Date.
Page No.

Software Requirements Engineering

A process of defining, documenting and maintaining the requirements is known as requirement engineering

Software requirements refer to the system's features, conditions or capability that is yet to be developed.

There are three imp people in collecting requirements: the customer, developer and end-users.

Requirements are always specified for the proposed system.

Requirements are functional and non-functional types.

Functional Requirements

The features of the software or intended functions are covered in functional requirements

If you enter login and password in the mail login page, it should take you to the mail box if the password is correct

Date.
Page No.

If a password is incorrect, it must give the related message and suggest the different options to log in

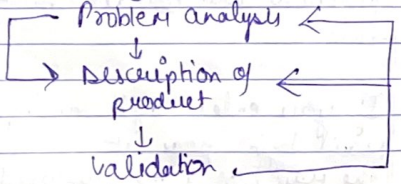
Non-functional Requirements
Security, reliability, performance, maintainability, scalability, and usability are the imp non-functional requirements.

When you click on a website, how fast it gets loaded or how good its design is, or the different types of downloads it supports

In some critical application Non-functional requirements may be as imp as functional requirements.

Requirement Analysis Process

Gain customer's Needs



Date.
Page No.

Problem analysis
A high level problem statement is defined in the problem analysis phase.

It is the initial phase in the requirement process.

In this activity, what should be provided by the software is understood.

The efforts, conditions of the system, input, output, etc are modeled in this phase.

The system analyst plays a major role in meeting the clients and end users.
System analyst keeps all the information with systematic documentation
He will conduct several meetings and brainstorm with his team to produce ideas.

In these early meetings the analyst listens to absorb the information about the system.

In the final stage, the system analyst produces a description of what the system should do and what are the objectives

Date.
Page No.

Software Requirements Specification

The outcome of the problem analysis is used as a base of requirement specification.

A document is prepared to clearly specify the requirements

The tools, representation and specification languages are used in this activity

Organization and description of the requirement document with a few redundancies is the main goal of this activity.

The analyst collects all the required information regarding the software to be developed

He removes all incompleteness and inconsistencies from the specification

The requirements are systematically organized in the form of a SRS document

Date.
Page No.

SRS Document

SRS is a document that describes what the software will do and what is its expected performance along with the functions of the product as per the users or business.

The SRS is the official statement of what is required of the software system It includes a definition of user requirements and a specification of the system requirements

Characteristics

Correct: SRS is correct if every requirement included in the SRS represents something required in the final system

Complete: If the details of what the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS then it is complete

Unambiguous: SRS is said to be unambiguous if all the requirements stated have only one interpretation

Date.
Page No.

Consistent: No more than one requirement should conflict with another. SRS should have the exact meaning for all requirements

Concise: A good SRS is a short and concise document

Modifiable / Maintainable: SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent.

Traceable: One should be able to trace a requirement to the design component and then to coding and testing

Testable: SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

Verifiable: SRS is verifiable if every stated requirement can be verified

Date.
Page No.

SRS document structure

Introduction
Purpose, Document Conventions, Intended Audience, and Reading suggestions, Project Scope, References.

Overall Description
Product Perspective, Product features, User Classes, and characteristics, Operating Environment, Design and Implementation Constraints, User Documentation, Assumptions and Dependencies

System Features
System Feature1, System feature2 (and so on)

External Interface Requirements
User Interfaces, Hardware Interfaces, Software Interfaces, Communications Interfaces.

Other Non functional Requirements
Performance Requirements, safety requirements security requirements, software quality attributes

Other Requirements
Appendix A: Glossary
Appendix B: Analysis Models
Appendix C: Issues list.

Date.
Page No.

Q Describe the requirement analysis process and explain its steps

Q Diff btw functional & non-functional requirements

Q List functional & non-functional requirements for

Hostel MS
Result MS and University
Library MS

Date.
Page No.

Software design

The design phase begins after the completion of the requirement analysis.

The design specifies how to develop the system so that it meets requirements.

Software design deals with transforming the customer's requirements into a form that can be implemented using a programming language.

Software design concentrates on the solution domain.

Design principles

Problem Partitioning or Divide & Conquer: It means breaking a problem into smaller bite-sized subproblems to reduce the complexity of the problem.

Abstraction: An abstraction uses components at an abstract level by hiding the internal details.

Date.
Page No.

Modularity: Modularity allows for decomposing a problem into sub problems. The software uses separate modules which are differently named and addressed and are integrated later for fully functional software.

Increase cohesion: Cohesion means grouping things that make sense together. The higher the cohesion, the better the design.

Reduce coupling: When modules or subproblems are dependent on each other then it is known as coupling. Good software should have as independent components as possible.

Increase Reusability: A good design would always ensure re-usability.

Design for flexibility: A good design should be flexible and anticipate changes to the system.

Design for portability: While designing a system keep in mind that it may be used on a different platform or device in the future. Hence portability becomes important.

Design for Testability: Designing for testability becomes very imp in large systems with large codebases.

Software Architecture

- It is a blueprint of a system
- It represents a software system in a structured form where all the functional and non-functional requirements are covered.

There are two forms of design software architecture

- Small architecture gives details of components of an individual program
- Large architecture for complex software distributed across several computers owned and managed by different companies.

Preliminary or High-Level Design
Detailed Designed

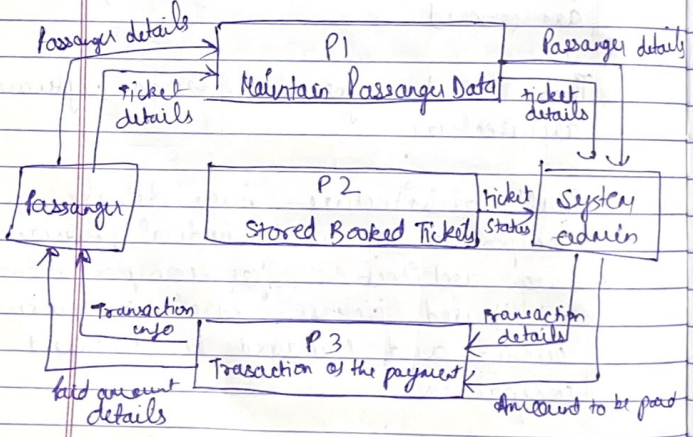
Date.
Page No.

High-level Design: College Management software

Admission, Exam
Departments, laboratories
Library
Hostel
Sports complex, cafeteria

Detailed Design.
Functions and data such as
Admission: Name, RollNo, Marks, Age - - -
Library: RollNo, ID, Name, Book Issued - - -

Software architecture for a ticket reservation

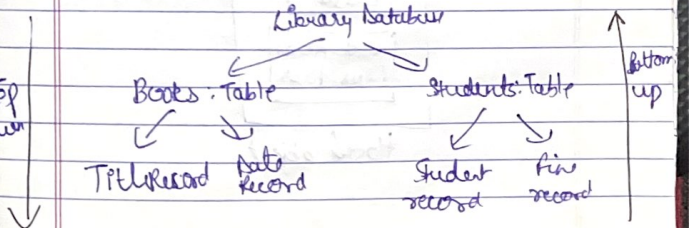


Date.
Page No.

Top-down Design
There are two approaches to level-oriented strategy.

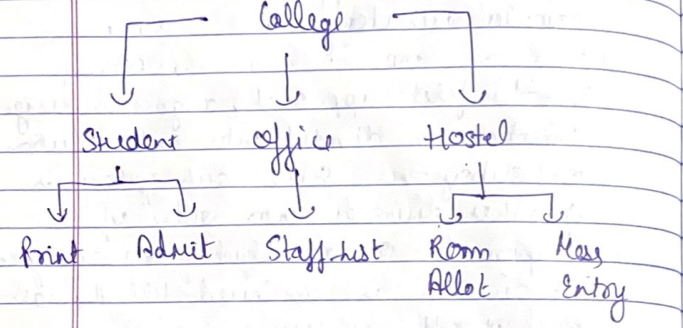
In the first approach, a general design solution is divided into sub-solutions or subsystems. Each sub-system is further divided into sets of components or a collection of sub-systems. The division is continued till the last level of the system.

Bottom-up Design
In the bottom-up design strategy, the lowest level subsystems and components are designed first. The bottom-up design process further combines them into the next abstraction level. This process is continued till a single system is formed that includes all the sub-systems and components

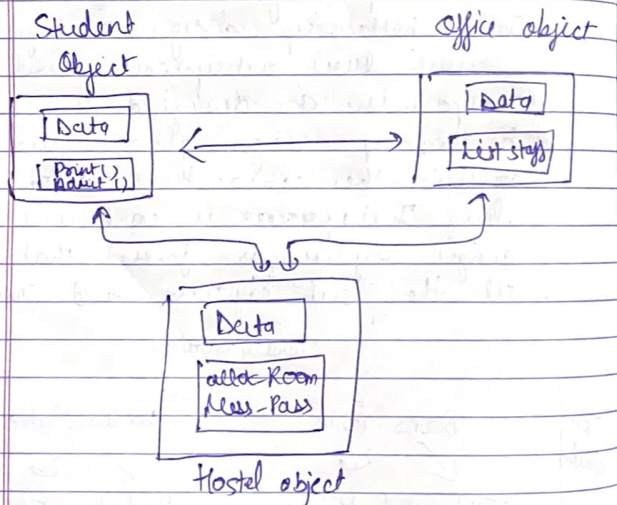


Date.
Page No.

Top-down



Bottom-up



Date.
Page No.

Structured Design Approach

Structured design is generally based on a divide and conquer strategy

It is easy to understand and simple

The small problems are designed using the modules. Modules are systematically organized in a hierarchy.

Modules follows rules of coupling and cohesion

Cohesion- the grouping of all functionally related elements

Coupling - communication b/w different modules

High cohesion and low coupling are desirable

Approaches to Software Design

(1) Function-oriented design

Date.
Page No.

The design is divided into a collection of units based on the function. Each unit is defined with a task and several units interact with each other. There are three steps in functional design.

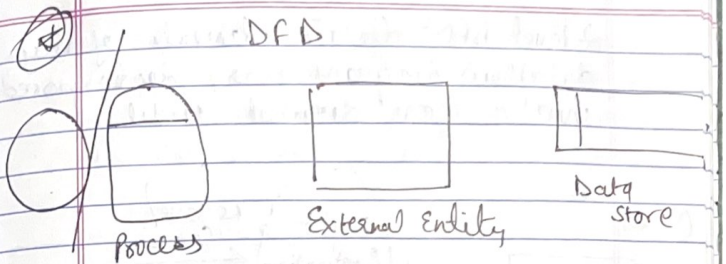
Design of data flows: In this process the processing of data is shown using data flow diagrams (DFDs).

Structure division: In this process how the given project (system) is divided into different parts (sub systems) is shown using structure objects.

Detailed design: In this process, the components, interfaces, and relationships among the subsystems are described. A data dictionary tool is used here.

Date.
Page No.

DFD



Data flows

DFD shows the system requirements in a graphical manner.

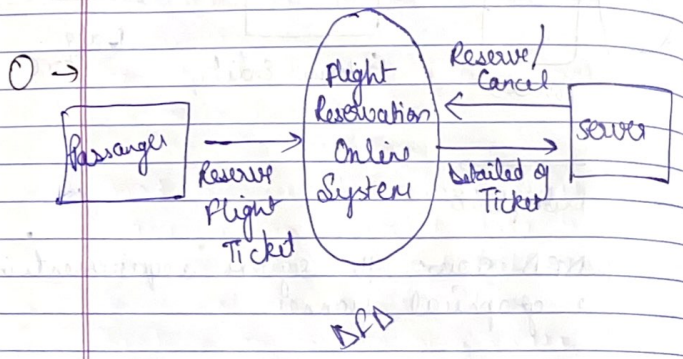
DFD expresses the functions or activities in the system and data flows. Rounded rectangles, circles, and arrows are the symbols used in DFDs.

0-level DFD/Context Diagram: Presents an overview of the system and its interaction with the rest of the world

1-Level DFD: Presents a more detailed view of the system than context diag by showing the main sub processes and stores of data.

Date.
Page No.

2-level DFD: Certain elements of any dataflow diagram are decomposed into a more detailed model.



DFD

Date.
Page No.

Structure Charts

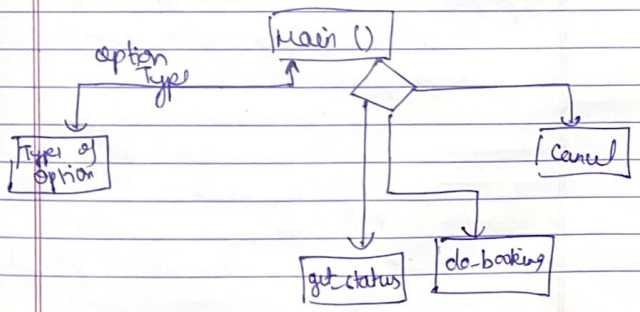
Used in function-oriented design. Structure charts are derived from the DFDs. It decomposes the problem into subproblems to show the hierarchical relationship.

Eg Organization Chart

Modules are represented by rectangular boxes. Arrows with annotations are used to pass control from one module to another with a given direction

Library modules are represented by a rectangle with double edges.

Diamond for the selection command and loops are represented using the repitition around the control flow.



Structure chart

Date.
Page No.

(2) Object - oriented design

OOD is a conceptual model where a system is viewed as a collection of objects. The object is an entity in a solution.

Eg offices, users, schools, shops, companies etc.

Object is described using class. A class is an abstract data type that involves attributes and methods.

Object is an instance of a class.

Date.
Page No.

Function - oriented	Object - oriented

Date.
Page No.

UML

Unified Modelling Language

The UML notations are used in OOD models.

It is now a de facto standard for OO modeling.
↓
the standard that is commonly used in the market

The notations in UML are divided into three categories

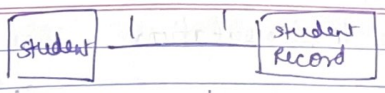
1) Things include notations for class, object, interface, etc
2) Relationships are used in showing the behavior.
3) Diagrams are used to show the structure, behavior, and interaction in a system.

Relationships

- Association
- Inheritance
- Implementation
- Dependency
- Aggregation
- Composition

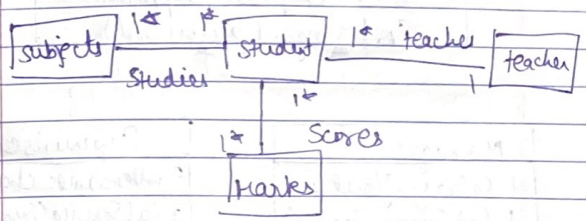
Date.
Page No.

Association

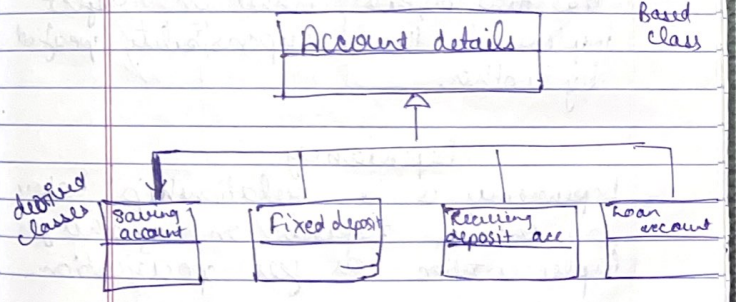
eg 

Association: known as "has a" relationship

A student has a record, a bus has an engine and wheels

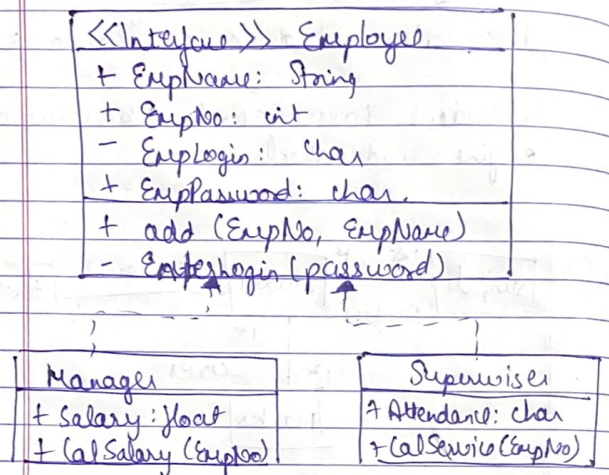
eg 

Generalization / Inheritance



Date.
Page No.

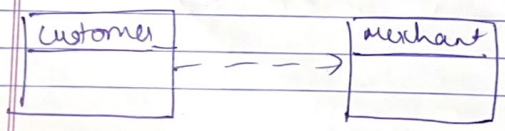
Implementation



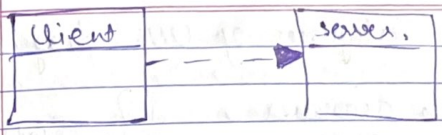
Implementation is a relationship b/w two objects, where one object implements the responsibility specified by another.

Dependency

Dependency is a relationship where one entity depends on another for implementation or for specification.

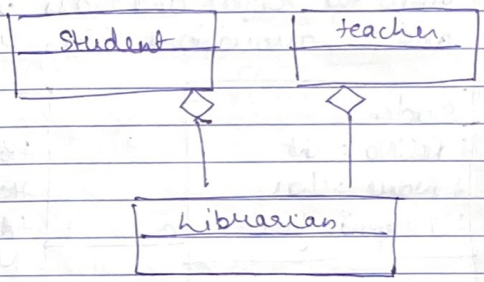


Date.
Page No.



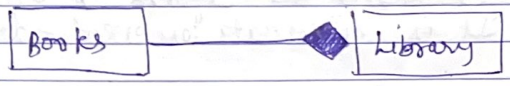
Aggregation

Aggregation denotes the combination of more than one object that is configured together



Composition

Composition implies one class containing another. The contained class cannot exist independently. eg a library cannot exist without books. as shown below



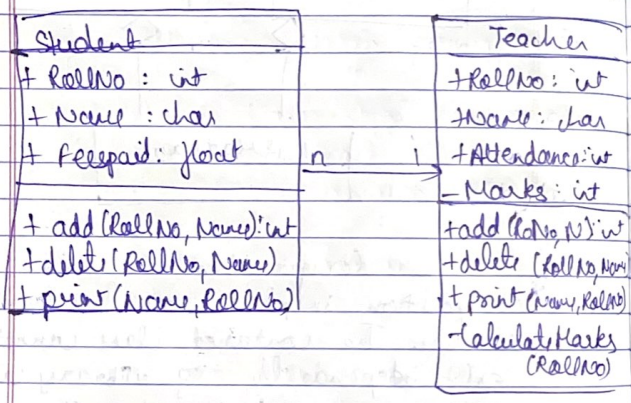
Date.
Page No.

Categories of UML diagrams

Structure diagrams
Behavioural diagrams
Interaction diagrams
} - Types

Structure Diagrams

It includes class diagrams, which show the object and class in the system and the associations b/w these class.



Behavioural Diagrams

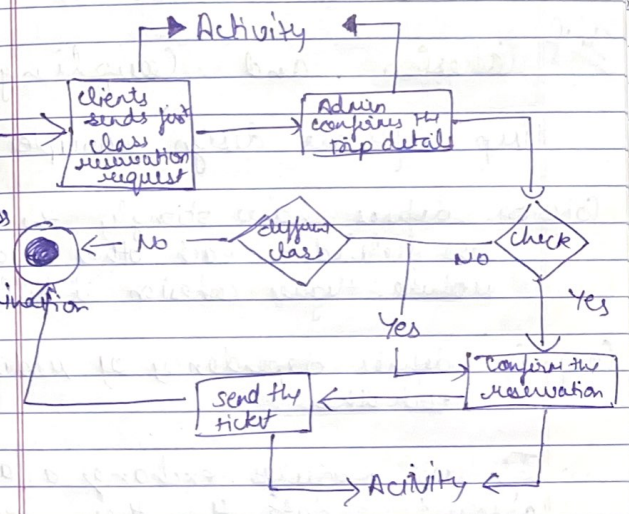
It shows the system in execution. It is either data or event - driven

eg a telephone bill is generated using calls and Internet usage of a customer.
eg of system responding to data or data-driven modelling

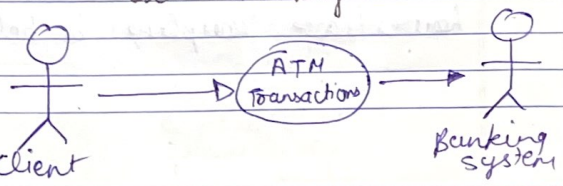
Date.
Page No.

when a landline telephone switching office responds to "dial a no", "keep the receiver on the hook" This is an eg of event driven modelling.

eg Activity diagram - data driven



Interaction Diagrams



Date.
Page No.

Use case diagrams to depict the interaction of how the system works with actors and test cases.

A use case is shaven as an eclipse with the actors involved in the use case represented as stick figures

Imp Cohesion and Coupling.

Imp software design principles.

Cohesion: defines how strongly the elements are related to each other in a module. Higher cohesion is better

Coupling: defines dependency of modules on each other

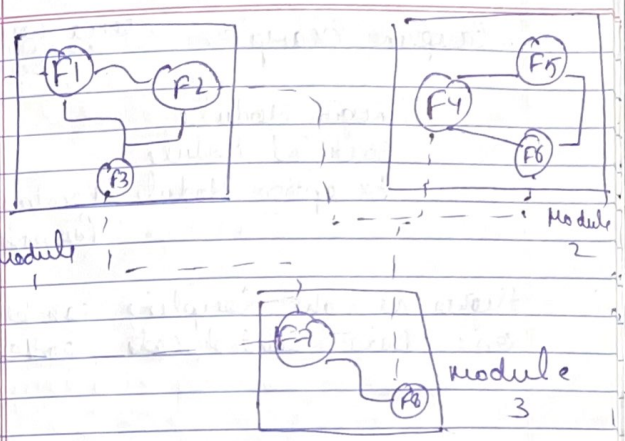
If two modules exchange a large amount of data, then they are highly coupled.

Lower/Loose coupling is better

High

Low

Date.
Page No.



coupling
cohesion

Cohesion Example Higher Cohesion is Better

Student
Name
Roll No
PrintRecord (Name, RollNo)
printResults (RollNo)
print Attendance (RollNo)

Student
checkEmail ()
Print ()
Print Stafflist ()
Calculate Fees ()
CheckName ()

Date.
Page No.

Coupling Example — lower coupling is Better

Login Module
Back End Module
Exception Module (For Incorrect Password etc)

Higher or tight Coupling can be based on: Data, control, code, content, etc.

Date.
Page No.

Functional Independence

A module with high cohesion and low coupling is said to be functionally independent of other modules.

A functionally independent module performs a single task without much interaction with other modules.

Functional independence is the key to good software design

With functional independence errors can be isolated, more scope for reuse and improves understandability

Design Verification and Inspection process

Design Verification

The design output must be verified before giving it to the next phase.

There are multiple tools available for checking the inconsistency

Design review method is used for the verification.

Date.
Page No.

The aim of verification is to ensure that all the requirements are fulfilled and of good quality.

Errors or faults are detected and removed in this step.

Inspection process

The inspection process for the design review is done by a group of people checking for errors and undesirable properties.

The makers of general and detailed design, authors of SRS, and software engineers conduct a meeting to detect errors.

A formal report is prepared at the end of the design review process uncovering any unattended requirements or defects.