★ What is Software engineering ?

Software Engineering is the process of designing, developing, testing, and maintaining software.

Software engineering is an engineering discipline focused on all aspects of software production, from initial specification to maintenance after deployment.

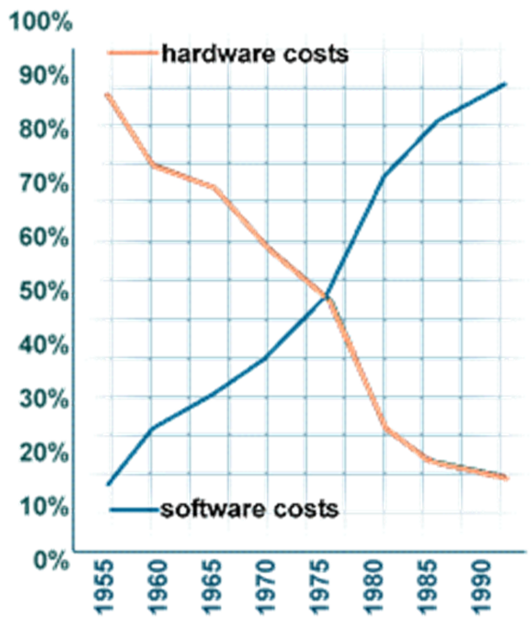
Software engineering is concerned with theories, methods and tools for professional software development.

★ Software Development Cost

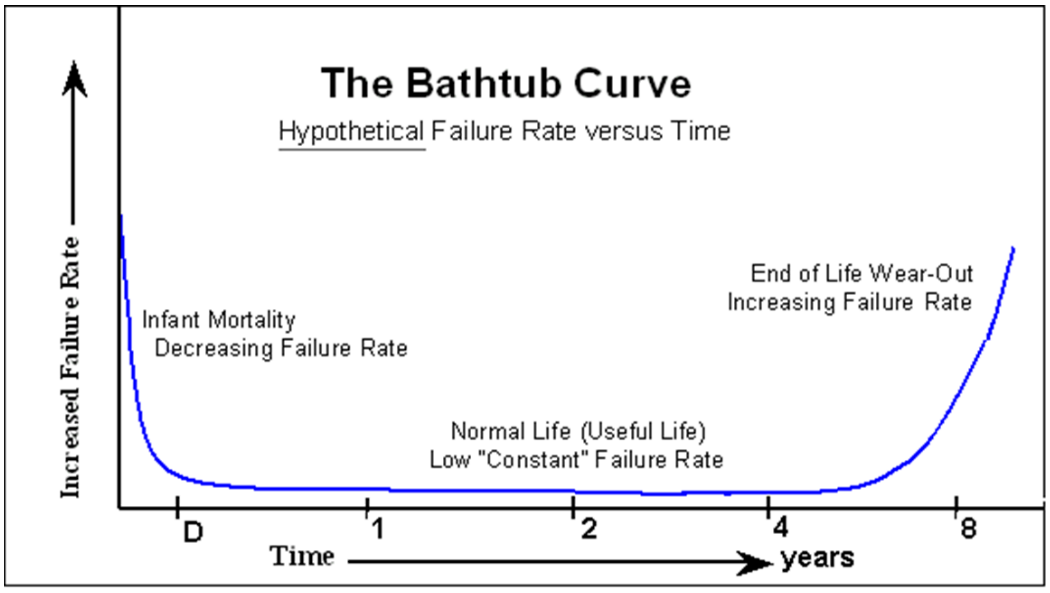
The total expense from the initial concept to the final launch of the software.

Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.

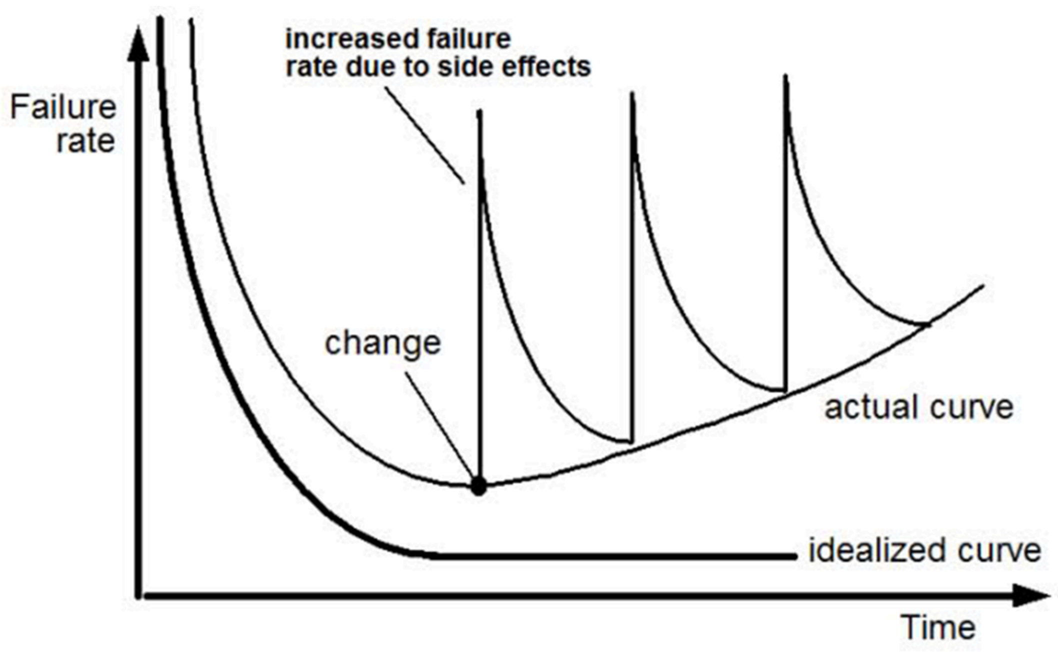
Hardware vs Software cost



Hardware Cost



Software Failure Rate



★ **Software products**

1. **Generic products**

   Stand-alone systems that are marketed and sold to any customer who wishes to buy them.

   Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.

2. **Customized products**

   Software that is commissioned by a specific customer to meet their own needs.

   Examples – embedded control systems, air traffic control software, traffic monitoring systems.

★ **What are the key challenges facing software engineering?**

➔ Coping with increasing diversity.
➔ demands for reduced delivery times.
➔ developing trustworthy software.

★ **What are the costs of software engineering?**

Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.

★ **What are the best software engineering techniques and methods?**

While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of systems.

For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can’t, therefore, say that one method is better than another.

★ **What differences has the web made to software engineering?**

The web has led to the availability of software services and the possibility of developing highly distributed service-based systems.

Web-based systems development has led to important advances in programming languages and software reuse.

★ **Essential attributes of good software**

1. Maintainability
2. Dependability and security
3. Efficiency
4. Acceptability

★ **General issues that affect most software**

1. Heterogeneity
2. Business and social change
3. Security and trust

★ **Application types**

1. **Stand-alone applications**
   These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.

2. **Interactive transaction-based applications**
   Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.

3. **Embedded control systems**
   These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.

4. **Batch processing systems**
   These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.

5. **Entertainment systems**
   These are systems that are primarily for personal use and which are intended to entertain the user.

6. **Systems for modeling and simulation**
   These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.

7. **Data collection systems**
   These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.

8. **Systems of systems**
   These are systems that are composed of a number of other software systems.

★ **What is Cloud computing ?**

Cloud computing is an approach to the provision of computer services where applications run remotely on the 'cloud'.

★ **Software engineering ethics**

1. Software engineering involves wider responsibilities than simply the application of technical skills.
2. Software engineers must act honestly and ethically to be respected as professionals.

Ethical behavior
Ethical behavior is more than simply upholding the law but involves following a set of principles that are morally correct.

★ **Issues of professional responsibility**

1. Confidentiality
2. Competence
3. Intellectual property rights
4. Computer misuse

★ **ACM / IEEE Code of Ethics**

1. Public - Prioritize the public interest.
2. Client and Employer - Serve clients and employers in ways that align with the public interest.
3. Product - Ensure high professional standards for products and modifications.
4. Judgment - Maintain integrity and independence in professional judgment.
5. Management - Promote ethical management of software development and maintenance.
6. Profession - Uphold the integrity and reputation of the profession.
7. Colleagues - Be fair and supportive to colleagues.
8. Self - Commit to lifelong learning and ethical practice.

★ **Many different software processes but all involve:**

➔ **Specification –** defining what the system should do
➔ **Design and implementation –** defining the organization of the system and implementing the system
➔ **Validation –** checking that it does what the customer wants
➔ **Evolution –** changing the system in response to changing customer needs.

★ **Plan-driven processes**

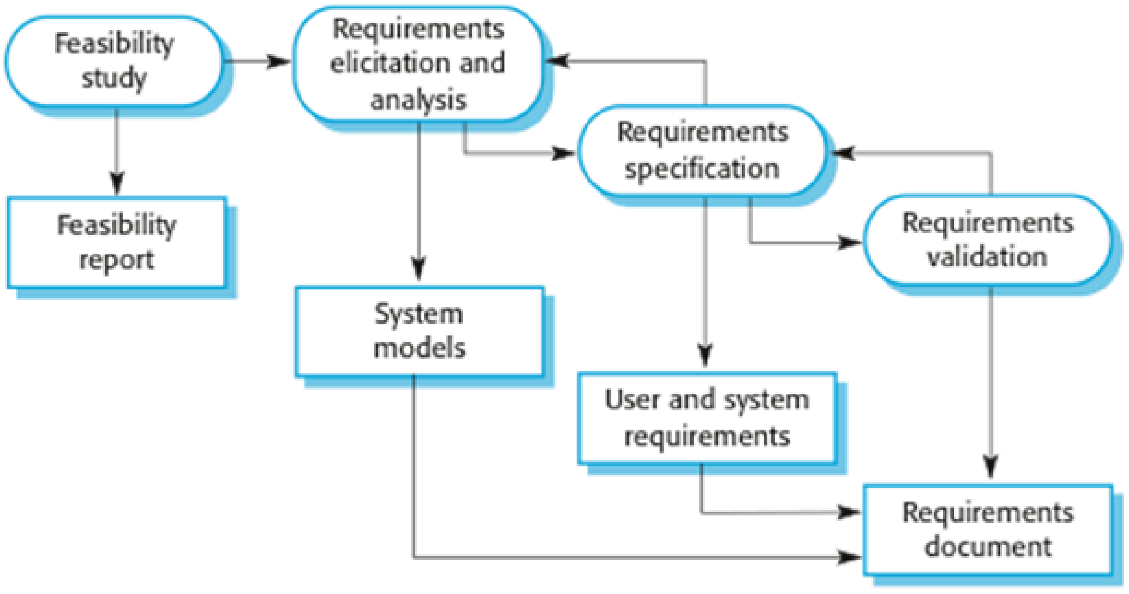
Plan-driven processes are those where all activities are pre-planned and progress is measured against this plan.

★ **Agile processes**

planning is incremental and it is easier to change the process to reflect changing customer requirements.

★ **Requirements engineering process**

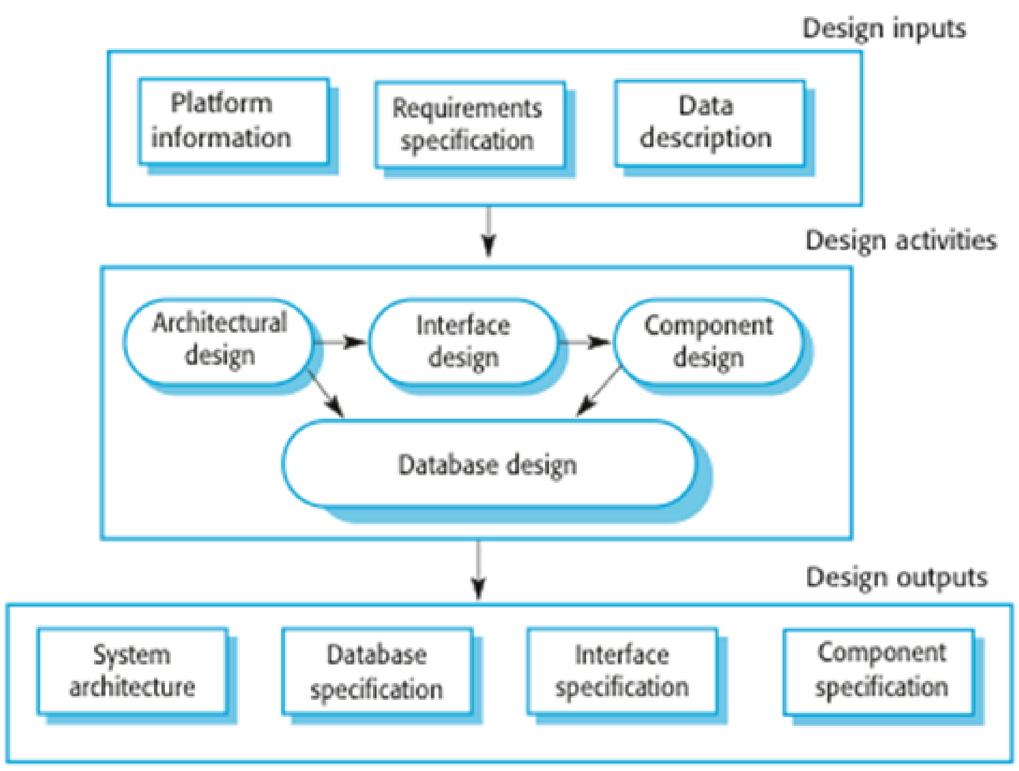
1. **Feasibility study**
   Is it technically and financially feasible to build the system?
2. **Requirements elicitation and analysis**
   What do the system stakeholders require or expect from the system?
3. **Requirements specification**
   Defining the requirements in detail
4. **Requirements validation**
   Checking the validity of the requirements



★ **Software design**
Software design is a mechanism to transform user requirements (SRS) into some suitable form.

★ **Implementation**
Translate this structure into an executable program.

★ **A general model of the design process**



★ **The software design process can be divided into the following three levels or phases of design.**

1. Interface Design
2. Architectural Design
3. Detailed Design

★ **Software validation**

Validation ensures that a system meets its specifications and fulfills the requirements of the customer.

★ **What is Software Testing ?**
Testing is the process of executing a program with the intent of finding errors.

★ **Testing stages**

1. **Development or component testing**
   Individual components are tested independently.
   Components may be functions or objects or coherent groupings of these entities.

2. **System testing**
   Testing of the system as a whole. Testing of emergent properties is particularly important.

3. **Acceptance testing**
   Testing with customer data to check that the system meets the customer's needs.

★ **Software process models**

1. **The waterfall model**
   Plan-driven model. Separate and distinct phases of specification and development.

2. **Incremental development (Evolutionary)**
   Specification, development and validation are interleaved. May be plan-driven or agile.

3. **Reuse-oriented software engineering**
   The system is assembled from existing components. May be plan-driven or agile

1. The waterfall model

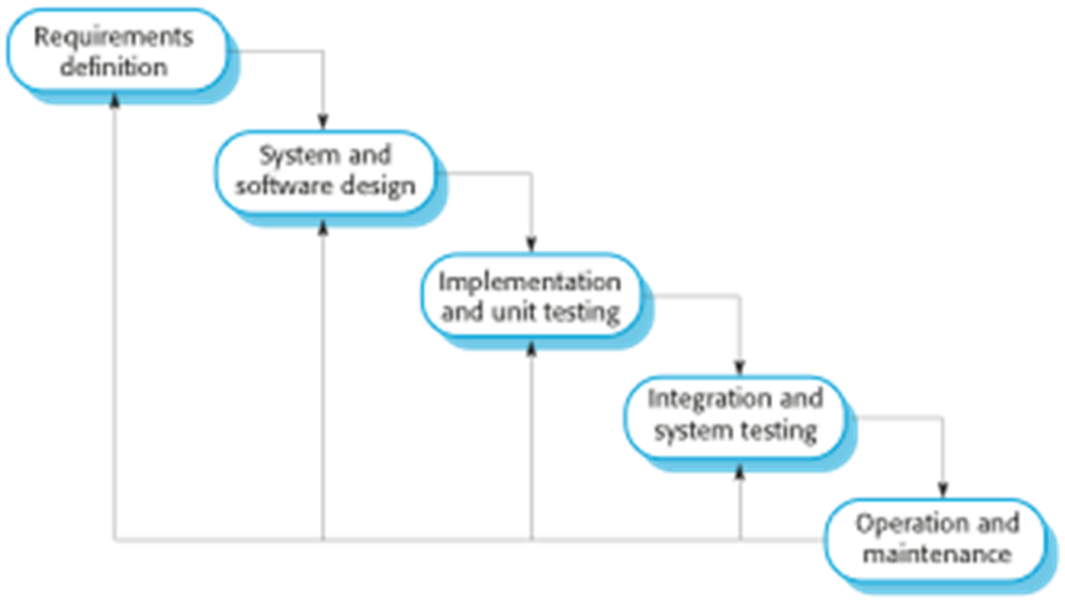
The Waterfall Model is a linear application development model that uses rigid phases,When one phase ends, the next begins.

There are separate identified phases in the waterfall model

1. Requirements analysis and definition
2. System and software design
3. Implementation and unit testing
4. Integration and system testing
5. Operation and maintenance

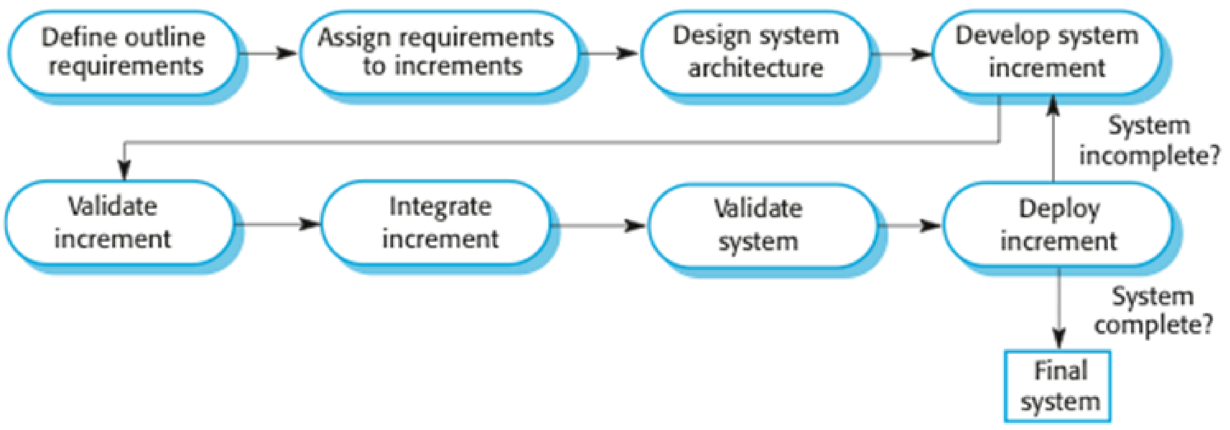
The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway.

Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

2. Incremental development (Evolutionary)

Incremental development is a method of building software products in which a system is built piece-by-piece.



Incremental delivery advantages

1. Delivers customer value early with each increment.
2. Early increments serve as prototypes for requirements.
3. Reduces overall project failure risk.
4. Ensures most testing for high-priority services.

Incremental delivery problems

1. Most systems require a set of basic facilities that are used by different parts of the system.
2. The essence of iterative processes is that the specification is developed in conjunction with the software.

Incremental Development Types

1. **Exploratory Development**
   Spiral Model

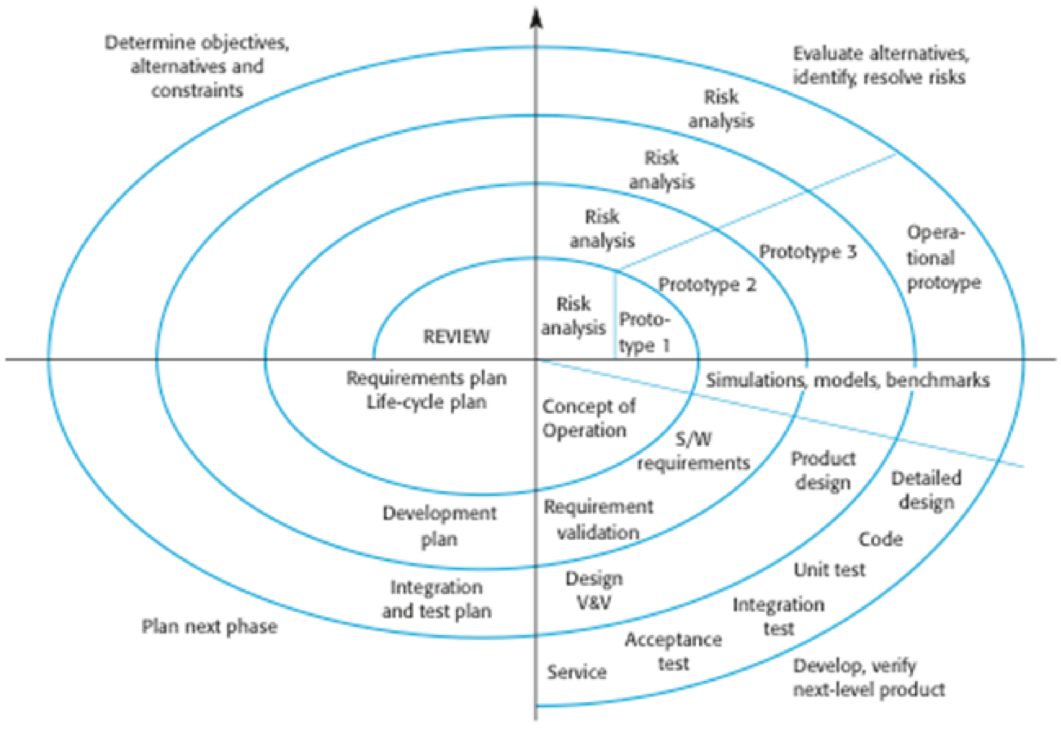
2. **Throwaway Prototyping**
   Rapid Prototyping

   1. **Boehm's spiral model**

      The Spiral Model that provides a systematic and iterative approach to software development.

      Process is represented as a spiral rather than as a sequence of activities with backtracking.

      No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.



Spiral model sectors

1. **Objective setting**
   Specific objectives for the phase are identified.
2. **Risk assessment and reduction**
   Risks are assessed and activities put in place to reduce the key risks.
3. **Development and validation**
   A development model for the system is chosen which can be any of the generic models.
4. **Planning**
   The project is reviewed and the next phase of the spiral is planned.

2. Software prototyping

A prototype is an initial version of a system used to demonstrate concepts and try out design options.

A prototype can be used in:
1. The requirements engineering process to help with requirements elicitation and validation.
2. In design processes to explore options and develop a UI design.
3. In the testing process to run back-to-back tests.

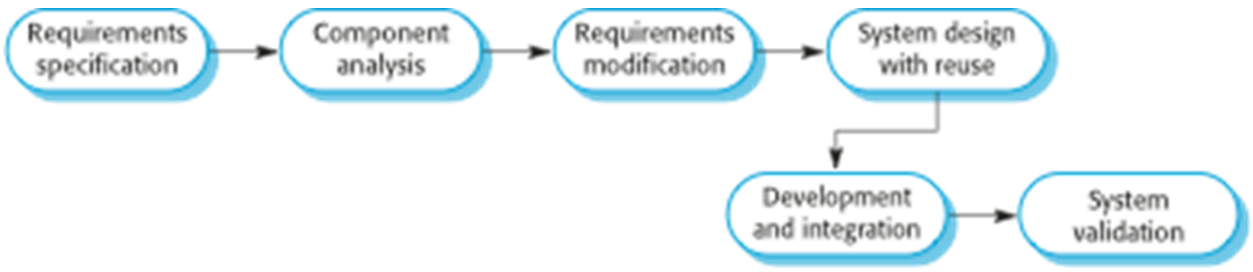
Benefits of prototyping

1. Improved system usability.
2. Improved design quality.
3. Improved maintainability.
4. Reduced development effort.
5. A closer match to users' real needs.

3. Reuse-oriented software engineering

Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.

1. Process stages
2. Component analysis
3. Requirements modification
4. System design with reuse
5. Development and integration.


Requirements specification
Component analysis
Requirements modification
System design with reuse
Development and integration
System validation

★ **Types of software component**

➔ Web services that are developed according to service standards and which are available for remote invocation.

➔ Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.

➔ Stand-alone software systems (COTS) that are configured for use in a particular environment

★ **Requirements engineering**

Requirements engineering is the process of developing a software specification.

★ **Agile Software Development**

Agile software development is an iterative and incremental approach to software development.

Agile software development is an approach to software development that emphasizes iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.

Key Principles of Agile

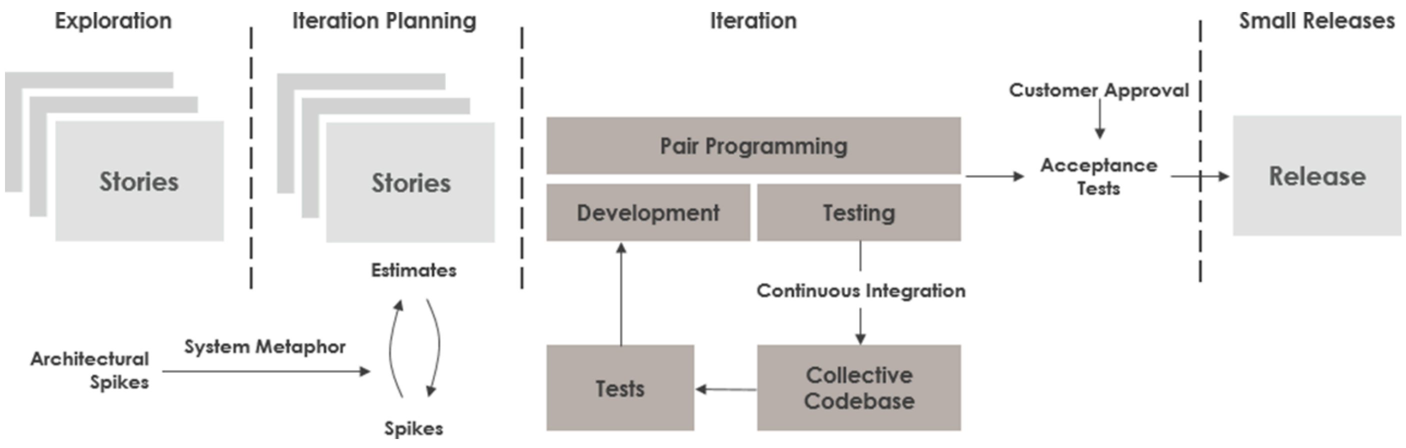
1. Iterative Development
2. Collaborative Approach
3. Adaptive Planning
4. Continuous Feedback
5. Self-organizing Teams
6. Customer Collaboration

Agile Frameworks

1. Scrum
2. Kanban
3. Extreme Programming (XP)
4. Lean Software Development.

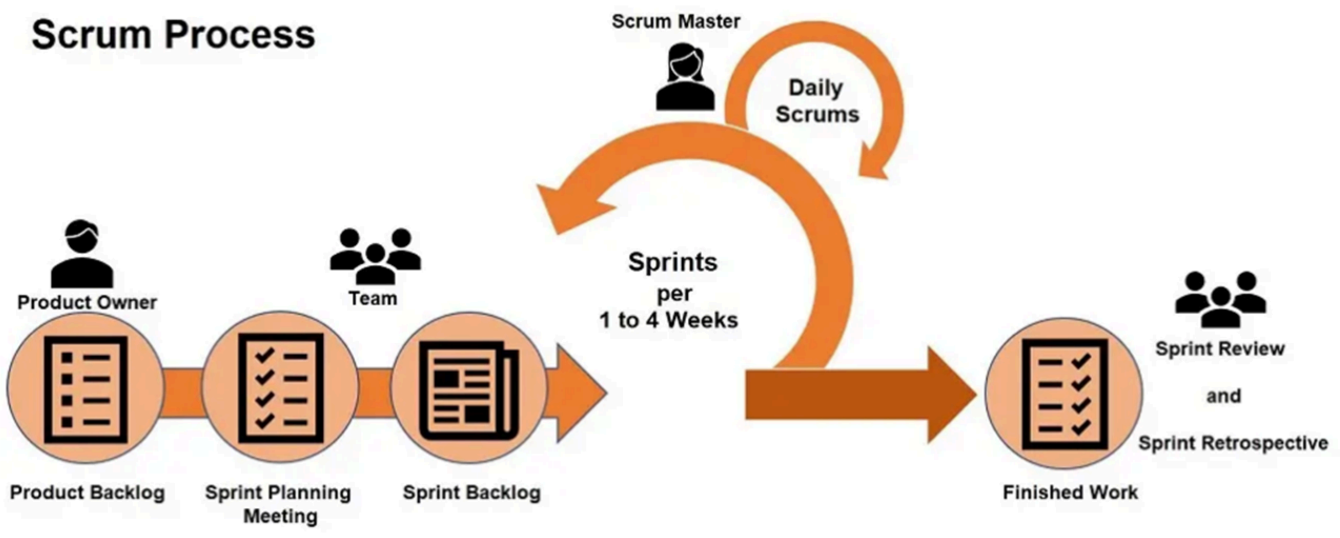
1. Extreme Programming

Extreme Programming (XP) is a software development methodology that focuses on delivering high-quality software quickly and efficiently.



2. Scrum

Scrum is an agile framework for managing and organizing work on complex projects, primarily used in software development but applicable to various other fields as well.



Scrum defines three primary roles

1. Product Owner
2. Scrum Master
3. Development Team

Artifacts

1. Product Backlog
2. Sprint Backlog
3. Increment

Events

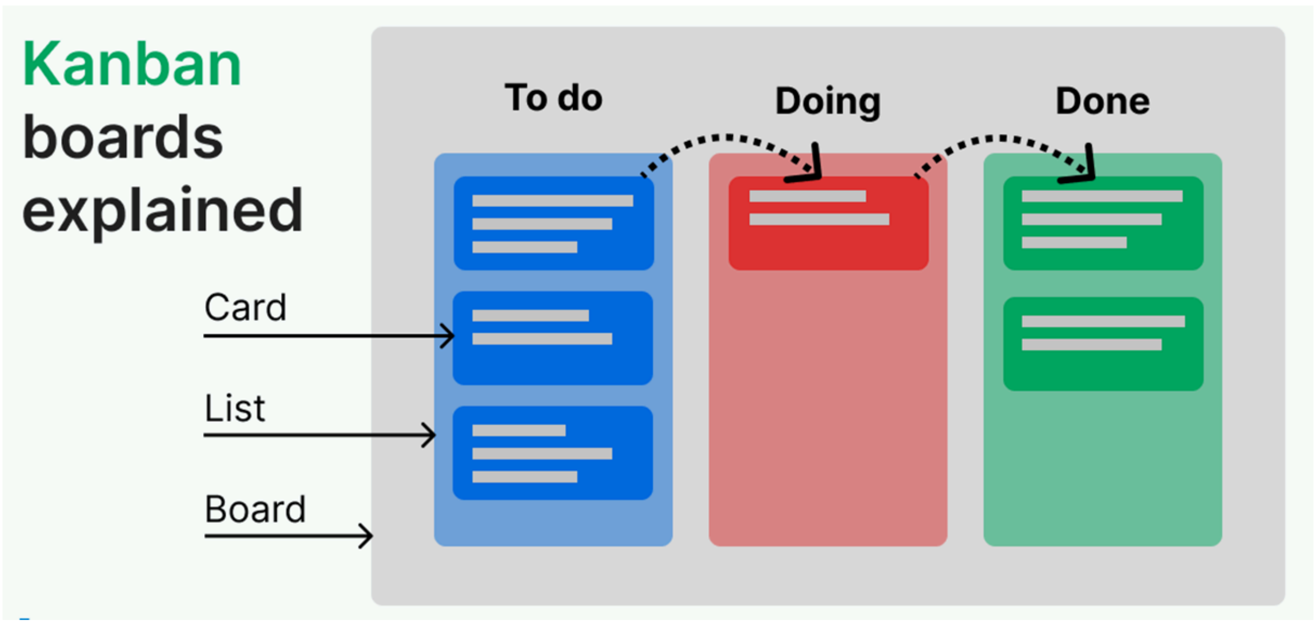
1. Sprint
2. Sprint Planning
3. Daily Scrum
4. Sprint Review

3. Kanban

Kanban is a method for managing workflow, originally developed by Toyota in the 1940s as part of its manufacturing process.

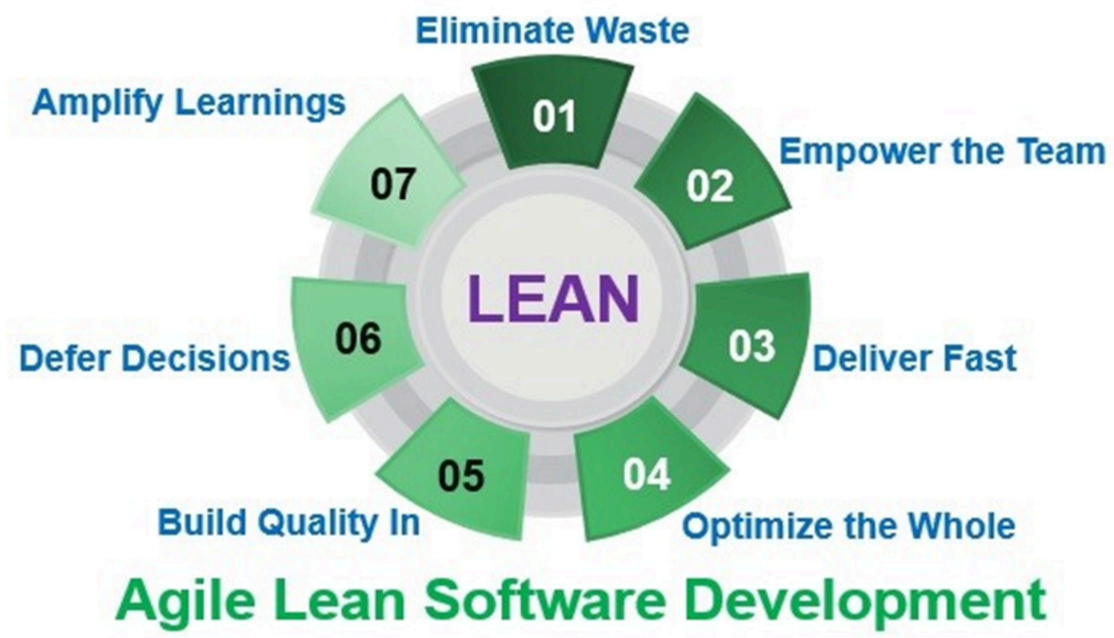
Key Elements of Kanban

1. Visualization
2. Limiting Work in Progress (WIP)
3. Flow
4. Pull-Based System
5. Continuous Improvement
6. Feedback Loops



4. Lean Software Development

Lean software development is a concept that emphasizes optimizing efficiency and minimizing waste in the software development 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.

★ **Types of Software Requirements**

Software Requirements are mainly classified into three types:

1. Functional requirements
2. Non-functional requirements
3. Domain requirements

1. Functional requirements

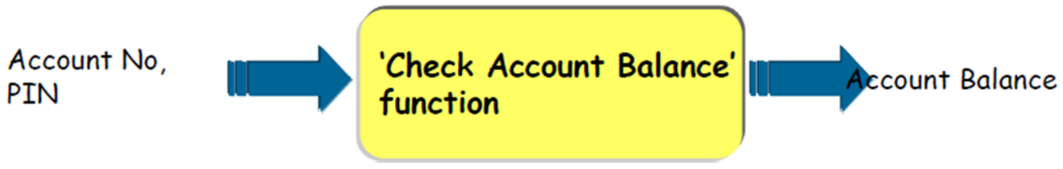
Specify what the information system must do

Example : Library system,

Functional - borrowing books, returning process
Non functional - security level, interfaces

Example: 2 – Find Books
Input: the name of an author.
Output: details of any books by the author ,title, publisher, ISBN, etc.
Processing: search library catalog for books by the specified author.

Example 3 - ATM System



Function Name: Check Account Balance

Input: Account No, PIN

Process: Check the balance of the relevant account number and display it.

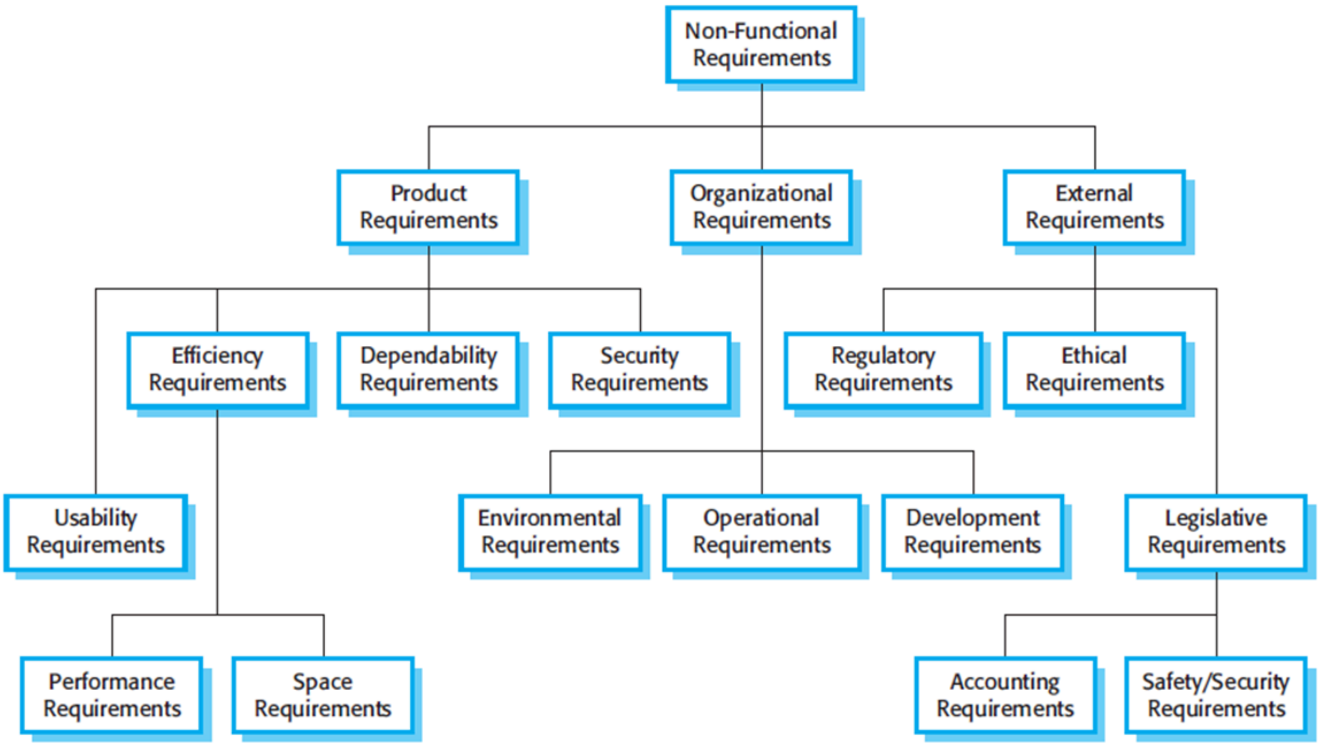
Output: Display Account Balance

2. Non-functional requirements

Specify a property / quality the system must have

These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.

Types of non functional requirement



Product requirements

Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.

Organizational requirements

Requirements which are a consequence of organizational 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.

3. **Domain requirements**

The system's operational domain imposes requirements on the system.

For example, a train control system has to take into account the braking characteristics in different weather conditions.

If domain requirements are not satisfied, the system may be unworkable.

★ **The software Requirements document (SRS)**

The software 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.

- ➔ Introduction:
  - ◆ Background, system environment
- ➔ Functional Requirements:
  - ◆ Numbered list of functions with inputs & outputs.
- ➔ Non-Functional Requirements
  - ◆ System characteristics
  - ◆ External interfaces(e.g. user interface)
- ➔ Constraints.
- ➔ Verification(Acceptance) Criteria.

★ **Requirements engineering processes**

There are a number of generic activities common to all processes

1. Requirements elicitation
2. Requirements analysis
3. Requirements validation
4. Requirements management.

★ **Stakeholders**

Stakeholders range from end-users of a system through managers to external stakeholders

★ **Scenarios**

Scenarios are real-life examples of how a system can be used.

They should include

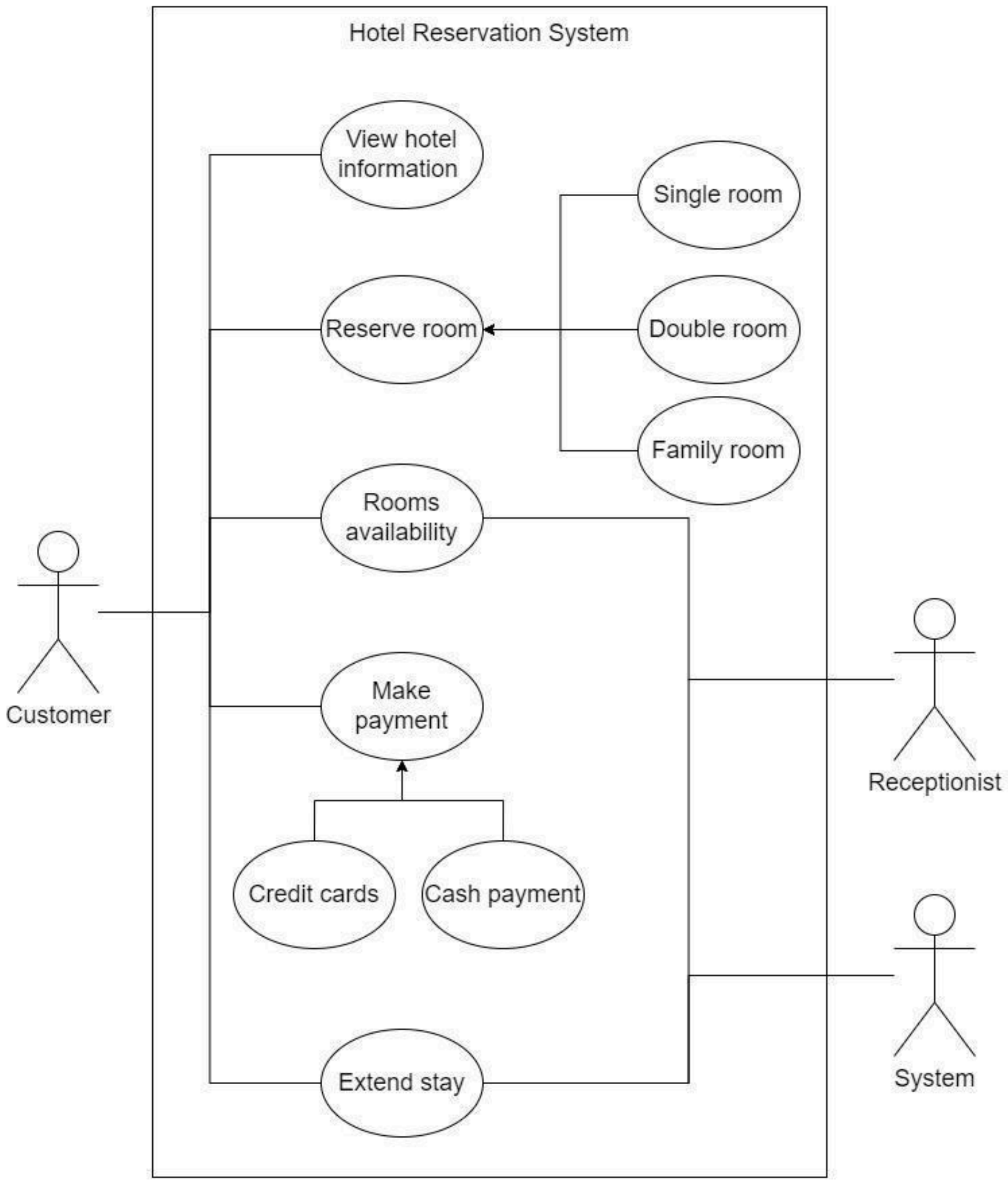
1. A description of the starting situation;
2. A description of the normal flow of events;
3. A description of what can go wrong;
4. Information about other concurrent activities;
5. A description of the state when the scenario finishes.

★ Use case diagram

It consists of use cases, actors and their relationships. Use case diagrams are used at a high level design to capture the requirements of a system. So it represents the system functionalities and their flow.

Question 1 -
Draw a class diagram for the following scenario in a hotel The following text describes the proposed Hotel Reservation System.

A customer can view the hotel information and reserve a room online or visit the place and reserve rooms. Rooms can be Single rooms, double rooms, or family rooms.If they are online customers, the system will check the room availability and notify the customer availability of rooms. If not, the receptionist does the task.When the customer makes the payment, the room is allocated for them. Online customers should pay via credit cards and others are allowed to pay by credit cards or cash payment.When the customer checks in the system is updated. The customer can check out from the hotel at any time or extend after informing the receptionist. It will be accepted if rooms are available



★ **Requirements checking**

1. **Validity -** Does the system provide the functions which best support the customer's needs?
2. **Consistency -** Are there any requirements conflicts?
3. **Completeness** - Are all functions required by the customer included?
4. **Realism -** Can the requirements be implemented given available budget and technology
5. **Verifiability** - Can the requirements be checked?

★ **Requirements validation techniques**

1. Requirements reviews
   Systematic manual analysis of the requirements.
2. Prototyping
   Using an executable model of the system to check requirements. Covered in Chapter 2.
3. Test-case generation
   Developing tests for requirements to check testability.

★ **Requirements reviews**

➔ Regular reviews should be held while the requirements definition is being formulated.
➔ Both client and contractor staff should be involved in reviews.
➔ Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

★ **Review checks**

1. **Verifiability**
   Is the requirement realistically testable?
2. **Comprehensibility**
   Is the requirement properly understood?
3. **Traceability**
   Is the origin of the requirement clearly stated?
4. **Adaptability**
   Can the requirement be changed without a large impact on other requirements?

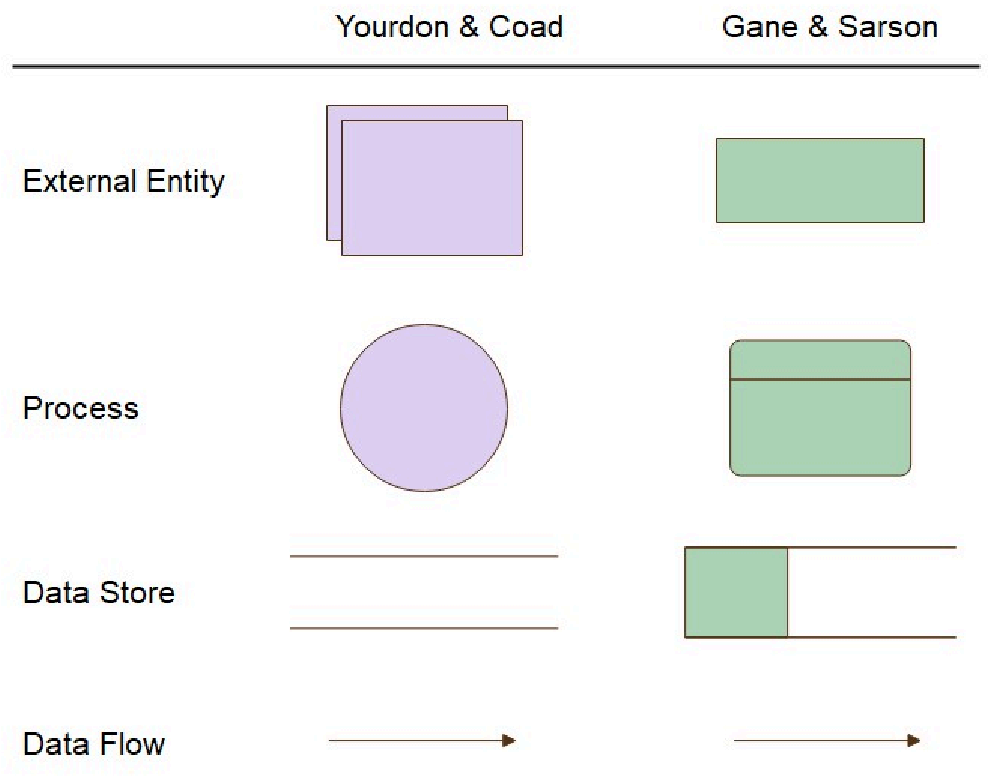
★ **Requirements management**

Requirements management is the process of managing changing requirements during the requirements engineering process and system development.

★ **Data Flow Diagram**

Data flow diagram is a graphical representation of flow of data in an information system.

DFD Components



Levels of DFD

1. Context Diagram
2. DFD Level 0
3. DFD Level 1

Q1) Answer the following questions based on the below scenario.

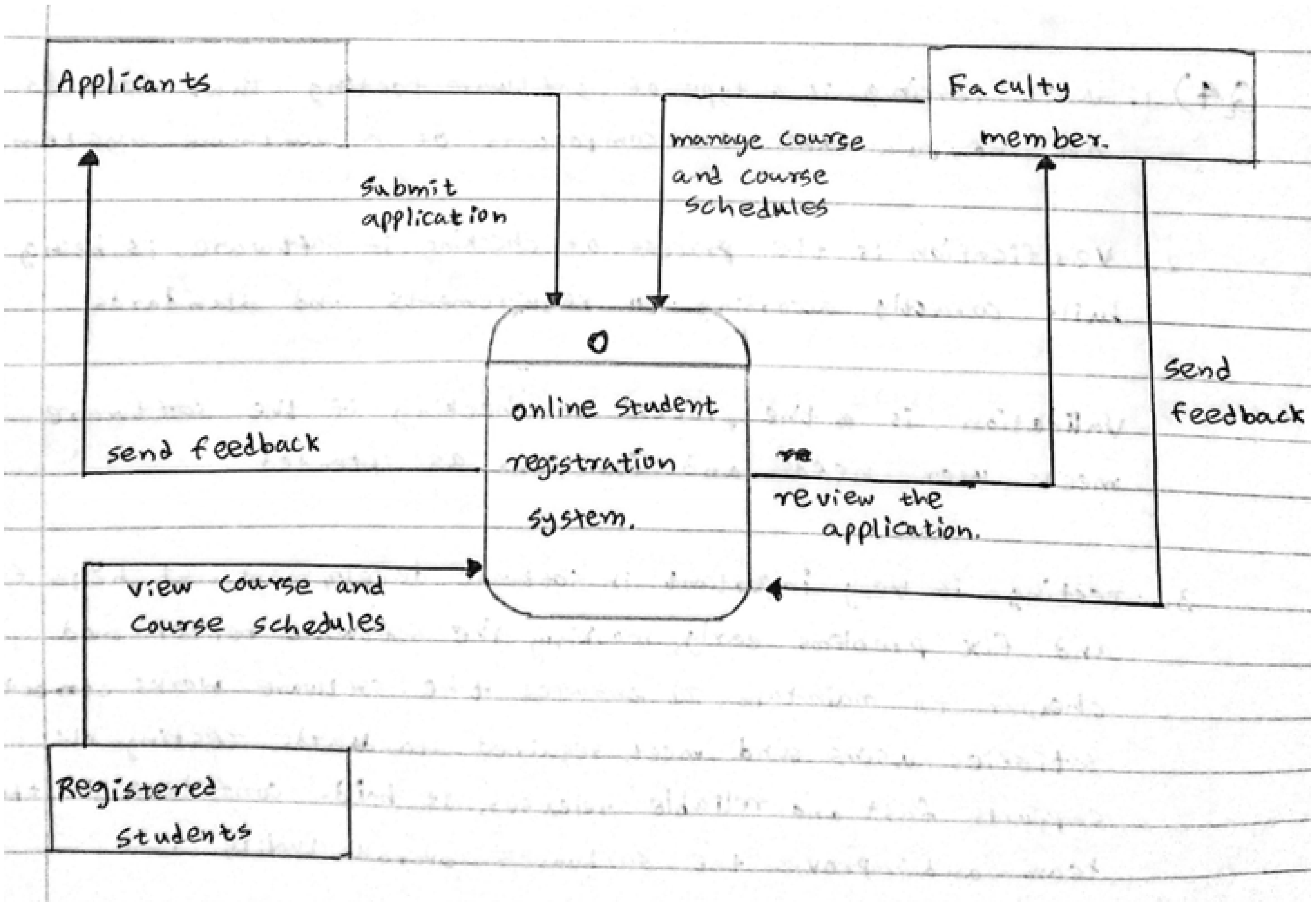
A university wants to develop an online Student registration system to automate and streamline the student enrollment process.

The system will allow applicants to register for a course by submitting an application. Faculty members will review the application and send feedback to the applicant. Faculty members will also have access to the system to manage course and course schedules. Registered students can view course and course schedules. Scheduled files store information about the schedules where as the student information file stores student details and course details in the course file.

a. Identify the external entities of Student Registration System
b. Draw the Context Diagram
c. Identify the sub process of the system
d. Identify the data stores of the system
e. Draw the Level-0 DFD for the above scenario

Answer

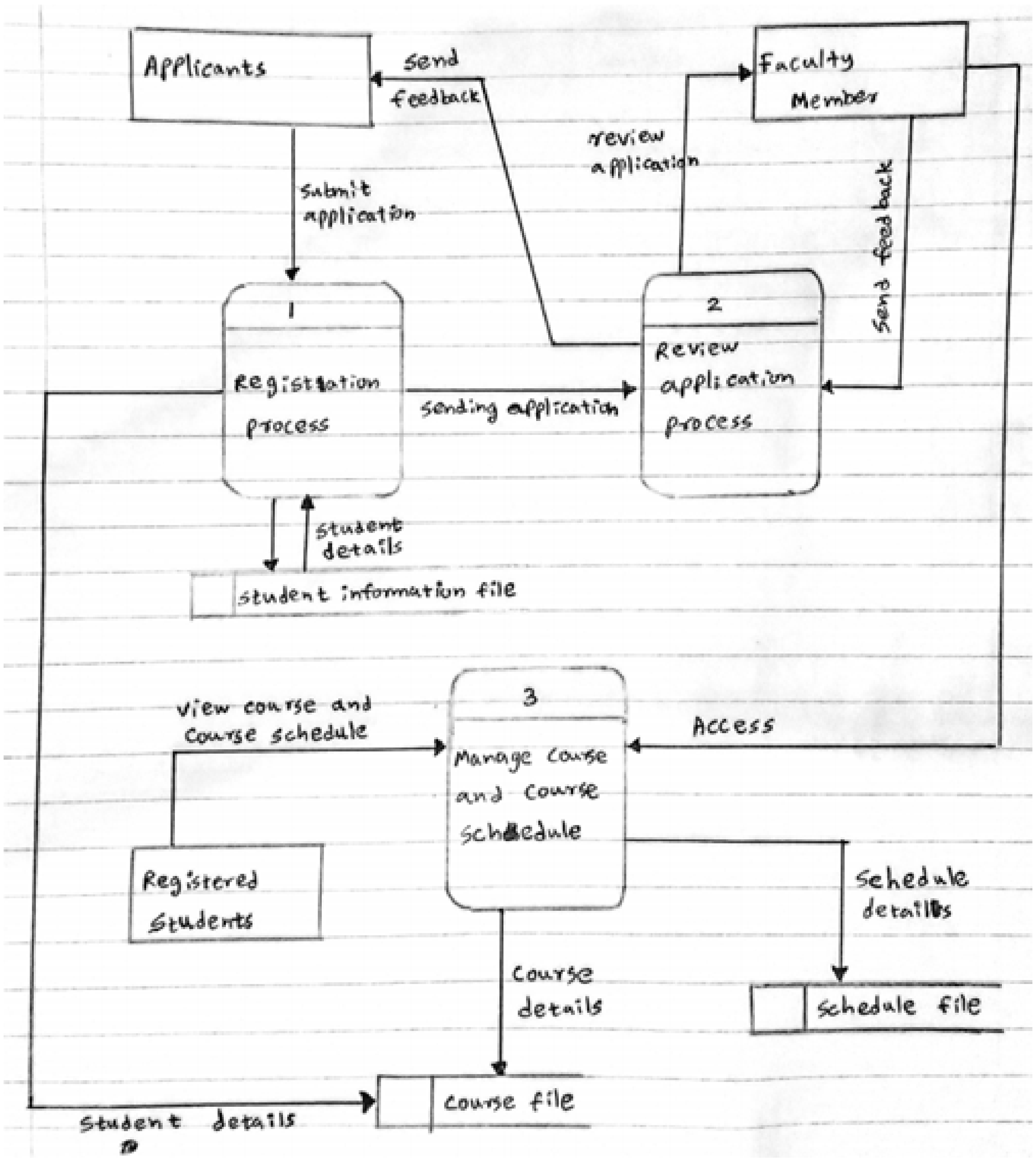
a. applicant, faculty member, registered student
b.


Applicants
Submit application
manage course and course schedules
Faculty member.
0
online Student registration system.
send feedback
~~re~~ review the application.
Send feedback
view course and Course schedules
Registered students

c. Registration process, Review application process, manage course and course schedule

d. Student information file, course file, schedule file

e. DFD Level 0

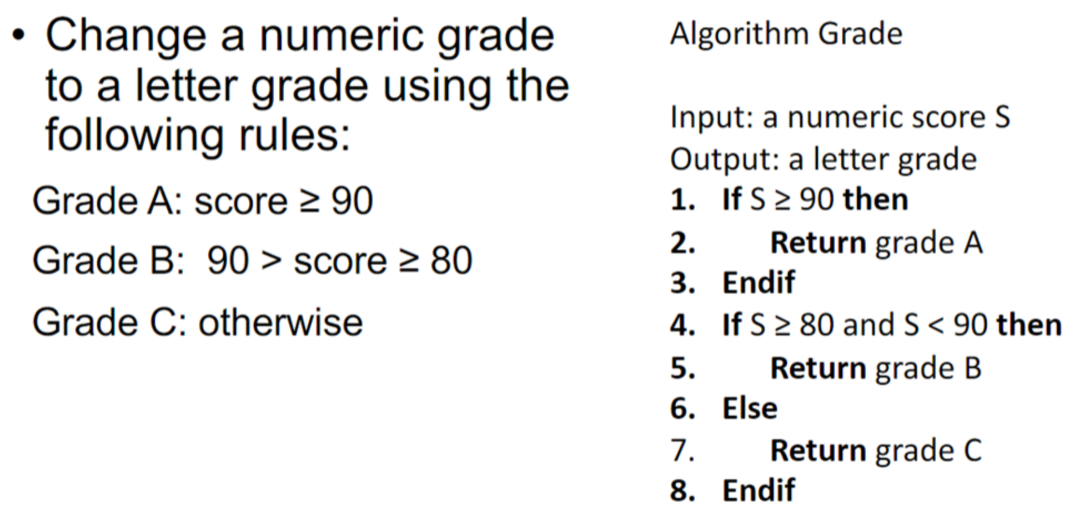


★ **Structure chart**

Structure chart is a chart derived from a Data Flow Diagram. It represents the system in more detail than DFD.

★ **Pseudo-Code**

Pseudo code is written more closely to the programming language.
It may be considered an augmented programming language, full of comments and descriptions.



- Change a numeric grade to a letter grade using the following rules:

  Grade A: score ≥ 90

  Grade B: 90 > score ≥ 80

  Grade C: otherwise

Algorithm Grade

Input: a numeric score S
Output: a letter grade

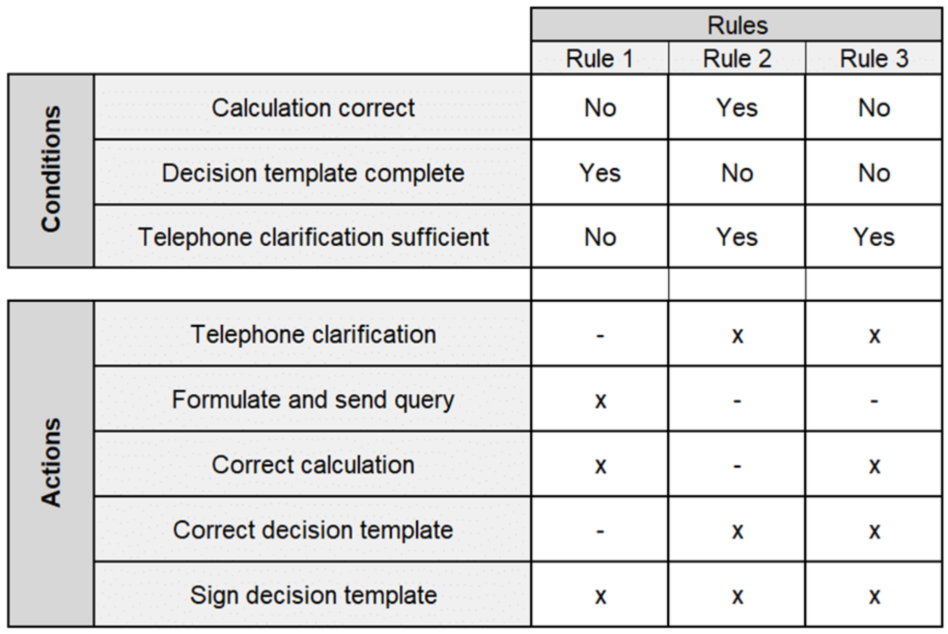
```
1. If S ≥ 90 then
2.       Return grade A
3. Endif
4. If S ≥ 80 and S < 90 then
5.       Return grade B
6. Else
7.       Return grade C
8. Endif
```

★ **Decision Tables**

A Decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.

It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

Example -



		Rules		
		Rule 1	Rule 2	Rule 3
Conditions	Calculation correct	No	Yes	No
	Decision template complete	Yes	No	No
	Telephone clarification sufficient	No	Yes	Yes
Actions	Telephone clarification	-	x	x
	Formulate and send query	x	-	-
	Correct calculation	x	-	x
	Correct decision template	-	x	x
	Sign decision template	x	x	x

★ **Entity-Relationship Model**

Entity-Relationship model is a type of database model based on the notion of real world entities and relationships among them.

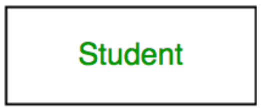
Components of E-R Model

1. Entity
2. Attribute
3. Key
4. Relationship
5. Structural constraints on relationship

1. **Entity**

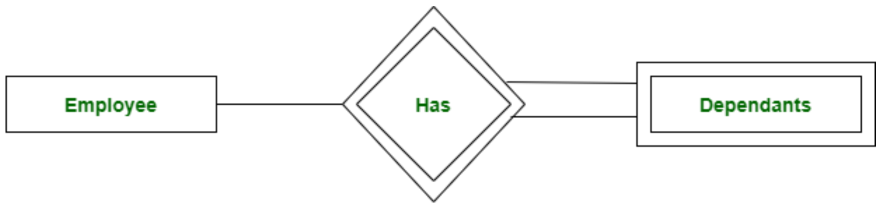
   An object or concept Thing in real world with independent existence

   "...anything (people, places, objects, events, etc.) about which we store information (e.g. supplier, machine tool, employee, utility pole, airline seat, etc.)."



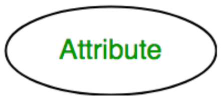
   Types

   - Strong entity (parent, owner, dominant) - Strong Entity does not depend on other entity.
   - Weak entity (child, dependent, or subordinate) - Weak entity is a depend on another entity



2. **Attribute**

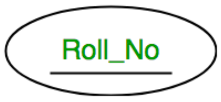
   attribute is Particular properties that describe entity



   Types
   - Key attribute: Uniqueness property must hold for every entity set of the entity type
   - Composite
   - Multi-valued
   - Derived
   - Stored
   - derived attributes

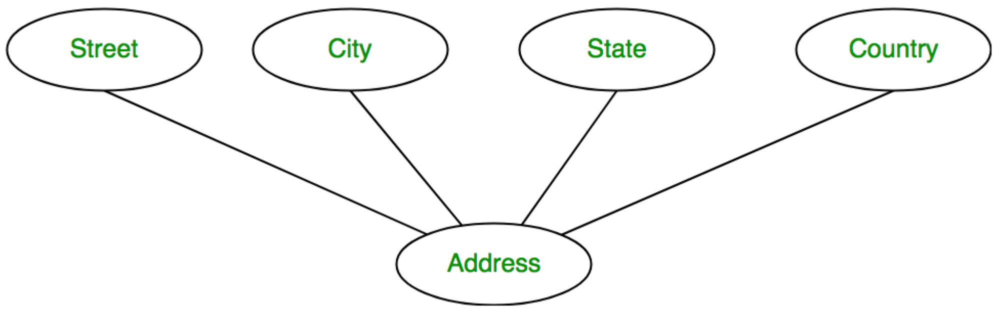
   1. Key attribute

      The attribute which uniquely identifies each entity in the entity set is called the key attribute.



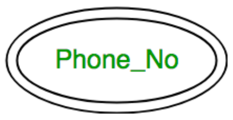
   2. Composite Attributes

      Can be subdivided into smaller subparts



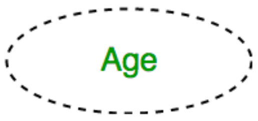
3. Multivalued Attribute

An attribute consisting of more than one value for a given entity.



4. Derived Attributes

An attribute that can be derived from other attributes of the entity type is known as a derived attribute.
e.g.; Age (can be derived from DOB).



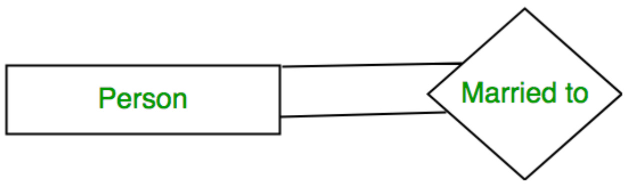
3. **Degree of a relationship**

Number of participating entities

Types
1. Unary (recursive relationship)
2. Binary
3. Ternary
4. Quaternary / N - ary

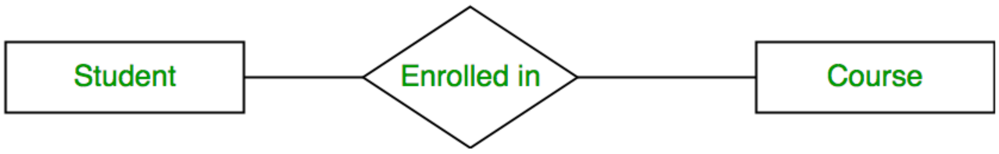
1. Unary (recursive relationship)

When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.



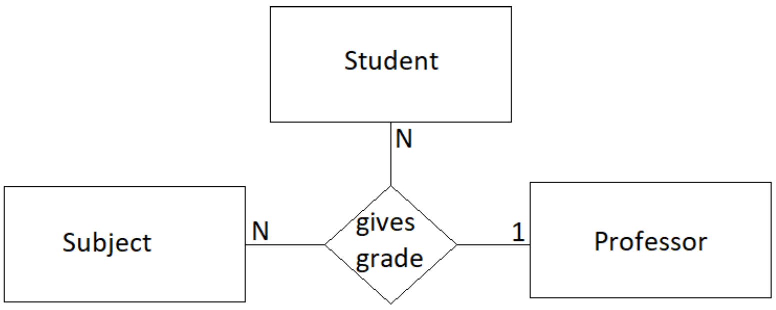
2. Binary relationship

When there are TWO entities participating in a relationship, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



3. Ternary relationship

When there are three entities participating in a relationship, the relationship is called a Ternary relationship.



4. Quaternary / N - ary relationship

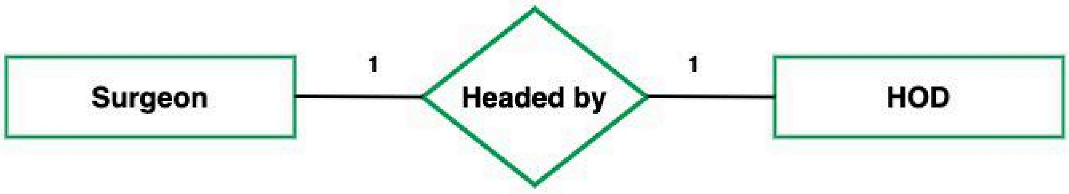
When there are n entities set participating in a relation, the relationship is called an n-ary relationship.

4. Structural constraints on relationships./ Cardinality

1. One-to-One

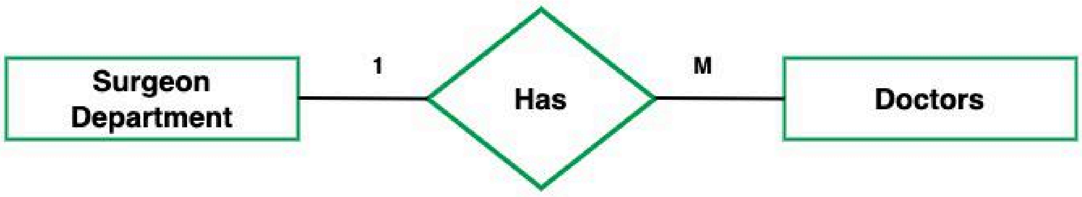
In a particular hospital, the surgeon department has one head of department. They both serve one-to-one relationships

Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one.



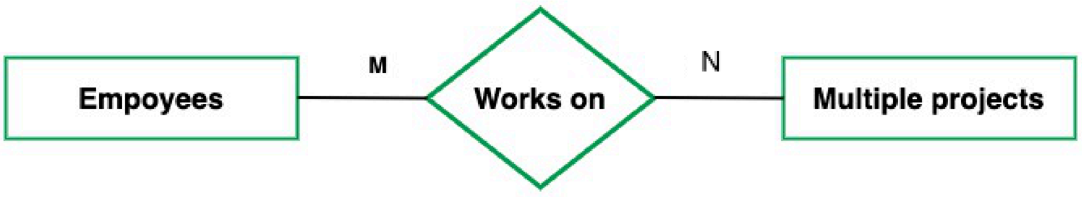
2. One-to-Many

In a particular hospital, the surgeon department has multiple doctors. They serve one-to-many relationships

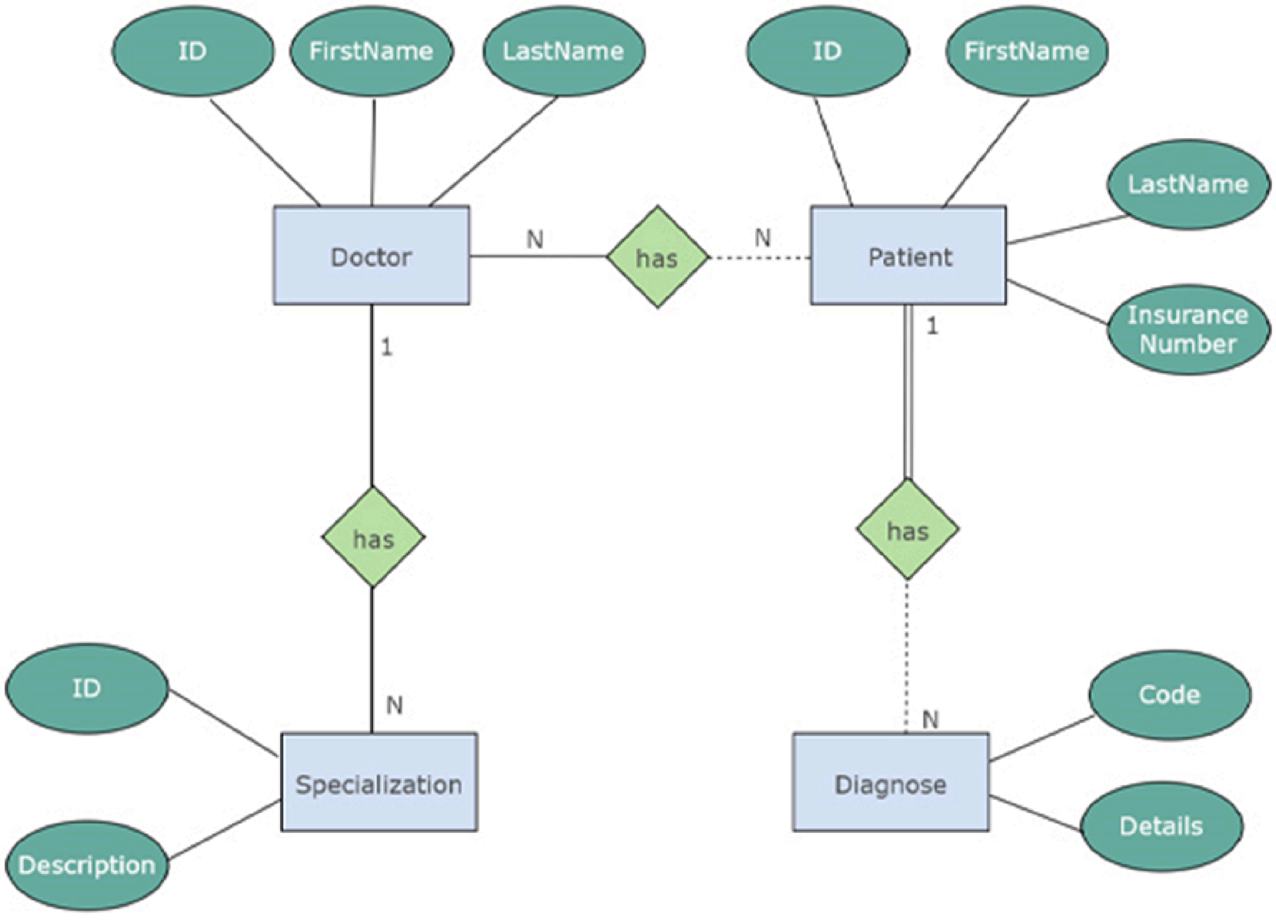


3. Many-to-Many

In a particular company, multiple people work on multiple projects. They serve many-to-many relationships.



Example -



★ **Data Dictionary**

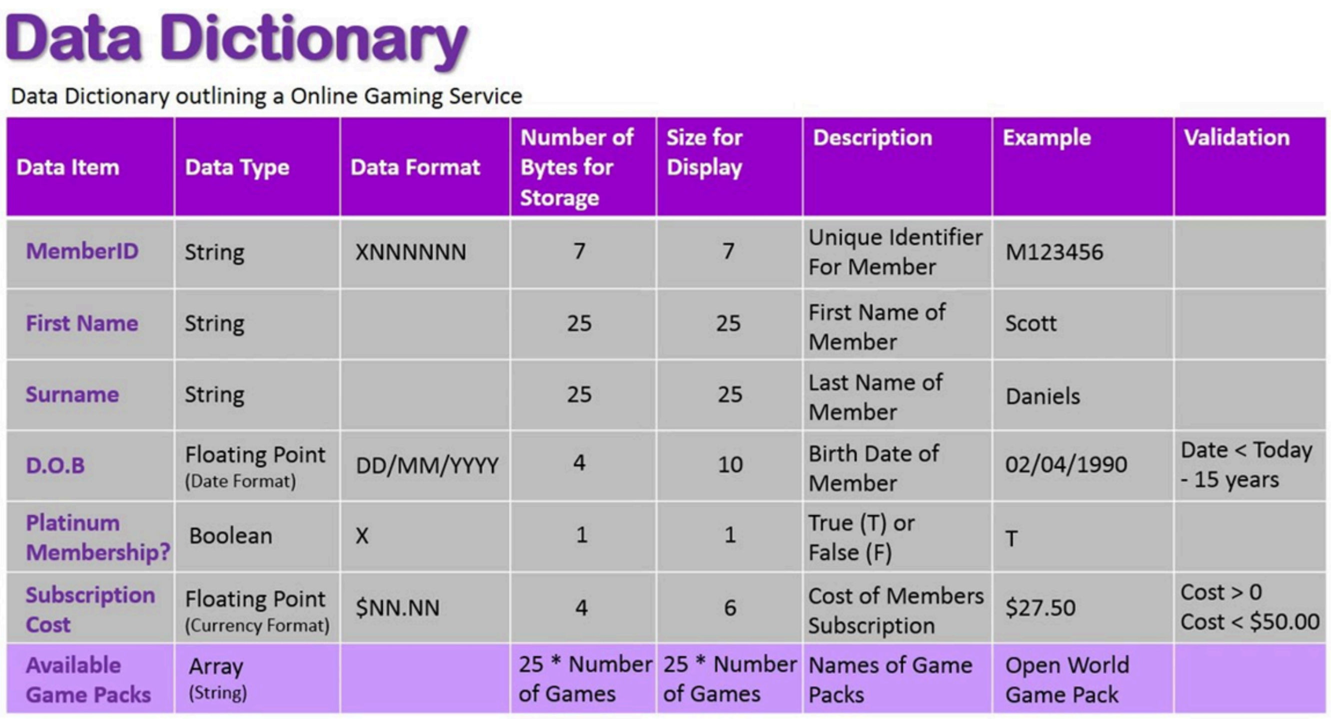
Data dictionary is the centralized collection of information about data.

It stores meaning and origin of data, its relationship with other data, data format for usage etc.

Elements of Data Dictionary

1. Data Flow
2. Data Structure
3. Data Elements
4. Data Stores
5. Data Processing

Example -



Data Dictionary

Data Dictionary outlining a Online Gaming Service

Data Item	Data Type	Data Format	Number of Bytes for Storage	Size for Display	Description	Example	Validation
MemberID	String	XNNNNNN	7	7	Unique Identifier For Member	M123456	
First Name	String		25	25	First Name of Member	Scott	
Surname	String		25	25	Last Name of Member	Daniels	
D.O.B	Floating Point (Date Format)	DD/MM/YYYY	4	10	Birth Date of Member	02/04/1990	Date < Today - 15 years
Platinum Membership?	Boolean	X	1	1	True (T) or False (F)	T	
Subscription Cost	Floating Point (Currency Format)	$NN.NN	4	6	Cost of Members Subscription	$27.50	Cost > 0 Cost < $50.00
Available Game Packs	Array (String)		25 * Number of Games	25 * Number of Games	Names of Game Packs	Open World Game Pack	

★ **What is Software Architecture ?**

Software architecture is the high-level structure of a software system, defining its components, relationships, and interactions.

Software architecture is the blueprint for building software.

Key Aspects of Software Architecture

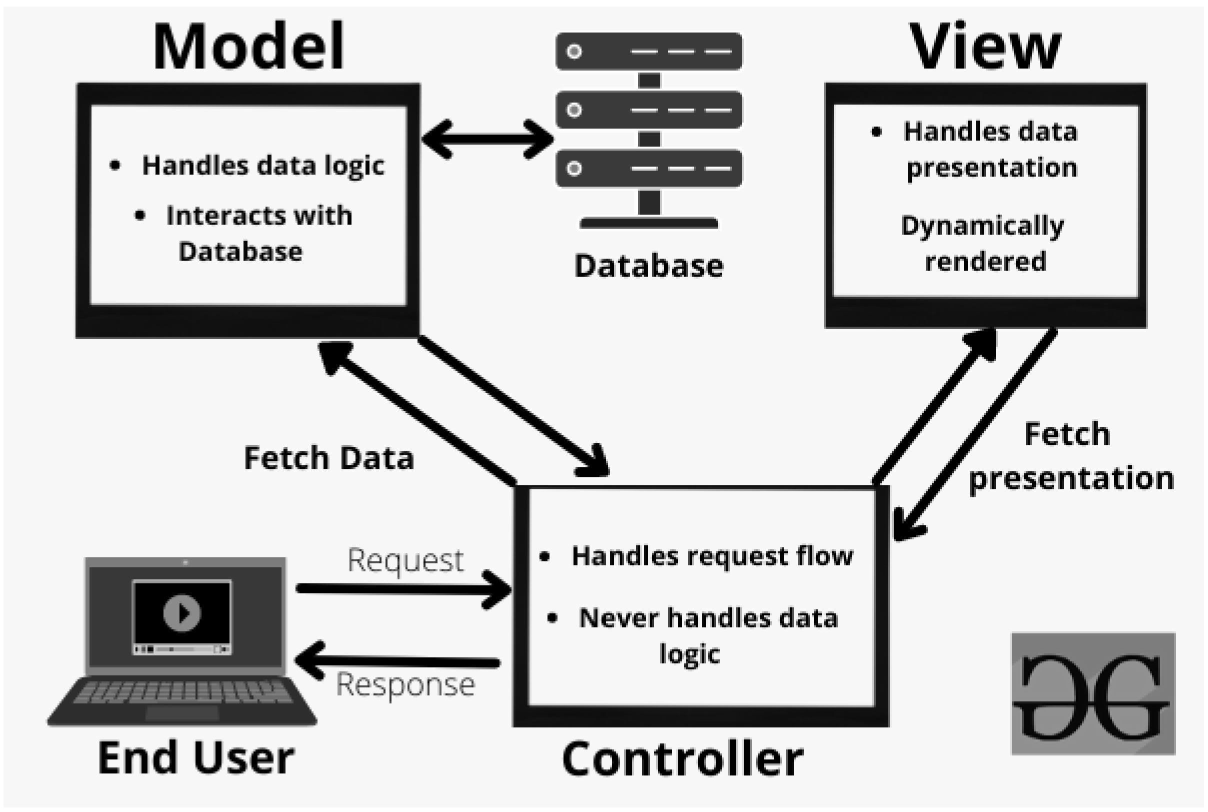
1. Components
2. Connectors
3. Architectural style and pattern

1. Architectural pattern

➔ **Model-View-Controller (MVC)**

This pattern is the basis of interaction management in many web-based systems.

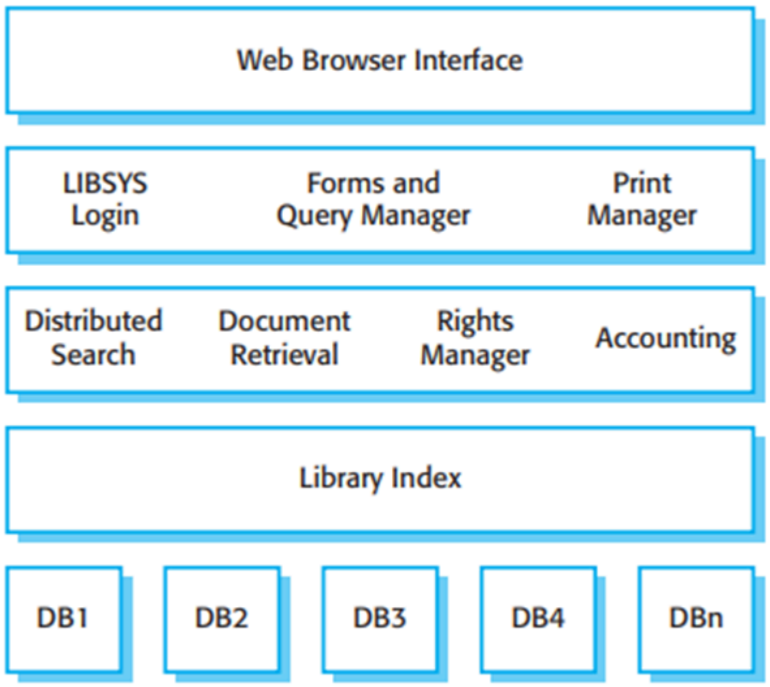
MVC (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic.



➔ **Layered architecture**

This layered approach supports the incremental development of systems. The architecture is also changeable and portable.

Example -



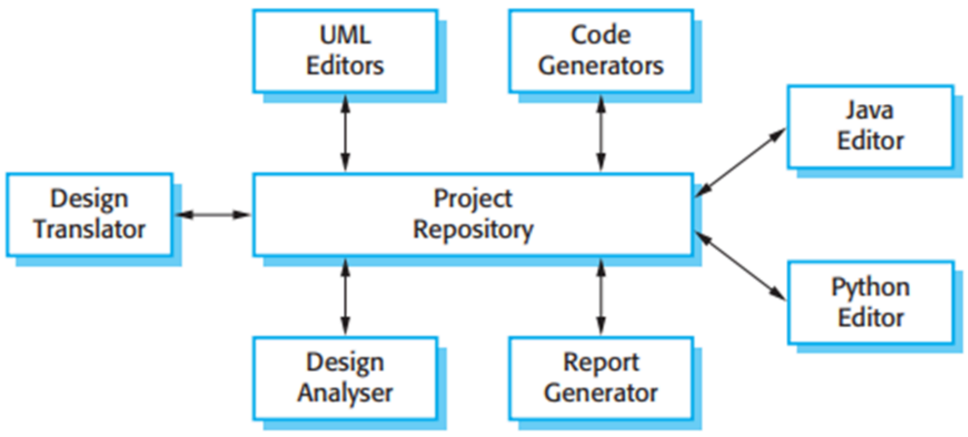
➔ Repository Architecture

All data in a system is managed in a central repository that is accessible to all system components.

Components do not interact directly, only through the repository.

For example, database management systems (DBMS) are based on the repository style,

Ex- A repository architecture for an IDE.

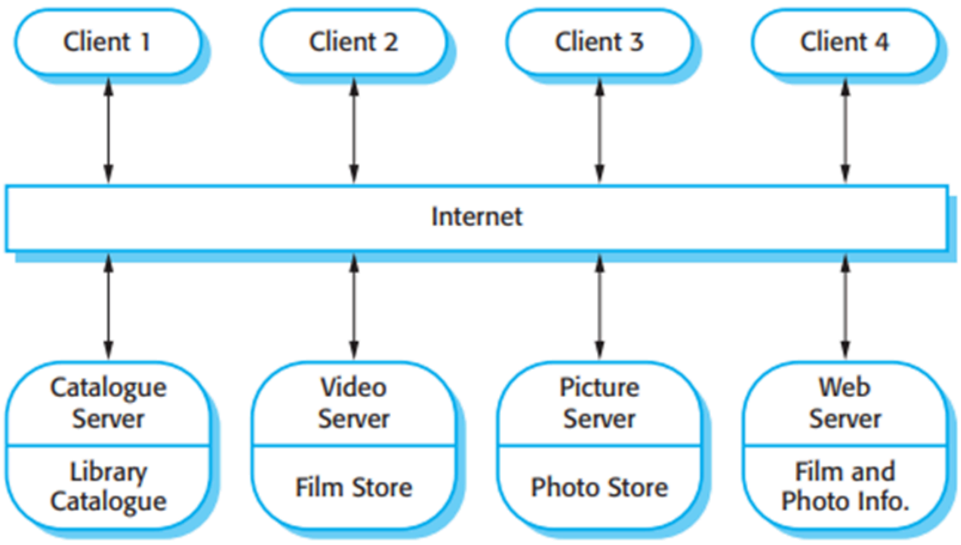


➔ Client Server Architecture

In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server.

Clients are users of these services and access servers to make use of them.

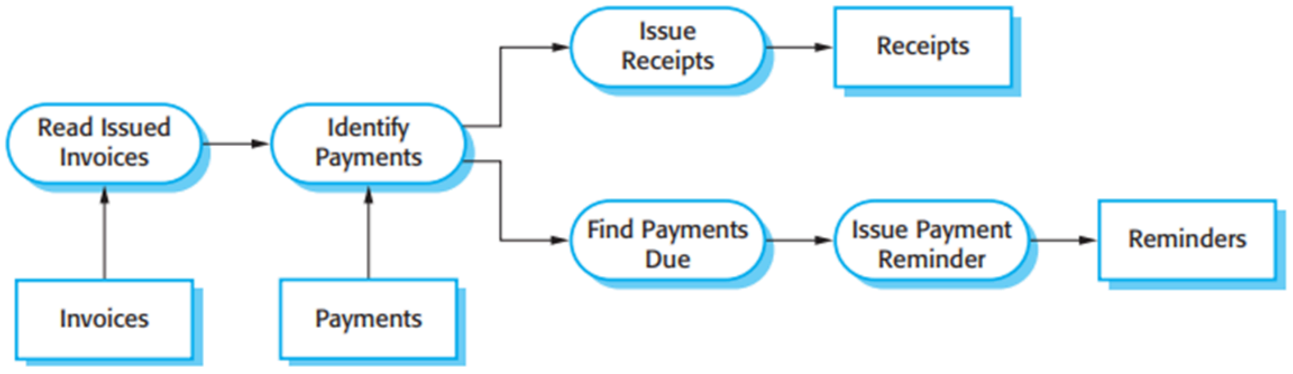
Example -



➔ **Pipe and Filter Architecture**

This is a model of the run-time organization of a system where functional transformations process their inputs and produce outputs.

Data flows from one to another and is transformed as it moves through the sequence.



➔ **Service-Oriented Architecture (SOA)**

Service-Oriented Architecture (SOA) is a stage in the evolution of application development and/or integration.

It defines a way to make software components reusable using the interfaces.

★ **What is Coding ?**

The coding phase involves converting the system design into high-level programming code and then performing unit tests on this code.

★ **Characteristics of a Programming Language**

1. Readability
2. Portability
3. Cost
4. Error checking
5. Generality
6. Familiar notation
7. Modularity

★ **Coding standards and guidelines**

1. Rules for limiting the use of global
2. Naming conventions for global variables, local variables, and constant identifiers
3. Contents of the headers preceding codes for different modules
4. Do not use a coding style that is too clever or too difficult to understand
5. The code should be well-documented
6. The length of any function should not exceed 10 source lines
7. Do not use an identifier for multiple purposes

★ Code Review

Code reviews are extremely cost-effective strategies for reduction in coding errors and to produce high quality code.

Normally, two types of reviews are carried out on the code of a module.

1. Code inspection
2. Code walk through

1. Code walk through

Code walk through is an informal code analysis technique.

The main objectives of the walk through are to discover the algorithmic and logical errors in the code.

A few members of the development team are given the code a few days before the walk through meeting to read and understand code.

2. Code inspection

The aim of code inspection is to discover some common types of errors caused due to oversight and improper programming.

Good software development companies collect statistics regarding different types of errors commonly committed by their engineers and identify the type of errors most frequently committed.

★ Classical Programming Errors

1. Use of uninitialized variables.
2. Jumps into loops.
3. Non terminating loops.
4. Incompatible assignments.
5. Array indices out of bounds.
6. Improper storage allocation and deallocation.
7. Use of incorrect logical operators or incorrect precedence among operators.

★ Software Implementation Challenges

1. Code-reuse
2. Version Management

   Every time a new software is issued to the customer, developers have to maintain version and configuration related documentation.

   This documentation needs to be highly accurate and available on time.

3. Target-Host

- ★ **The main advantages of adhering to a standard style of coding are as follows**

    1. It facilitates code of understanding.
    2. Promotes good programming practices.
    3. A coding standard gives uniform appearances to the code written by different engineers

- ★ **What is Software Testing ?**

    The aim of the testing process is to identify all defects existing in a software product.

    Software Testing is a method to assess the functionality of the software program. The process checks whether the actual software matches the expected requirements and ensures the software is bug-free.

- ★ **Aim of Testing**

    1. Errors
    2. Fault
    3. Failure

- ★ **Software Verification**

    Verification is confirming that the software meets business requirements and adheres to specified methodologies and standards.

- ★ **Software Validation**

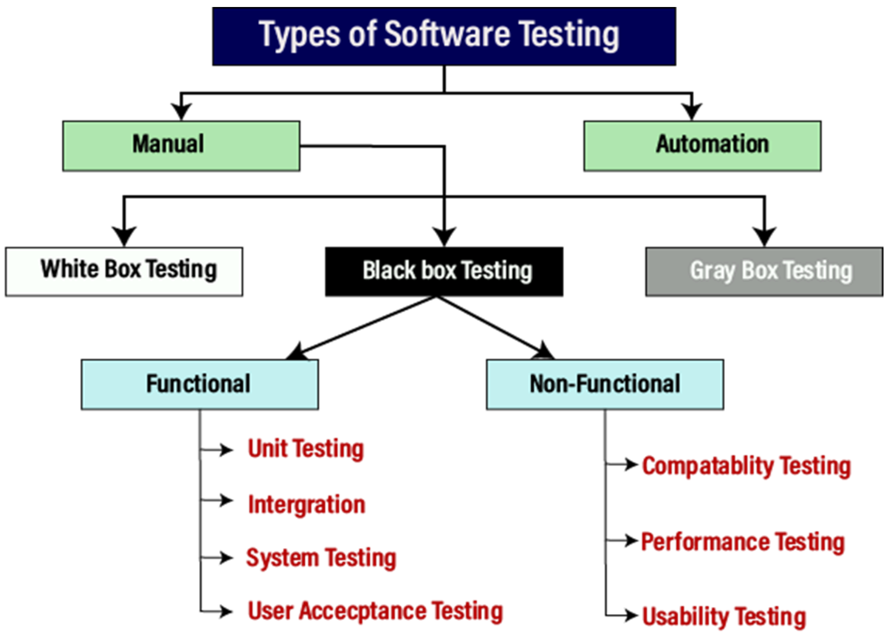
    Validation is the process of examining whether or not the software satisfies the user requirements.

- ★ **Testing Approaches**

    1. Functionality testing

        In the black-box testing approach, test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software.
        For this reason, black-box testing is known as functional testing.

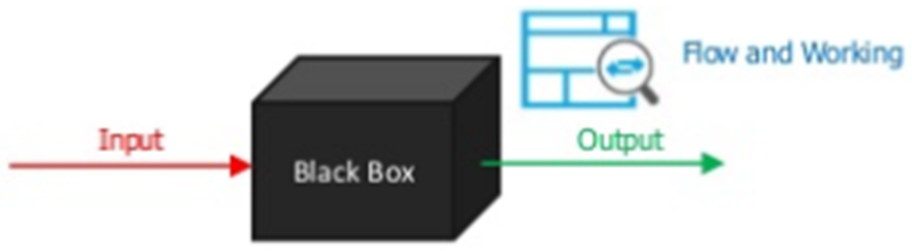
    2. Structural testing
        On the other hand, in the white-box testing approach, designing test cases requires thorough knowledge about the internal structure of software, and therefore the white-box testing is called structural testing.



★ **Black Box Testing**

Tester tests the application software by applying different inputs and comparing the output with expected results.
It is also known as Behavioural Testing.

Ex - Appium, Selenium, Microsoft Coded UI, Applitools, HP QTP.



Black-box Testing Techniques

1. Equivalence class partitioning
2. Boundary value analysis

1. **Equivalence Class Partitioning**

In this approach, the domain of input values to a program is partitioned into a set of equivalence classes.

Equivalence classes for a software can be designed by examining the input data and output data.

Q1) For a software that computes the square root of an input integer which can assume values in the range of 0 to 5000,

There are three equivalence classes: The set of negative integers, the set of integers in the range of 0 and 5000, and the integers larger than 5000.

Therefore, the test cases must include representatives for each of the three equivalence classes and a possible test set can be: {-5,500,6000}.

2. **Boundary Value Analysis**

A type of programming error frequently occurs at the boundaries of different equivalence classes of inputs.

For example, programmers may improperly use < instead of <=.

Boundary value analysis leads to selection of test cases at the boundaries of the different equivalence classes.

Q1) For a function that computes the square root of integer values in the range of 0 and 5000,

The test cases must include the following values: {0, -1, 5000, 5001}.

- ★ **White Box Testing**

White box testing is a verification technique where software engineers can use to examine.if the code works as expected or not.

Ex- Wireshark,PyUnit,Nunit,Nmap

White-box Testing Techniques

1. Statement Coverage
2. Decision Coverage
3. Branch Coverage
4. Condition Coverage
5. Path Coverage
6. Control flow testing
7. Data flow testing

- ★ **Manual Testing**

This testing is performed without taking help of automated testing tools.

The test cases are executed by the human tester.

- ★ **Automation Testing**

This testing is a testing procedure done with aid of automated testing tools.

★ Difference Between Manual Testing and Automated Testing

Manual Testing	Automated Testing
The test cases are executed by the human tester.	The test cases are executed by the software tools.
Manual testing is time-consuming.	Automation testing is faster than manual testing.
Manual testing is less reliable.	Automation testing is more reliable.
No need for programming knowledge.	Programming knowledge is a must.
Manual testing doesn't use frameworks.	Automation testing uses frameworks like Data Drive, Keyword, etc.

★ Software Testing Levels

1. Unit Testing
2. Integration Testing
3. System Testing

1. Unit Testing

Unit testing is a type of software testing that focuses on individual units or components of a software system.

Testing is performed under white-box testing approach.

Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

2. Integration Testing

The primary objective of integration testing is to test the module interfaces,

i.e. there are no errors in the parameter passing, when one module invokes another module.

There are four types of integration testing approaches.

1. Big bang approach
2. Bottom- up approach
3. Top-down approach
4. Mixed-approach

3. System Testing

System tests are designed to validate a fully developed system to assure that it meets its requirements.

1. Alpha Testing

Alpha testing is a type of system testing conducted by the test team within the developing organization.

2. **Beta Testing**

   After internal testing, the software is given to users to test in their production environment.

3. **Acceptance Testing**

   Even if the software meets all user requirements, it may be rejected if users dislike its appearance or functionality.

4. **Performance Testing**

   Performance testing is conducted to ensure the system meets the non-functional requirements specified in the SRS document.

   Types of Performance Testing

   1. Stress testing
   2. Volume testing
   3. Configuration testing
   4. Compatibility testing
   5. Regression testing
   6. Recovery testing
   7. Maintenance testing
   8. Documentation testing
   9. Usability testing

★ Testing Documentation

Testing documents are prepared at different stages.

1. SRS document
2. Test Policy document
3. Test case document
4. Test report
5. Test logs
6. Test summary

★ Debugging

Debugging is the process that helps fix identified defects.

Identifying errors in a program code and then fixing them up are known as debugging.

★ What is Software Quality ?

Software Quality shows how good and reliable a product is.

Software Quality describes the desirable attributes of software products.

★ **Software Quality Factors**

- ➔ Portability
- ➔ Usability
- ➔ Reusability
- ➔ Correctness
- ➔ Maintainability

★ **The quality system activities encompass the following:**

1. auditing of projects
2. review of the quality system
3. development of standards, procedures, and guidelines, etc.
4. production of reports for the top management summarizing the effectiveness of the
5. quality system in the organization.

★ **ISO 9000 Certification**

- ➔ ISO published its 9000 series of standards in 1987.
- ➔ ISO 9000 specifies a set of guidelines for repeatable and high quality product development.
- ➔ The ISO 9000 standard specifies the guidelines for maintaining a quality system.

Types of ISO 9000

- ➔ ISO 9000 is a series of three standards: ISO 9001, ISO 9002, and ISO 9003.
- ➔ ISO 9001 applies to the organizations engaged in design, development, production, and servicing of goods.
- ➔ This is the standard that is applicable to most software development organizations.

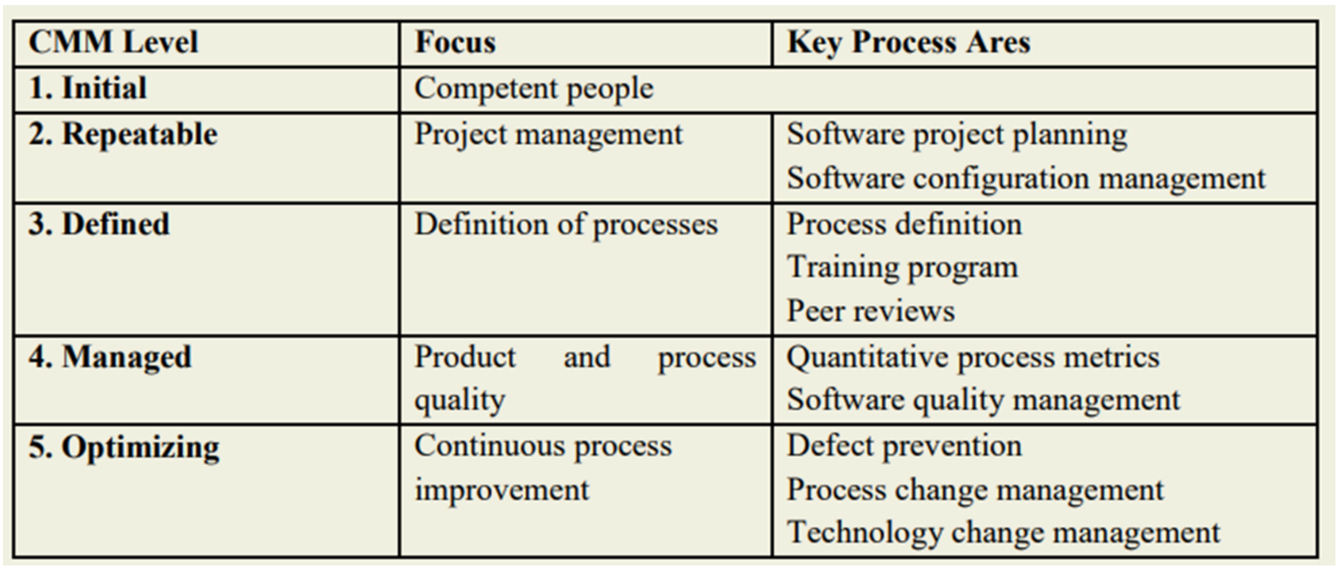
★ **SEI Capability Maturity Model**

SEI Capability Maturity Model (SEI CMM) helped organizations to improve the quality of the software.

SEI CMM classifies software development industries into the following five maturity levels.

Level 1: Initial
Level 2: Repeatable
Level 3: Defined
Level 4: Managed
Level 5: Optimizing

★ **Key process areas (KPA)**



CMM Level	Focus	Key Process Ares
1. Initial	Competent people	
2. Repeatable	Project management	Software project planning Software configuration management
3. Defined	Definition of processes	Process definition Training program Peer reviews
4. Managed	Product and process quality	Quantitative process metrics Software quality management
5. Optimizing	Continuous process improvement	Defect prevention Process change management Technology change management

★ **What is Software maintenance ?**

Software maintenance is a continuous process that occurs throughout the entire life cycle of the software system.

★ **Necessity of Software Maintenance**

➔ Market Conditions
➔ Client Requirements
➔ Host Modifications
➔ Organization Changes

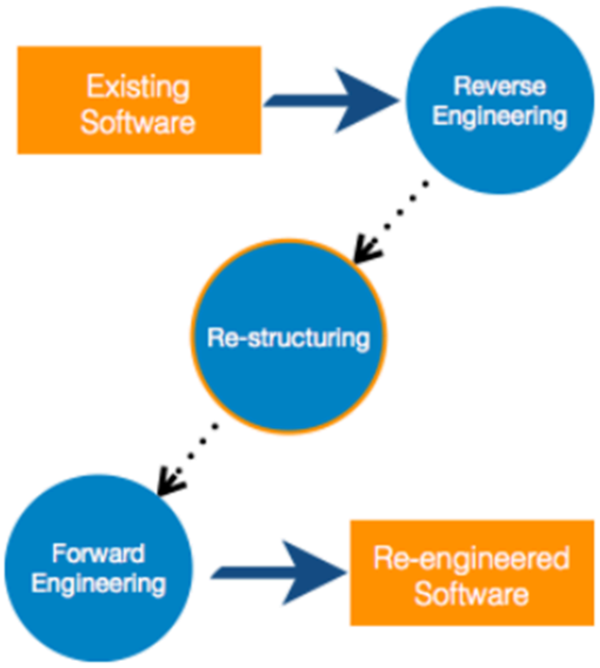
★ **Types of maintenance**

1. Corrective Maintenance
2. Adaptive Maintenance
3. Perfective Maintenance
4. Preventive Maintenance

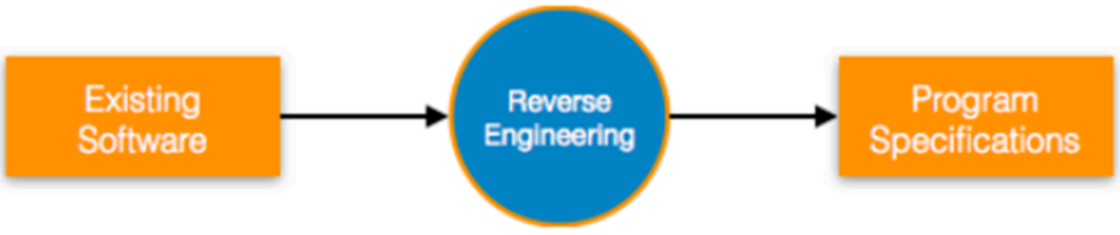
★ **Software reverse engineering**

Software reverse engineering is the process of analyzing code to understand its design and requirements specification.

Re-Engineering Process

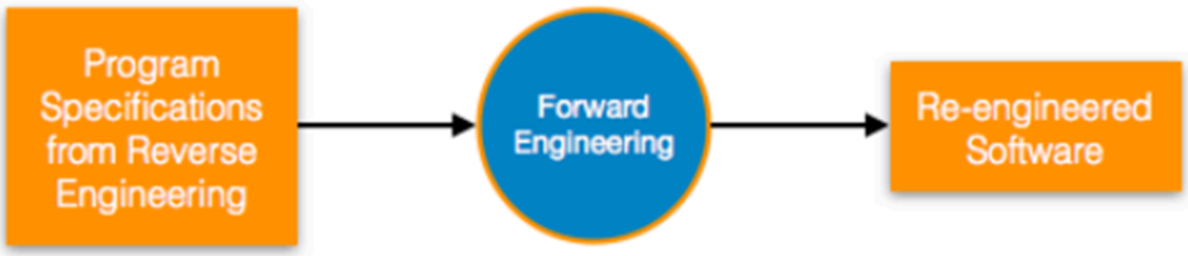


Reverse Engineering



Forward Engineering

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering.



★ **Legacy Software**

It is prudent to define a legacy system as any software system that is hard to maintain.

A legacy software is any system that's difficult to maintain.