are implemented. The process is called Extreme Prototyping to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the services other than their contract.

6. Conclusion:

We have developed prototype of our project (selected / allotted project.)(Write the name of your project) using the RSA tool. With the help of Prototype we have analyze our system (Write the name of your project) and validate the software solution. Also we have tried to match the industry requirements in the domains of Database management, Programming and Networking with the required management skills.

7. Viva Questions:

- What is Incrementing prototyping?
- What is Extreme prototyping?
- What are the different types of prototyping?

8. References:

- The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication.
- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- "Modern Systems Analysis and Design by Jeffrey A. Hoffer, Joey F. George, Joseph S.Valacich, Prabin K. Panigrahi, Pearson Education Publication, 4th Edition."