Sri Venkateswara College of Engineering

CS22502 ---Software Engineering

by

P.KAVIYA

Assistant Professor

Unit 2:

- UNIT II REQUIREMENTS ANALYSIS AND MODELING 9

  Software Requirements: Functional and Non-Functional, User requirements, System requirements, Software Requirements Document –Coupling and Cohesion- Requirement Engineering Process: Feasibility Studies, Requirements elicitation and analysis, requirements validation, requirements management. Classical analysis: Structured system Analysis, Petri Nets- Data Dictionary.

Requirements engineering

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.

- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

What is a requirement?

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

- This is inevitable as requirements may serve a dual function
  - May be the basis for a bid for a contract - therefore must be open to interpretation;
  - May be the basis for the contract itself - therefore must be defined in detail;
  - Both these statements may be called requirements.

Requirements abstraction (Davis)

"If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organisation's needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the *requirements document* for the system."

Types of requirement

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

Definitions and specifications

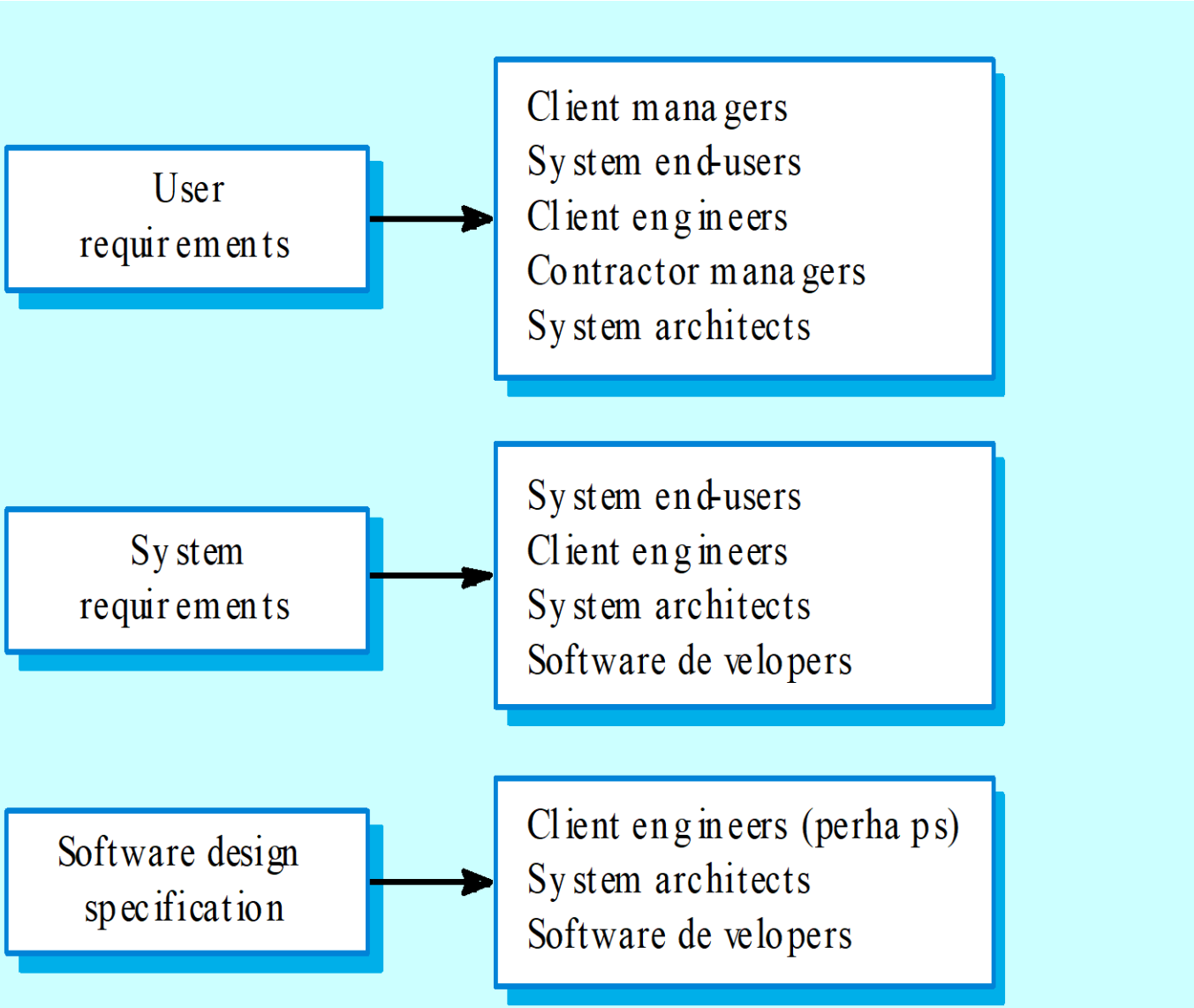
User requirement definition

1. The software must provide a means of representing and accessing external files created by other tools.

System requirements specification

1.1 The user should be provided with facilities to define the type of external files.
1.2 Each external file type may have an associated tool which may be applied to the file.
1.3 Each external file type may be represented as a specific icon on the user's display.
1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.
1.5 When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

Requirements readers



Functional and non-functional requirements

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- Non-functional requirements
  - constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Domain requirements
  - Requirements that come from the application domain of the system and that reflect characteristics of that domain.

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.

The LIBSYS system

- A library system that provides a single interface to a number of databases of articles in different libraries.
- Users can search for, download and print these articles for personal study.

Examples of functional requirements

- The user shall be able to search either all of the initial set of databases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area.

Requirements imprecision

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term ‘appropriate viewers’
  - User intention - special purpose viewer for each different document type;
  - Developer interpretation - Provide a text viewer that shows the contents of the document.

Requirements completeness and consistency

- In principle, requirements should be both complete and consistent.
- Complete
  - They should include descriptions of all facilities required.
- Consistent
  - There should be no conflicts or contradictions in the descriptions of the system facilities.
- In practice, it is impossible to produce a complete and consistent requirements document.

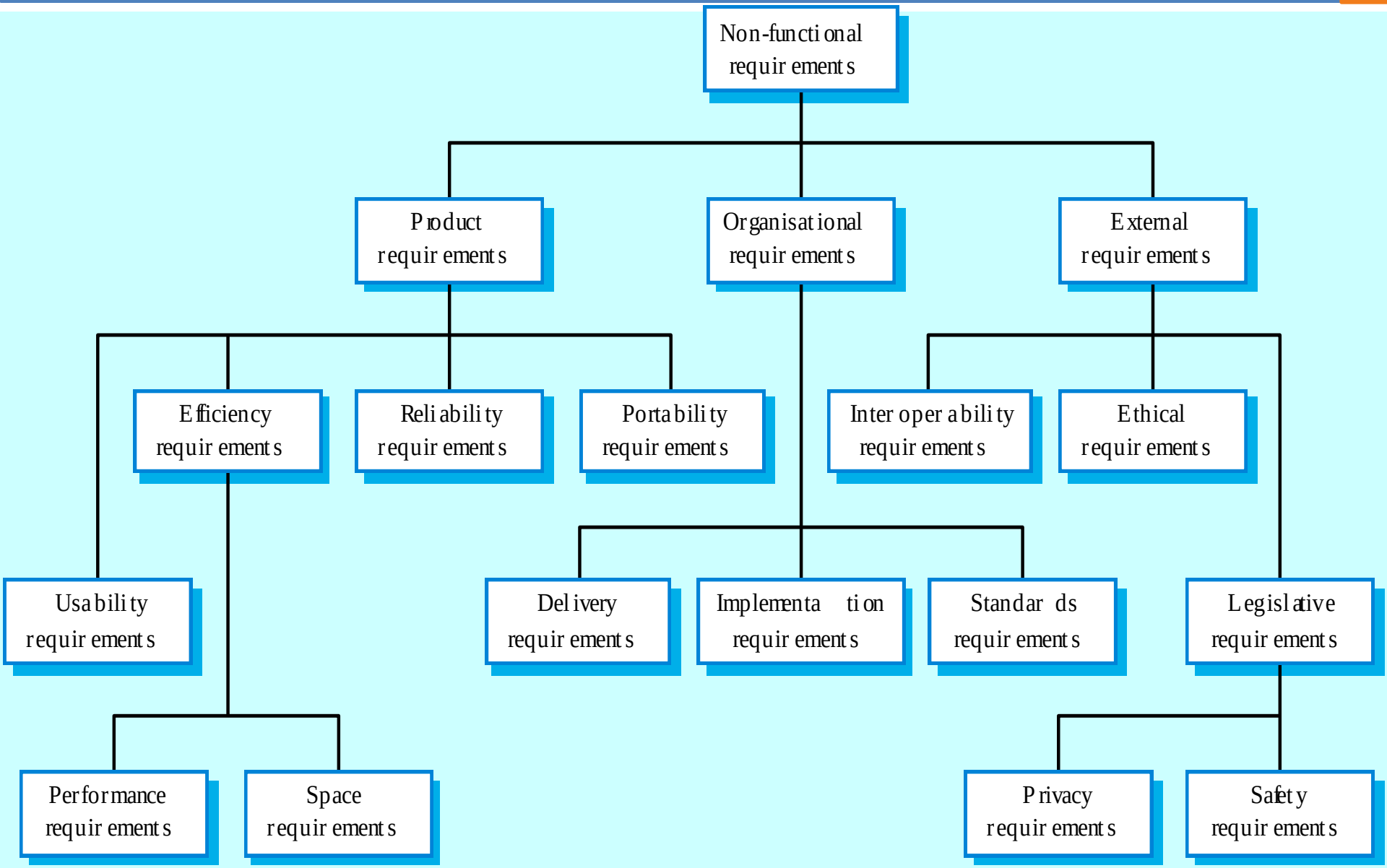
Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular CASE system, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Non-functional requirement types



Non-functional requirements examples

- Product requirement

  8.1 The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.

- Organisational requirement

  9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.

- External requirement

  7.6.5 The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.

Goals and requirements

- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
- Goal
  - A general intention of the user such as ease of use.
- Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested.
- Goals are helpful to developers as they convey the intentions of the system users.

Examples

- **A system goal**
  - The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised.
- **A verifiable non-functional requirement**
  - Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Requirements measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	M Bytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Requirements interaction

- Conflicts between different non-functional requirements are common in complex systems.
- Spacecraft system
  - To minimise weight, the number of separate chips in the system should be minimised.
  - To minimise power consumption, lower power chips should be used.
  - However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?

Domain requirements

- Derived from the application domain and describe system characteristics and features that reflect the domain.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Library system domain requirements

- There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.
- Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

Domain requirements problems

- Understandability
  - Requirements are expressed in the language of the application domain;
  - This is often not understood by software engineers developing the system.
- Implicitness
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

User requirements

- Should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
- User requirements are defined using natural language, tables and diagrams as these can be understood by all users.

Problems with natural language

- Lack of clarity
  - Precision is difficult without making the document difficult to read.
- Requirements confusion
  - Functional and non-functional requirements tend to be mixed-up.
- Requirements amalgamation
  - Several different requirements may be expressed together.

Guidelines for writing requirements

- Invent a standard format and use it for all requirements.
- Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.

System requirements

- More detailed specifications of system functions, services and constraints than user requirements.
- They are intended to be a basis for designing the system.
- They may be incorporated into the system contract.

Requirements and design

- In principle, requirements should state what the system should do and the design should describe how it does this.
- In practice, requirements and design are inseparable
  - A system architecture may be designed to structure the requirements;
  - The system may inter-operate with other systems that generate design requirements;
  - The use of a specific design may be a domain requirement.

Problems with NL specification

- Ambiguity
  - The readers and writers of the requirement must interpret the same words in the same way. NL is naturally ambiguous so this is very difficult.
- Over-flexibility
  - The same thing may be said in a number of different ways in the specification.
- Lack of modularisation
  - NL structures are inadequate to structure system requirements.

Alternatives to NL specification

Notation	Description
Structured natural language	This approach depends on defining standard forms or templates to express the requirements specification.
Design description languages	This approach uses a language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system. This approach is not now widely used although it can be useful for interface specifications.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. An early example of such a graphical language was SADT. Now, use-case descriptions and sequence diagrams are commonly used .
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

Structured language specifications

- The freedom of the requirements writer is limited by a predefined template for requirements.
- All requirements are written in a standard way.
- The terminology used in the description may be limited.
- The advantage is that the most of the expressiveness of natural language is maintained but a degree of uniformity is imposed on the specification.

Form-based specifications

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Indication of other entities required.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

Form-based node specification

Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: Safe sugar level

Description Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2), the previous two readings (r0 and r1)

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose Š the dose in insulin to be delivered

Destination Main control loop

Action: CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requires Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition The insulin reservoir contains at least the maximum allowed single dose of insulin..

Post-condition r0 is replaced by r1 then r1 is replaced by r2

Side-effects None

Tabular specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.

Tabular specification

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2-r1)<(r1-r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing. ((r2-r1) □ (r1-r0))	CompDose = round ((r2-r1)/4) If rounded result = 0 then CompDose = MinimumDose

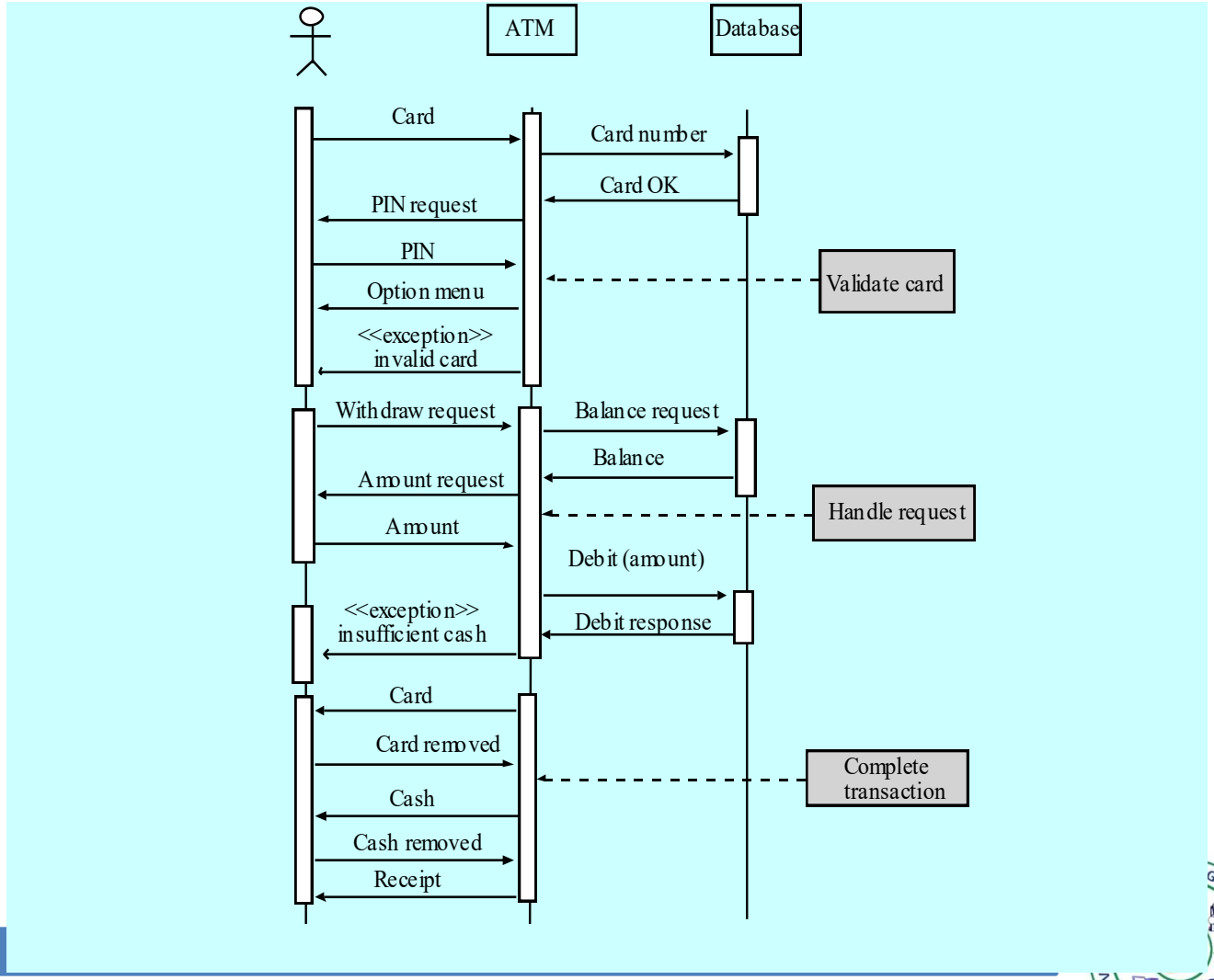
Graphical models

- Graphical models are most useful when you need to show how state changes or where you need to describe a sequence of actions.

Sequence diagrams

- These show the sequence of events that take place during some user interaction with a system.
- You read them from top to bottom to see the order of the actions that take place.
- Cash withdrawal from an ATM
  - Validate card;
  - Handle request;
  - Complete transaction.

Sequence diagram of ATM withdrawal



Interface specification

- Most systems must operate with other systems and the operating interfaces must be specified as part of the requirements.
- Three types of interface may have to be defined
  - Procedural interfaces;
  - Data structures that are exchanged;
  - Data representations.
- Formal notations are an effective technique for interface specification.

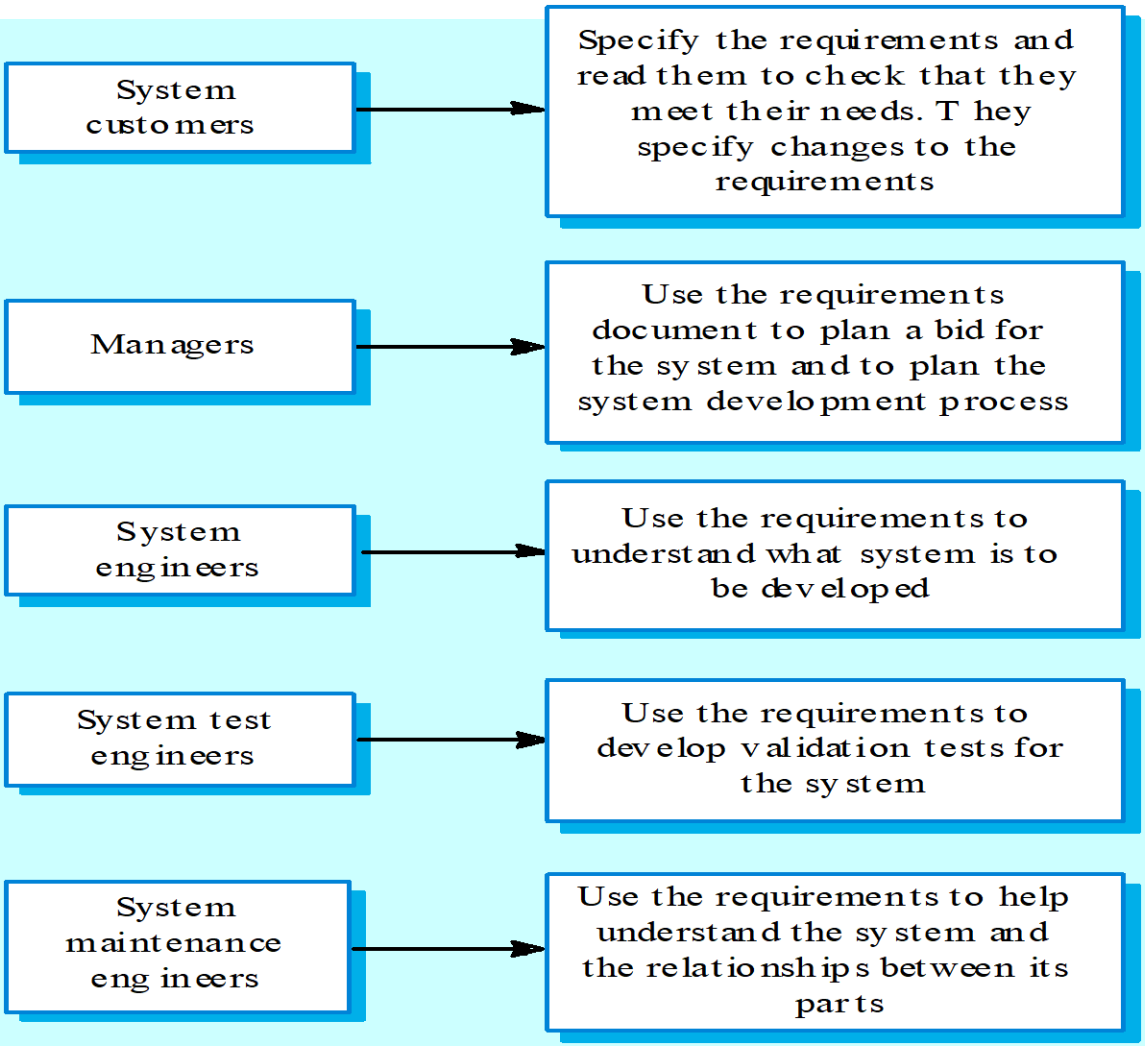
PDL interface description

```
interface PrintServer {

// defines an abstract printer server
// requires:          interface Printer, interface PrintDoc
// provides: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter

        void initialize ( Printer p ) ;
        void print ( Printer p, PrintDoc d ) ;
        void displayPrintQueue ( Printer p ) ;
        void cancelPrintJob (Printer p , PrintDoc d) ;
        void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;
} //PrintServer
```

The requirements document

- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it

Users of a requirements document



IEEE requirements standard

- Defines a generic structure for a requirements document that must be instantiated for each specific system.
  - Introduction.
  - General description.
  - Specific requirements.
  - Appendices.
  - Index.

Requirements document structure

- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

Key points

- Requirements set out what the system should do and define constraints on its operation and implementation.
- Functional requirements set out services the system should provide.
- Non-functional requirements constrain the system being developed or the development process.
- User requirements are high-level statements of what the system should do. User requirements should be written using natural language, tables and diagrams.

Key points

- System requirements are intended to communicate the functions that the system should provide.
- A software requirements document is an agreed statement of the system requirements.
- The IEEE standard is a useful starting point for defining more detailed specific requirements standards.

ONLINE FOOD DELIVERY SYSTEM

TABLE OF CONTENTS

1. Introduction

The purpose of this document is to identify unambiguously the user requirements and clearly define both functional and nonfunctional requirements of the Online Food Delivery System. In addition, this document is intended to cover technical goals as well as objectives of the proposed System.

1.1 Purpose

The main purpose of this project is to develop the most cost-effective and user-friendly food delivery system system. The primary purpose of an online food ordering system is to allow customers to easily place orders from a restaurant over the internet.

1.2 Document Conventions

The following typographical conventions are used in this document.

Main Section Titles

Specification

For

Version 1.0 approved

Heading
•Font: Times New Roman
• Face: Bold
• Size: 20

Subsection Titles
• Font: Times New Roman
• Face: Bold
• Size: 14 Other Text Explanations
• Font: Times New Roman

Body
• Font: Times New Roman
• Face: Normal
• Size: 14

1.3 Intended Audience and Reading Suggestions

The first section of SRS gives a brief idea of the proposed food delivery system and what is the need behind having it.

The second section shows the way to overall description of application, functions, perspective, operating environment, design and implementation constraints, data inputs required.

The third section is written with a goal to show the various system features in detail. The subsections give an elaborate description of individual features.

The fourth section explores the various descriptions of external interfaces such as user interface, hardware interface and software interface.

The fifth section focuses on details of non-functional requirements such as security requirements, safety requirements etc.

1.4 Product Scope

The Scope of the project (Application) are as follows: Food Ordering app can sell Food product, preferred brands, kitchen needs, essential restaurant supplies and more, through this online, onestop Food store. It provides you with a convenient way to sell from your Food shopping app. You can use this app as one big supermarket app to sell products in your store. This app makes it easy for users to buy products from the store with easy steps and the store can get easy orders.

2. Overall Description

2.1 Product Perspective

The users are provided with a graphical interface through which they can interact with the system. The application helps users to list searched results, process order, payment, modification and cancellation to existing orders. Unlike in the previous stage, people have to walk to restaurants for food. This problem is overcome by introducing this application. This project will provide an option to customers to order food online and to check the progress online.

2.2 Product Functions

The delivery system is designed using Flutter. And developed using the Dart programming language. All the record stores in the Firebase database.

2.3 User Classes and Characteristics

The system provides access to Customers and Administrators.

• Customer: Customers can check availability of the food and also can make payment online. They can also view their ordering details, payment details etc.

• Administrator: Administrator can manage the menu, food details, pricing etc. Also admin can view and print transaction reports, order reports, cancellation reports, etc.

2.4 Operating Environment

2.4.1 Software Interfaces

- Flutter
- Database: Firebase
- Operating System: Android or IOS

2.4.2 Hardware Interfaces

- User Side: Smartphone
- Server: Google Cloud services

2.5 Design and Implementation Constraints

- Customer not has any rights to edit or delete booking and payment records
- The online payment gateway is a virtual payment gateway for testing purposes.
- System is limited to HTTP/HTTPS Protocols.
- Basic computer knowledge must be required.
- Internet connection required

2.6 User Documentation

• Final release will be accompanied with a user guide to inform new users how to use the e Travel agency website.

• The system will be designed as user friendly as possible

2.7 Assumptions and Dependencies

• The user should operate smartphones along with internet facility.
• Roles and responsibilities are already established.
• Administrator is already created.

3. External Interface Requirements

The following sections will introduce the numerous requirements of the system from the point of view of different users and will introduce a number of

decisions that have been made regarding implementation. These sections also attempt to somewhat describe the role of each user group in the system, discussing their individual roles through the functions they can perform.

3.1 User Interfaces

The user interface for this system will have to be simple and clear. Most importantly, the pages must be easy to read, easy to understand and accessible. The color scheme should be appropriate to provide familiarity with the travel site and there should be no contrast issues.

3.2 Hardware Interface

The system will run on a smartphone with a reliable internet connection.

3.3 Software Interface

The software runs on android or ios devices.

3.4 Communications Interfaces

Internet connection is required to establish a connection between backend and to the server.

4. System Features

4.1 Functional Requirements

The structure of the system can be divided into 3 main logical components:

• Web Ordering System- provides the functionality for customers to place their order and supply necessary details.

• Menu Management-allows the restaurant to control what can be ordered by the customers

• Order Retrieval System-This is a final logical component. Allows restaurant to keep track of all orders placed. This component takes care of order retrieving and displaying order information.

Product Function:

The Online Food Order System application would have the following basic functions:

Ordering System Module

This module provides the functionality for customers to place their order and supply necessary details. Users of the system, namely restaurant customers, must be provided the following functionality:

- Create an account.
- Manage their account.
- Log in to the system.
- Navigate the restaurant's menu.
- Select an item from the menu.
- Add an item to their current order.
- Review their current order.
- Remove an item/remove all items from their current order.
- Provide payment details.
- Place an order.
- Receive confirmation in the form of an order number.
- Receive confirmation in the form of an order number.
- View order placed.

Additional Feature:

- eClub- Allows users to subscribe to eClub to get promotional deals and discounts offers.

Out of all the functions outlined above, Account Creation and Management only will be used every time a customer places an order. This will allow us to simplify the overall user experience.

Menu Management System Module

This module provides functionality for the power user-Administrator only. It will not be available to any other users of the system like Restaurant Employees or Customers. Using a graphical interface, it will allow an Admin to manage the menu that is displayed to users of the web ordering system:

- Add/update/delete food category to/from the menu.
- Add /update/delete food items to/from the menu.
- Update price for a given food item.
- Update additional information (description, photo, etc.) for a given food item.

Before customers can actually use this system, functionality provided by this component will have to be configured first. Once the initial configuration is done, this will be the least likely used component as menu updates are mostly seasonal and do not occur frequently.

Order Retrieval System Module

This is the most simplest module out of all 3 modules. It is designed to be used only by restaurant employees, and provides the following functions:

- Retrieve new orders from the database.
- Display the orders in an easily readable, graphical way

Menu Management System Module

This module provides functionality for the power user-Administrator only. It will not be available to any other users of the system like Restaurant Employees or Customers. Using a graphical interface, it will allow an Admin to manage the menu that is displayed to users of the web ordering system:

- Add/update/delete food category to/from the menu.
- Add /update/delete food items to/from the menu.
- Update price for a given food item.
- Update additional information (description, photo, etc.) for a given food item.

Before customers can actually use this system, functionality provided by this component will have to be configured first. Once the initial configuration is done, this will be the least likely used component as menu updates are mostly seasonal and do not occur frequently.

Order Retrieval System Module

This is the most simplest module out of all 3 modules. It is designed to be used only by restaurant employees, and provides the following functions:

- Retrieve new orders from the database.
- Display the orders in an easily readable, graphical way

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Some Performance requirements identified are listed below:

- The database shall be able to accommodate a minimum of 10,000 records of customers.
- The software shall support use of multiple users at a time.

There are no other specific performance requirements that will affect development

5.2 Security Requirements

Some of the factors that are identified to protect the software from accidental or malicious access, use, modification, destruction, or disclosure are described below. Specific requirements in this area could include the need to:

- Utilize certain cryptographic techniques
- Keep specific log or history data sets
- Assign certain functions to different modules
- Restrict communications between some areas of the program
- Check data integrity for critical variables
- Later version of the software will incorporate encryption techniques in the user/license authentication process.
- The software will include an error tracking log that will help the user understand what error occurred when the application crashed along with suggestions on how to prevent the error from occurring again.
- Communication needs to be restricted when the application is validating the user or license. (i.e., using https).

5.3 Software Quality Attributes

5.3.1 Portability

The user will be able to reset all options and all stored user variables to default settings.

5.3.2 Reliability

Some of the attributes identified for the reliability are listed below:

• All data storage for user variables will be committed to the database at the time of entry.

• Data corruption is prevented by applying the possible backup procedures and techniques.

6. Other Requirements

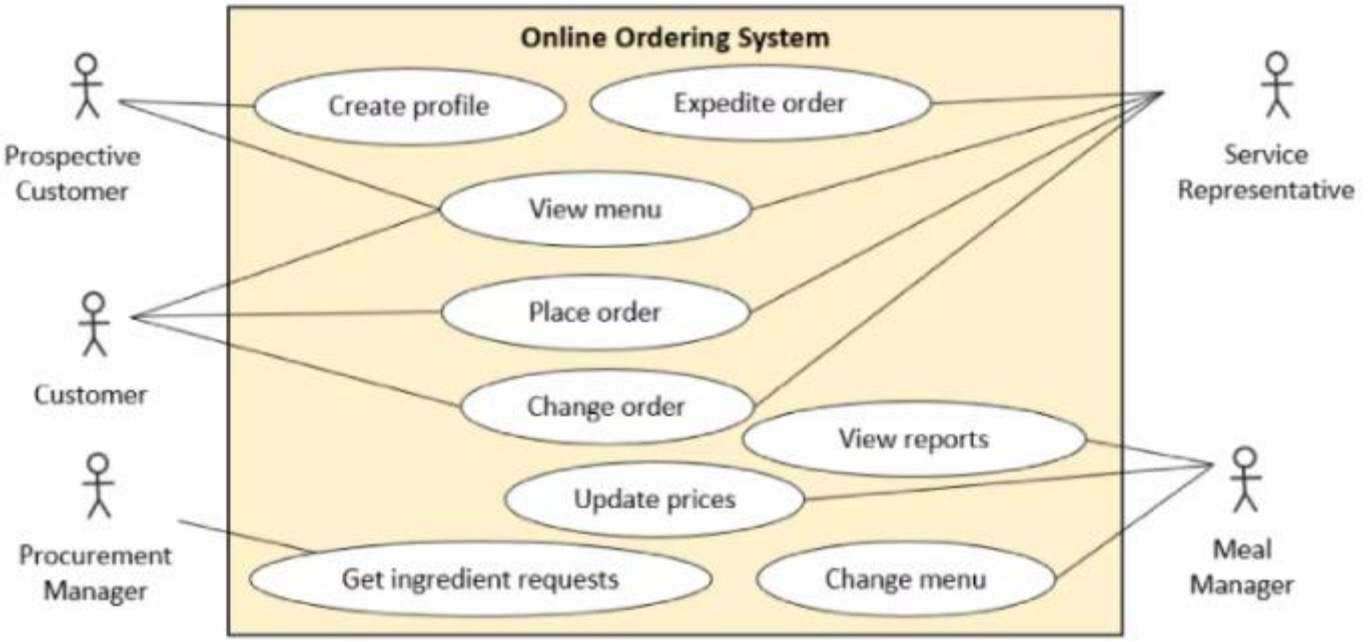
• Delivery people are required to have a great deal of knowledge about various destinations around the area.

•On Time Delivery: Clients now expect their orders speedier than at any other time. In the past, they were restricted to driving through places to get their food rapidly.

Appendix A: Glossary

• Database: A database is an organized collection of data, so that it can be easily accessed and managed. You can organize data into tables, rows, columns, and index it to make it easier to find relevant information.
• Operating System: An operating system is a software which acts as an interface between the end user and computer hardware
• Front-end: It is the modules or areas of software or website that are seen and directly used by the user.
• Back-end: It is the part of a software system or online service that the user does not interact with and that is usually accessible only to programmers or administrators

Appendix B: Analysis Models



Draw use case diagram form online booking system

2. Draw use case diagram for ATM and Library management system

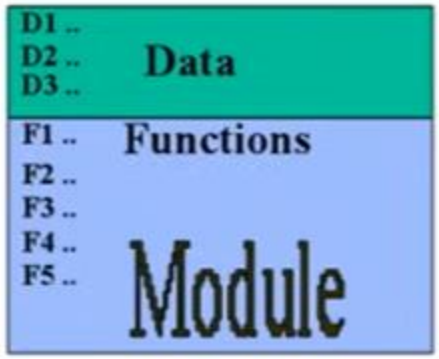
3. Draw DFD diagram food ordering system and library management system

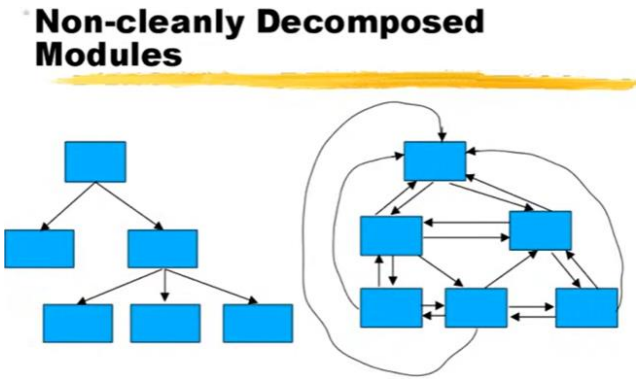
Coupling and Cohesion

- Low coupling means the different parts of the software don't rely too much on each other, which makes it safer to make changes without causing unexpected problems.
- High cohesion means each part of the software has a clear purpose and sticks to it, making the code easier to work with and reuse.

Modularization

- Modularization is the process of dividing a software system into multiple independent modules where each module works independently.
- There are many advantages of Modularization in software engineering. Some of these are given below:
- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.





Fan-in:

indicates how many modules directly invoke a given module.

High fan-in represents code reuse and is in general encouraged.

Depth:

number of levels of control

Width:

overall span of control.

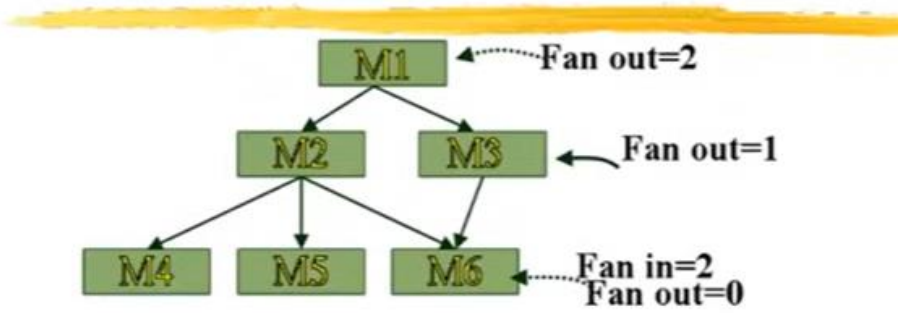
Fan-out:

a measure of the number of modules directly controlled by given module.

Indegree- Fan-in

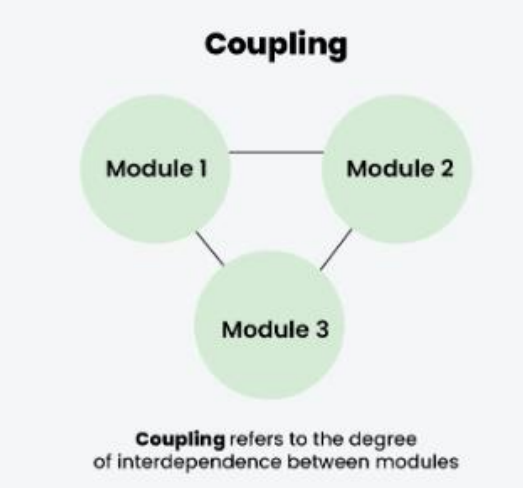
Outdegree- Fan-out

Module Structure

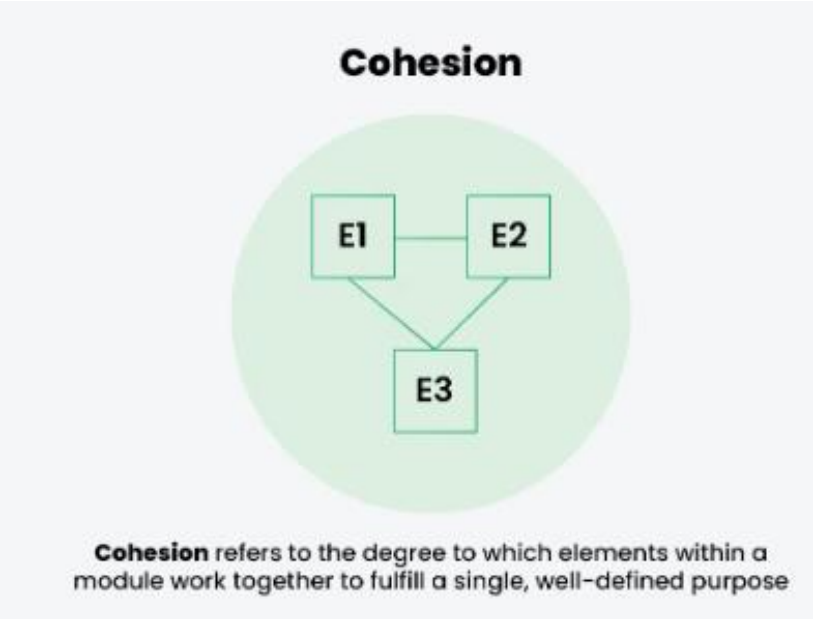


What is Coupling and Cohesion?

- **Coupling** refers to the degree of interdependence between software modules. High coupling means that modules are closely connected and changes in one module may affect other modules.
- Low coupling means that modules are independent, and changes in one module have little impact on other modules.



Cohesion refers to the degree to which elements within a module work together to fulfill a single, well-defined purpose.
High cohesion means that elements are closely related and focused on a single purpose, while low cohesion means that elements are loosely related and serve multiple purposes.



- Both coupling and cohesion are important factors in determining the maintainability, scalability, and reliability of a software system.
- High coupling and low cohesion can make a system difficult to change and test, while low coupling and high cohesion make a system easier to maintain and improve.

Basically, design is a two-part iterative process.

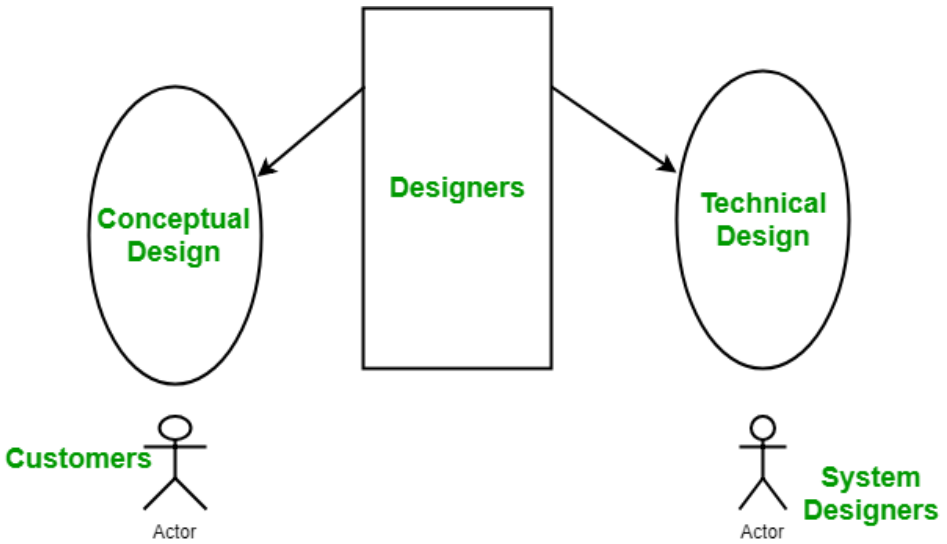
- The first part is **Conceptual Design** which tells the customer what the system will do.
- Second is **Technical Design** which allows the system builders to understand the actual hardware and software needed to solve a customer's problem.

Conceptual design of the system:

- Written in simple language i.e. customer understandable language.
- Detailed explanation about system characteristics.
- Describes the functionality of the system.
- It is independent of implementation.
- Linked with requirement document.

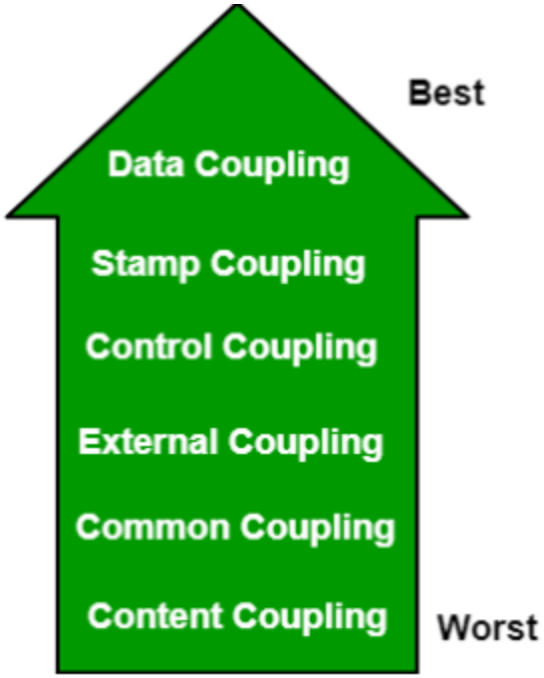
Technical Design of the System:

- Hardware component and design.
- Functionality and hierarchy of software
- components.
- Software architecture
- Network architecture
- Data structure and flow of data.
- I/O component of the system.
- Shows interface.



Types of Coupling

- *Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.*



Types of Coupling

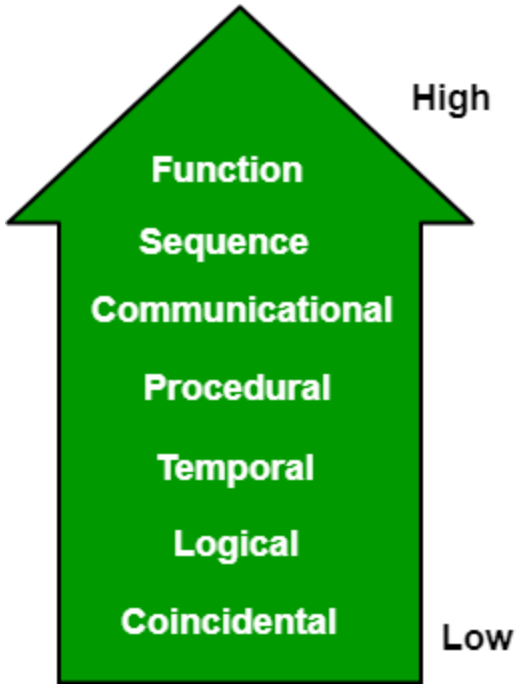
Types of Coupling:

- **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don't contain tramp data. Example-customer billing system.
- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves stamp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
- **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

- **Temporal Coupling:** Temporal coupling occurs when two modules depend on the timing or order of events, such as one module needing to execute before another. This type of coupling can result in design issues and difficulties in testing and maintenance.
- **Sequential Coupling:** Sequential coupling occurs when the output of one module is used as the input of another module, creating a chain or sequence of dependencies. This type of coupling can be difficult to maintain and modify.
- **Communicational Coupling:** Communicational coupling occurs when two or more modules share a common communication mechanism, such as a shared message queue or database. This type of coupling can lead to performance issues and difficulty in debugging.
- **Functional Coupling:** Functional coupling occurs when two modules depend on each other's functionality, such as one module calling a function from another module. This type of coupling can result in tightly-coupled code that is difficult to modify and maintain.
- **Data-Structured Coupling:** Data-structured coupling occurs when two or more modules share a common data structure, such as a database table or data file. This type of coupling can lead to difficulty in maintaining the integrity of the data structure and can result in performance issues.
- **Interaction Coupling:** Interaction coupling occurs due to the methods of a class invoking methods of other classes. Like with functions, the worst form of coupling here is if methods directly access internal parts of other methods. Coupling is lowest if methods communicate directly through parameters.

Types of Cohesion

- *Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.*



Types of Cohesion:

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.
-

- **Procedural Cohesion:** This type of cohesion occurs when elements or tasks are grouped together in a module based on their sequence of execution, such as a module that performs a set of related procedures in a specific order. Procedural cohesion can be found in structured programming languages.
- **Communicational Cohesion:** Communicational cohesion occurs when elements or tasks are grouped together in a module based on their interactions with each other, such as a module that handles all interactions with a specific external system or module. This type of cohesion can be found in object-oriented programming languages.
- **Temporal Cohesion:** Temporal cohesion occurs when elements or tasks are grouped together in a module based on their timing or frequency of execution, such as a module that handles all periodic or scheduled tasks in a system. Temporal cohesion is commonly used in real-time and embedded systems.
- **Informational Cohesion:** Informational cohesion occurs when elements or tasks are grouped together in a module based on their relationship to a specific data structure or object, such as a module that operates on a specific data type or object. Informational cohesion is commonly used in object-oriented programming.
- **Functional Cohesion:** This type of cohesion occurs when all elements or tasks in a module contribute to a single well-defined function or purpose, and there is little or no coupling between the elements. Functional cohesion is considered the most desirable type of cohesion as it leads to more maintainable and reusable code.
- **Layer Cohesion:** Layer cohesion occurs when elements or tasks in a module are grouped together based on their level of abstraction or responsibility, such as a module that handles only low-level hardware interactions or a module that handles only high-level business logic. Layer cohesion is commonly used in large-scale software systems to organize code into manageable layers.