SOFTWARE ENGINEERING

Unit 1: Introduction

Shree Harsh Attri
(Assistant Professor)
Department: Bachelor of Computer Applications

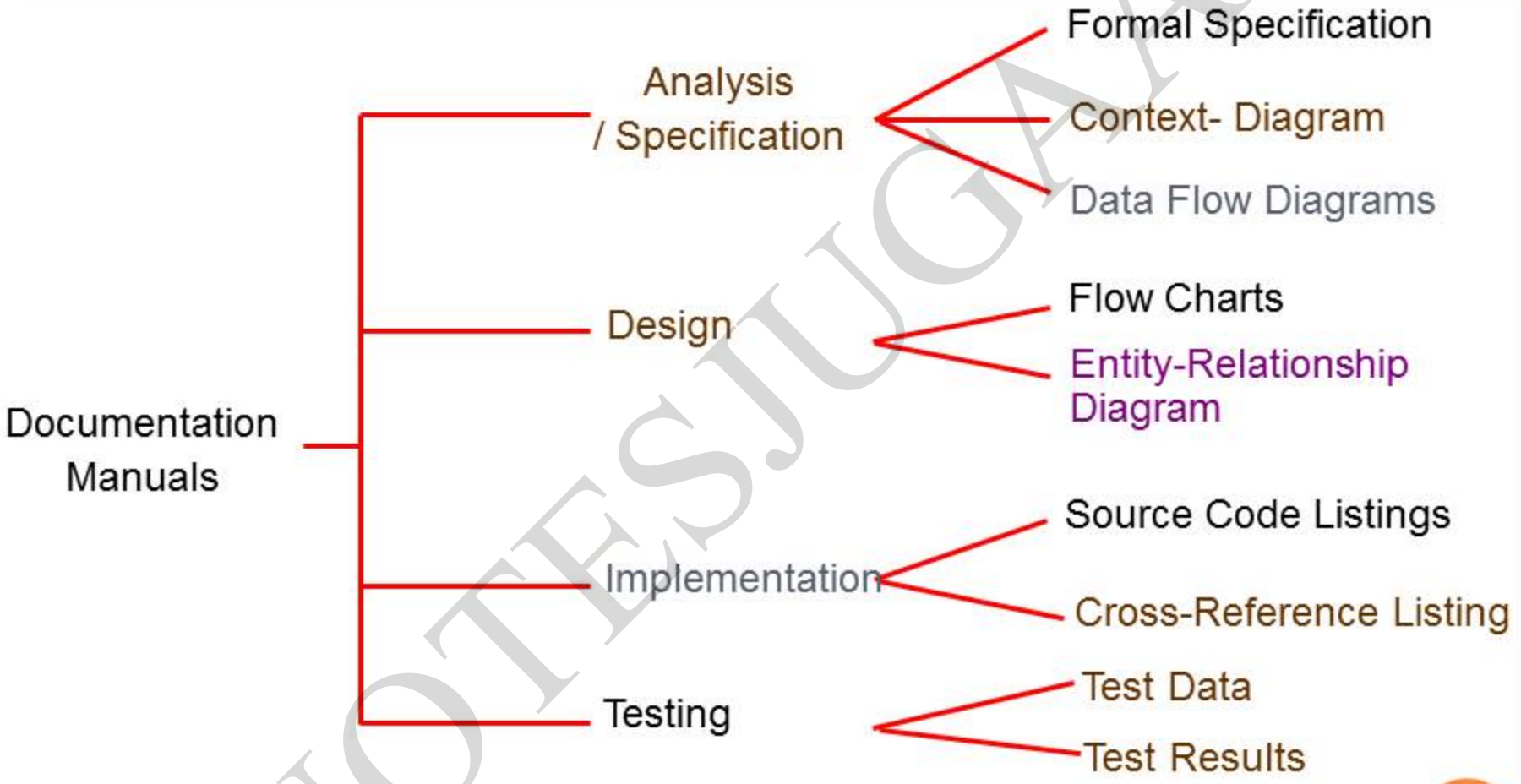
JIMS Engineering Management Technical Campus
Greater Noida (U.P)

WHAT IS SOFTWARE?

Computer Programs and **Associated Documentation**

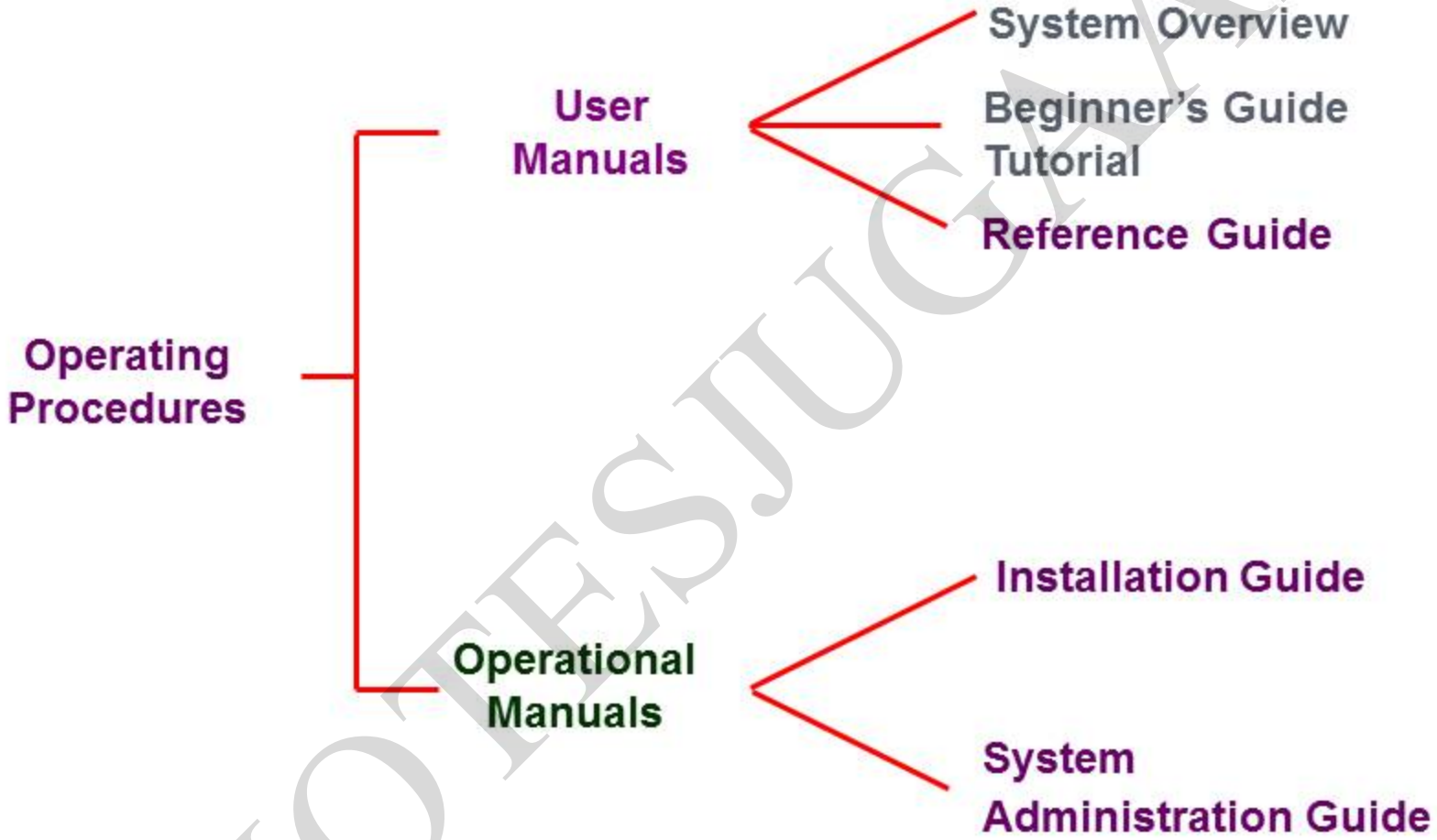
Software = Program + Documentation + Operating Procedures

DOCUMENTATION CONSISTS OF DIFFERENT TYPES OF MANUALS



List of documentation manuals

DOCUMENTATION CONSISTS OF DIFFERENT TYPES OF MANUALS



List of operating procedure manuals.

श्री हर्ष अत्रि (सहायक आचार्य)

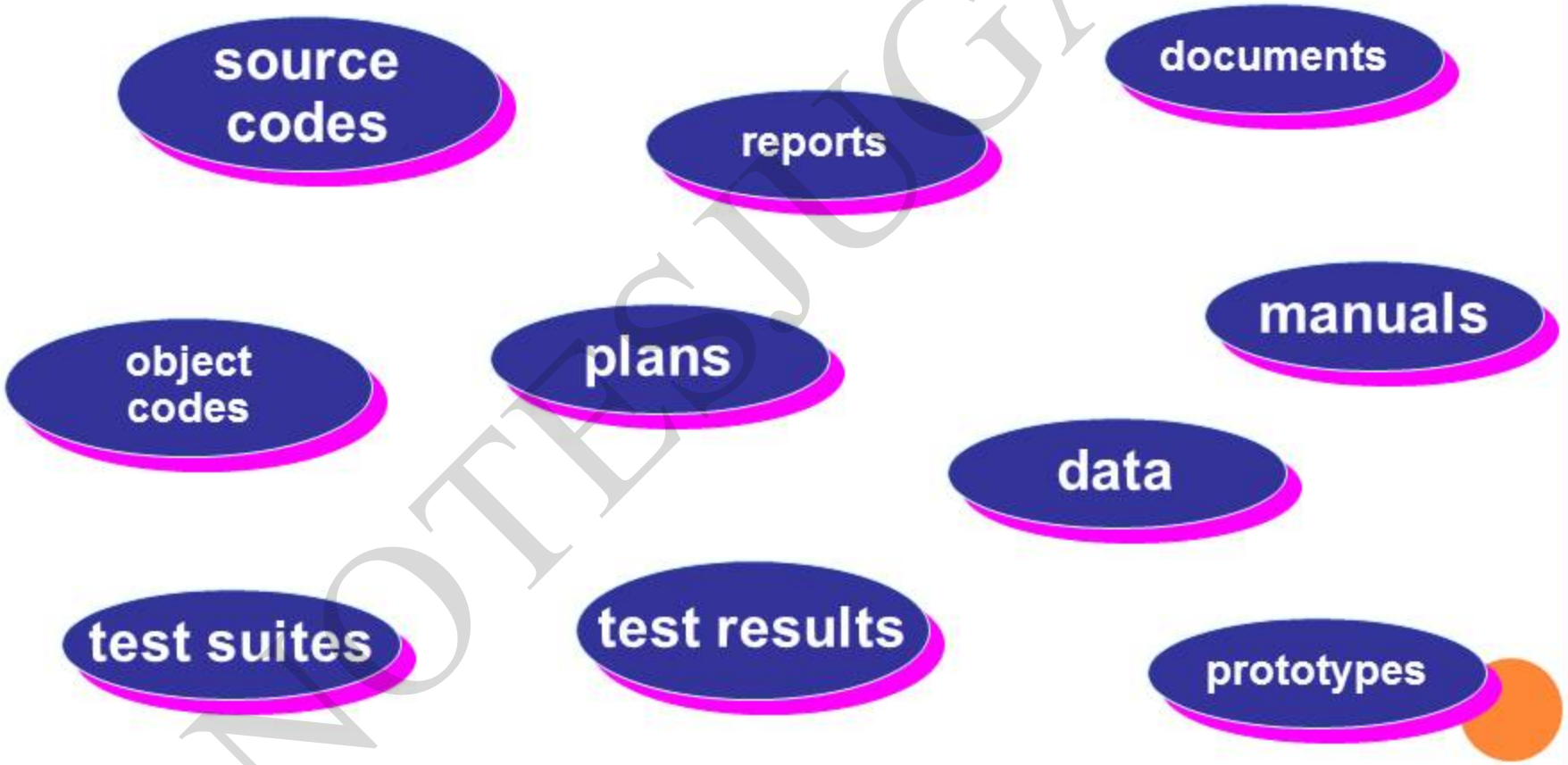
SOFTWARE PRODUCT

- **Software Products**: developed for a particular customer or may be developed for a general market.

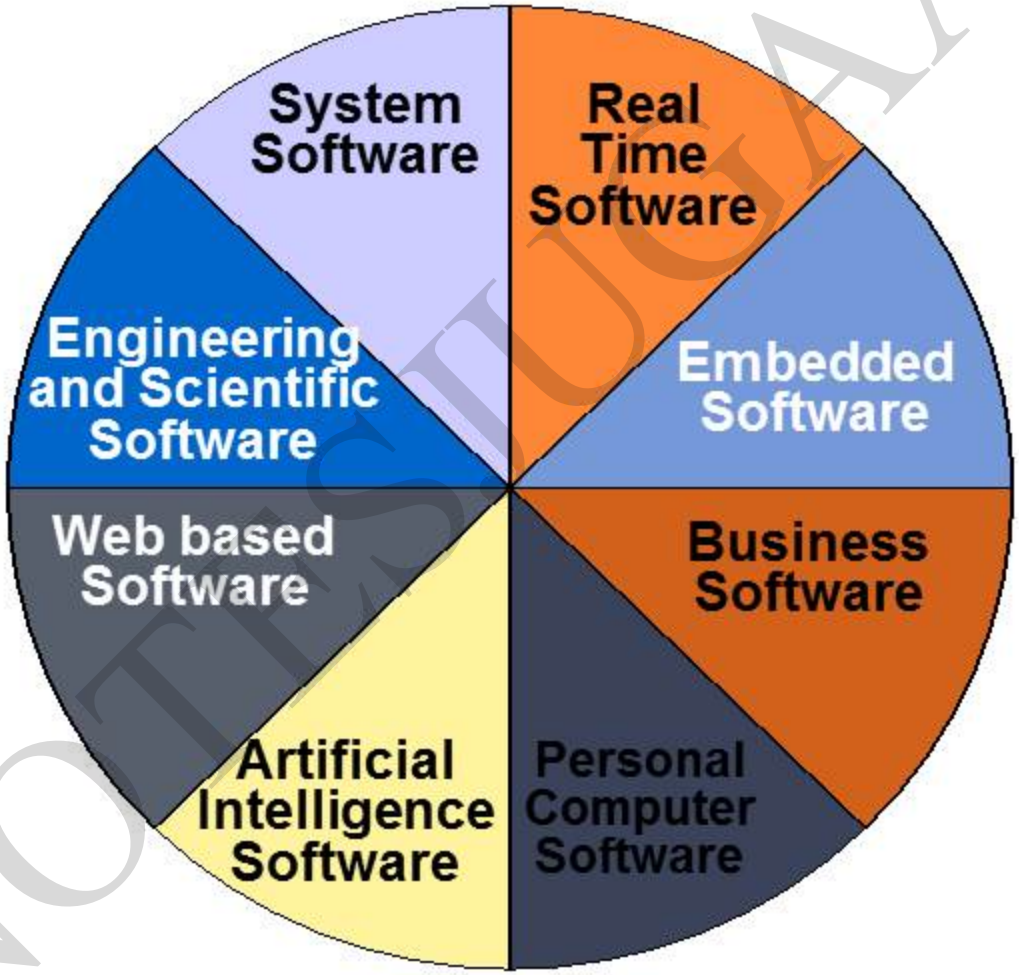
- **Software Products**:
  - **Generic** - developed to be sold to a range of different customers
  - **Be-spoke** (custom) - developed for a single customer according to their specification

SOFTWARE PRODUCT

A product designated for delivery to the user



SOFTWARE APPLICATIONS



WHAT IS SOFTWARE ENGINEERING?

Software Engineering is an engineering discipline which is concerned with all aspects of software production.

The development of software product is done by using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Software Engineers should

- adopt a systematic and organised approach to their work
- use appropriate tools and techniques depending on
  - the problem to be solved,
  - the development constraints and
- use the resources available

SOFTWARE PARADIGMS

Software paradigms refer to the methods and steps, which are taken while designing the software.

Software Development Paradigm

All engineering concepts are applied for the development of software. It includes various researches and requirement gathering which helps the software product to build.

It consists of : –

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development which includes : –

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is closely related to programming aspect of software development.

This includes : –

- Coding
- Testing
- Integration

CHARACTERISTICS OF GOOD SOFTWARE

A software product can be judged by what it offers and how well it can be used. The software must satisfy on the following grounds:

Operational

This tells us how well software works in operations. It can be measured on:

- **Budget**
- **Usability**
- **Efficiency**
- **Correctness**
- **Functionality**
- **Dependability**
- **Security**
- **Safety**

Transitional

This aspect is important when the software is moved from one platform to another:

- **Portability**
- **Interoperability**
- **Reusability**
- **Adaptability**

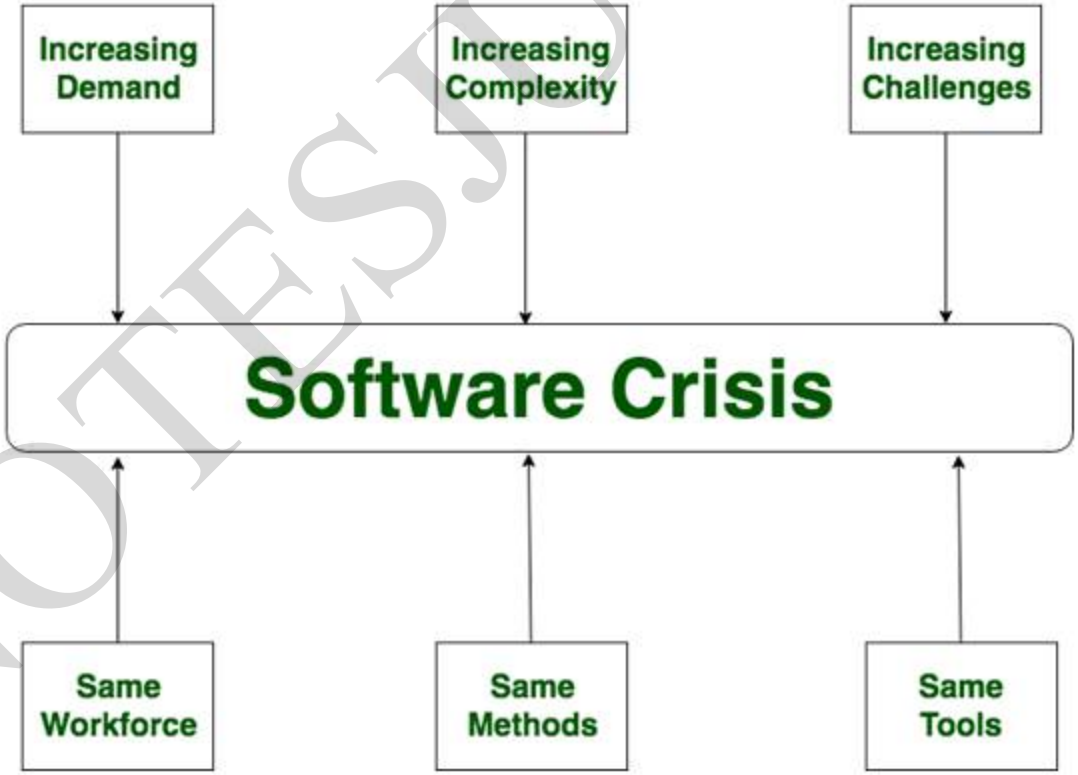
Maintenance

how well a software has the capabilities to maintain itself in the ever-changing environment:

- **Modularity**
- **Maintainability**
- **Flexibility**
- **Scalability**

SOFTWARE CRISIS

- **Software Crisis** is a term used for **difficulty in writing useful and efficient computer programs in the required time.**
- As, fast increasing in ***software demand, software complexity and software challenges***, if there is a use **same workforce, same methods and same tools**, then this will raise some problems like ***software budget problem, software efficiency problem, software quality problem, software managing and delivering problem etc***. This condition is called software crisis.



CAUSES OF SOFTWARE CRISIS

- Project running over budget
- Project running over time
- Software was very inefficient
- Software was of low Quality
- Software often didn't meet requirement
- Project were unmanageable and code difficult to maintain
- Software was never delivered

SOME SOFTWARE FAILURES

The Patriot Missile

- First time used in Gulf war
- Used as a defense from Iraqi Scud missiles
- Failed several times including one that killed 28 US soldiers in Dhahran, Saudi Arabia

Reasons:

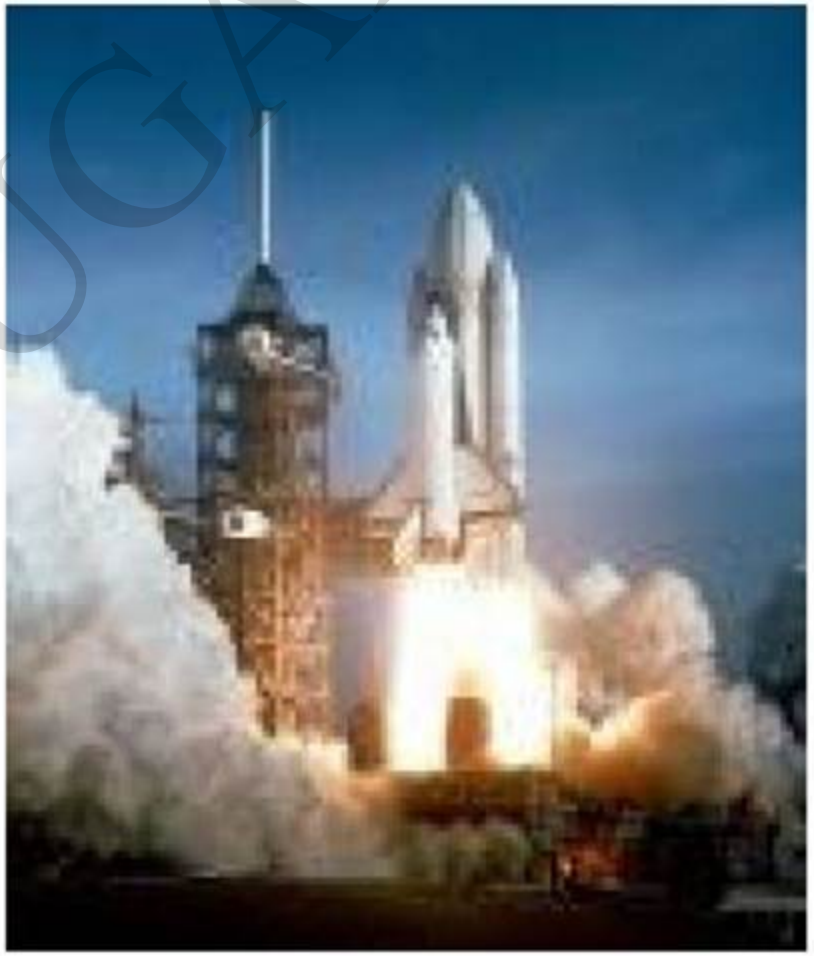
A small timing error in the system's clock accumulated to the point that after 14 hours, the tracking system was no longer accurate. In the Dhahran attack, the system had been operating for more than 100 hours.

SOME SOFTWARE FAILURES

The Space Shuttle

Part of an abort scenario for the Shuttle requires fuel dumps to lighten the spacecraft. It was during the second of these dumps that a (software) crash occurred.

...the fuel management module, which had performed one dump and successfully exited, restarted when recalled for the second fuel dump...



SOME SOFTWARE FAILURES

Financial Software

Many companies have experienced failures in their accounting system due to faults in the software itself. The failures range from producing the wrong information to the whole system crashing.

SOFTWARE PROCESSES

- The process that deals with the **technical and management** issues of software development is called a software process. The process should be able to produce *high-quality software at low cost, and scalability* means that it should also be applicable for large software projects.

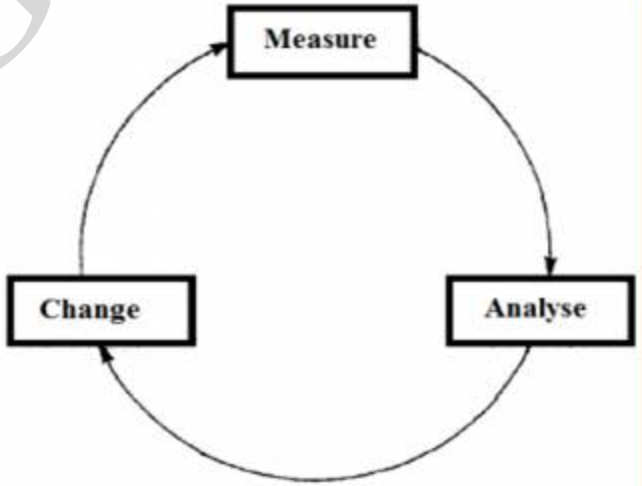
There are four fundamental software process that are general to all software processes as follows:

- ***Software specification:*** customer give its requirements and software engineers define the software to be produced and the constraints on its operation and functions.
- ***Software development:*** the software is designed and programmed as per the customer specification by the software development engineer team.
- ***Software validation:*** Software validation of software process activity is checked the software to ensure that it is what the customer requires as in the specification.
- ***Software evolution:*** Software evolution process activity includes the software modified to adapt it to varying customer and markets promote requirements.

SOFTWARE PROCESSES

Software process improvement is a cyclical activity; it involves three principal stages:

1. Process Measurement: It involves the measurement of current project attributes or the product. The aim is to advance the measured attributes as per the goals of the organization involved in process improvement.



2. Process Analysis: It involves the current process is assessed, and process weaknesses and bottlenecks are identified. Process models that describe the process are usually developed during process analysis stage.

3. Process Change: Changes to the process that have been identified during analysis are introduced.

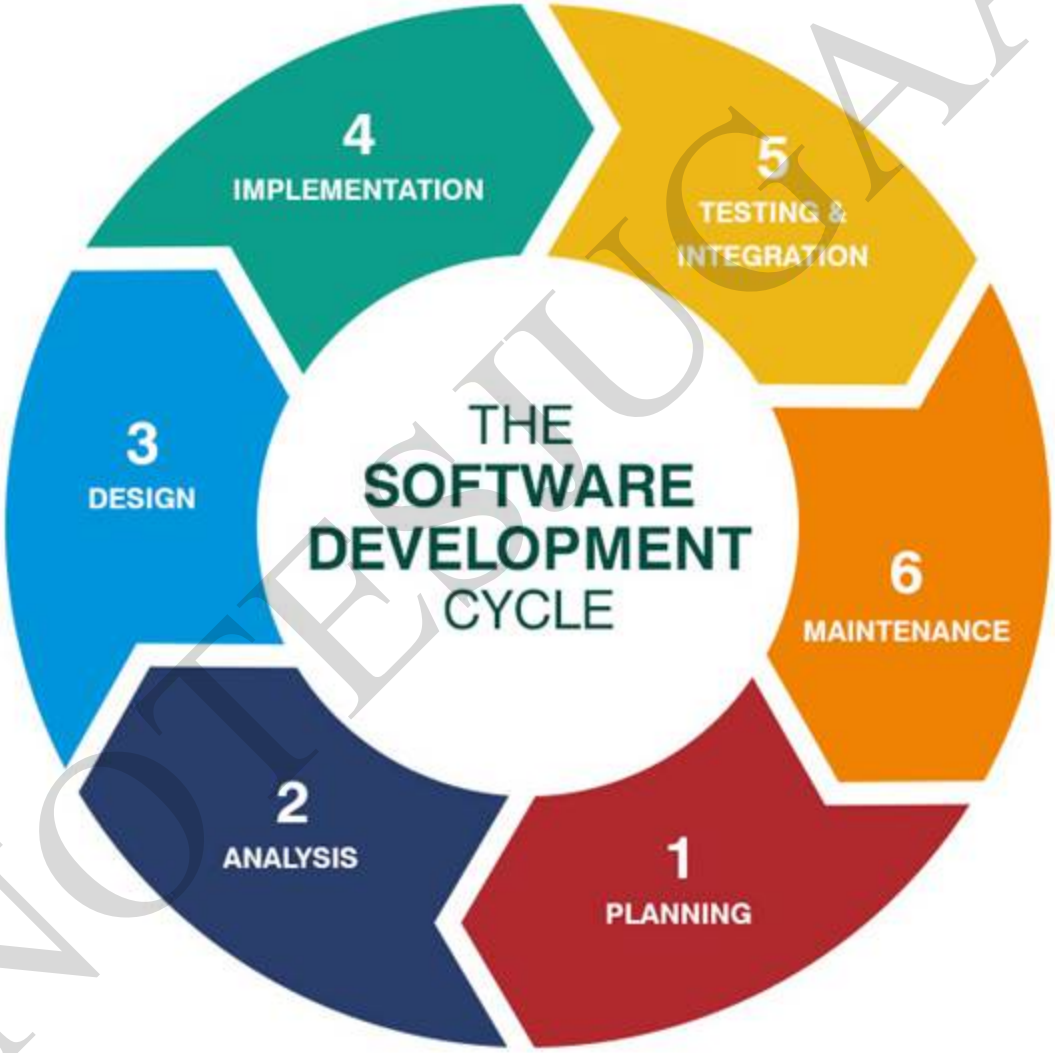
Note: Software process improvement is a long-term and continuous activity as, whatever new processes are introduced, the business environment will change and these processes will themselves have to evolve to take changes into account.

SOFTWARE PROCESSES CHARACTERISTICS

Software process have following Characteristics :

- **Understandability:** To what extent is the process explicitly defined and how easy is it to understand the process definition?
- **Visibility:** Do the process activities conclude in clear results so that the progress of the process is externally visible?
- **Supportability:** To what extent can CASE *(**C**omputer **A**ided **S**oftware **E**ngineering)* tools be used to support the process activities?
- **Acceptability:** Is the defined process acceptable to and usable by the engineers responsible for producing the software product?
- **Reliability:** Is the process designed in such a way that process errors are avoided or trapped before they result in product error?
- **Robustness:** Can the process continue in spite of unexpected problem?
- **Maintainability:** Can the process evolve to reflect changing organizational requirements or identified process improvements?
- **Rapidly:** How fast can the process of delivering a system from a given specification be completed?

SOFTWARE LIFE CYCLES



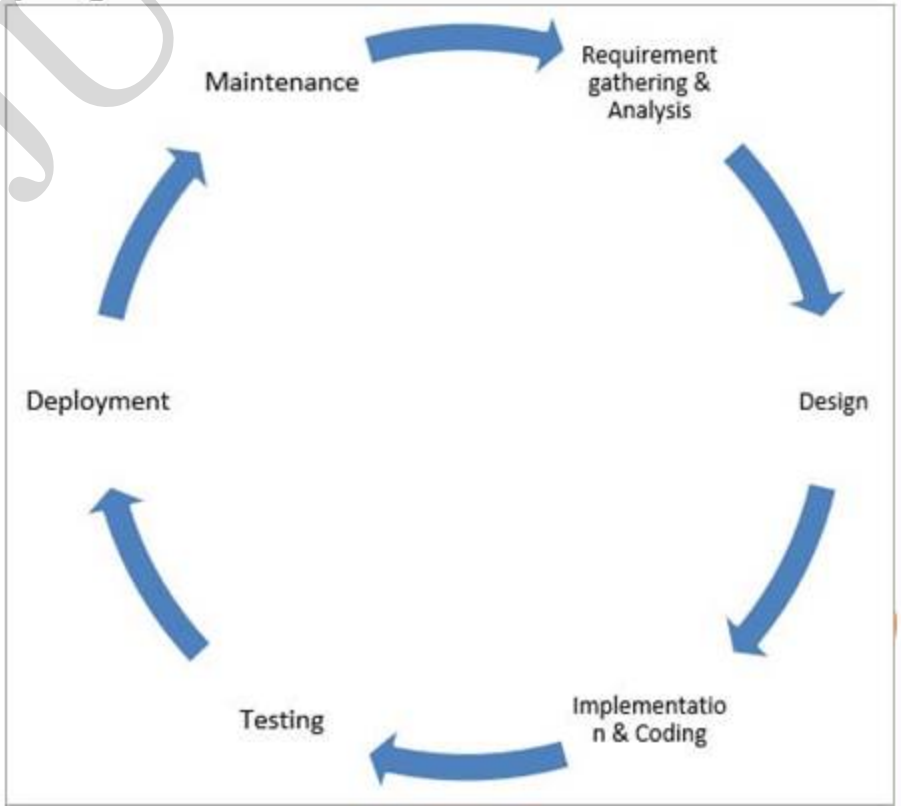
SOFTWARE LIFE CYCLES

- Software Development Life Cycle (SDLC) is a well-defined, **structured sequence** of stages in software engineering to develop the intended software product.
- Software Development Life Cycle (SDLC) covers the detailed **plan for building, deploying and maintaining the software**.
- SDLC defines the complete cycle of development i.e. all the tasks involved in planning, creating, testing, and deploying a Software Product.

SDLC Phases

Given below are the various phases:

- ✓ **Requirement Gathering and Analysis**
- ✓ **Design**
- ✓ **Implementation or Coding**
- ✓ **Testing**
- ✓ **Deployment**
- ✓ **Maintenance**



श्री हर्ष अत्रि (सहायक आचार्य)

SOFTWARE LIFE CYCLE PHASES

jims Greater Noida

1. Requirement Gathering and Analysis

- During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.
- Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

For Example:

- A customer wants to have an application which involves money transactions. In this case, the requirement has to be clear like what kind of transactions will be done, how it will be done, in which currency it will be done, etc.
- Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.
- Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

SOFTWARE LIFE CYCLE PHASES

2. **Design:** In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.
3. **Implementation or Coding:** Implementation / Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.
4. **Testing:** Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

Retesting, regression testing is done until the point at which the software is as per the customer's expectation. Testers refer SRS document to make sure that the software is as per the customer's standard.

SOFTWARE LIFE CYCLE PHASES

5. **Deployment:** Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.

   In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

6. **Maintenance:** After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

Software Life Cycles Models

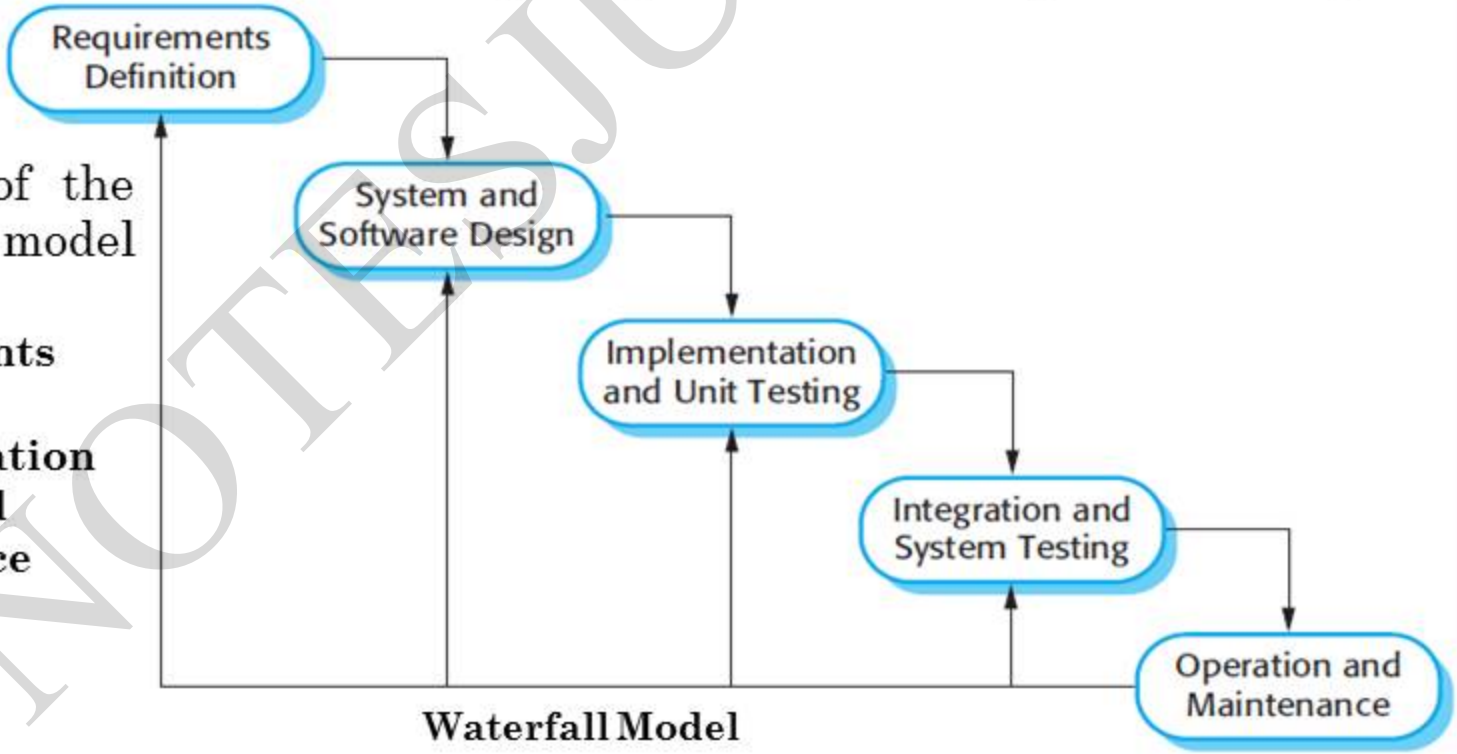
1. Water Fall Model
2. Prototype Model
3. Evolutionary Model
4. Spiral Model

WATERFALL MODEL

- The waterfall model is a sequential approach, where each fundamental activity of a process represented as a separate phase, arranged in linear order.
- Waterfall model is a **plan-driven process** in which all the activities including planning and scheduling must be considered before starting working on them.
  - **Plan-Driven Process** is a process where all the activities are planned first, and the progress is measured against the plan. While the **Agile Process**, planning is incremental and it's easier to change the process to reflect requirement changes.

The phases of the waterfall model are:

- ✓ **Requirements**
- ✓ **Design**
- ✓ **Implementation**
- ✓ **Testing and**
- ✓ **Maintenance**



Waterfall Model

WATERFALL MODEL

jims Greater Noida

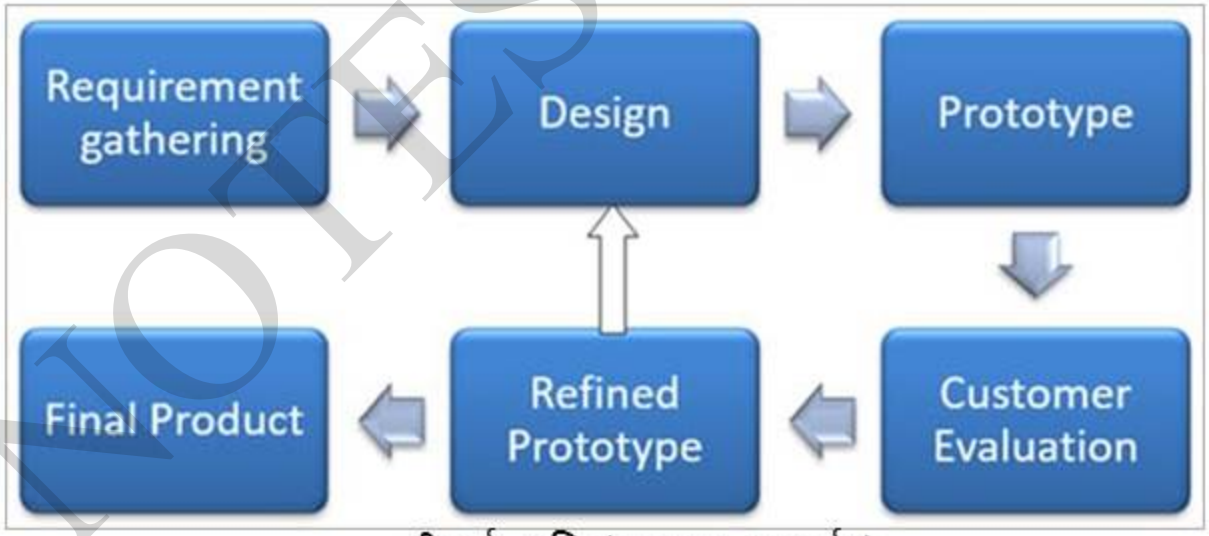
Different Phases	Activities performed in each stage
Requirement Gathering Stage	• During this phase, detailed requirements of the software system to be developed are gathered from client
Design Stage	• Plan the programming language, for Example Java, PHP, Dot Net etc. • or database like Oracle, MySQL, etc. • or other high-level technical details of the project
Built Stage	• After design stage, it is built stage, that is nothing but coding the software
Test Stage	• In this phase, testing of the software to verify that it is built as per the specifications given by the client.
Deployment stage	• Deploy the application in the respective environment
Maintenance stage	• Once system is in use, and later required to change the code as per customer request

Advantages and Dis-advantages of Waterfall Model

Advantages	Dis-Advantages
• Before the next phase of development, each phase must be completed	• Error can be fixed only during the phase
• Suited for smaller projects where requirements are well defined	• It is not desirable for complex project where requirement changes frequently
• They should perform quality assurance test (Verification and Validation) before completing each stage	• Testing period comes quite late in the developmental process
• Elaborate documentation is done at every phase of the software's development cycle	• Documentation occupies a lot of time of developers and testers
• Project is completely dependent on project team with minimum client intervention	• Clients valuable feedback cannot be included with ongoing development phase
• Any changes in software is made during the process of the development	• Small changes or errors that arise in the completed software may cause a lot of problems

PROTOTYPE MODEL

- The prototype model is a model in which the prototype is developed prior to the actual software.
- a prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc.
- the client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation
- Software prototypes are built prior to the actual software to get valuable feedback from the customer. Feedbacks are implemented and the prototype is again reviewed by the customer for any change. This process goes on until the model is accepted by the customer.



THE PHASES OF PROTOTYPE MODEL

1. **Establish objectives**: The objectives of the prototype should be made explicit from the start of the process. Is it to validate system requirements, or demonstrate feasibility, etc.
2. **Define prototype functionality**: Decide what are the inputs and the expected output from a prototype. To reduce the prototyping costs and accelerate the delivery schedule, you may ignore some functionality, such as response time and memory utilization unless they are relevant to the objective of the prototype.
3. **Develop the prototype**: The initial prototype is developed that includes only user interfaces.
4. **Evaluate the prototype**: Once the users are trained to use the prototype, they then discover requirements errors. Using the feedback both the specifications and the prototype can be improved. If changes are introduced, then a repeat of steps 3 and 4 may be needed.
5. **Prototyping** is not a standalone, complete development methodology, but rather an approach to be used in the context of a full methodology (such as incremental, spiral, etc).

PROTOTYPE MODEL

Advantages of Prototype Model:

- Prototype model reduces the cost and time of development as the defects are found much earlier.
- Missing feature or functionality or a change in requirement can be identified in the evaluation phase and can be implemented in the refined prototype.
- Involvement of a customer from the initial stage reduces any confusion in the requirement or understanding of any functionality.

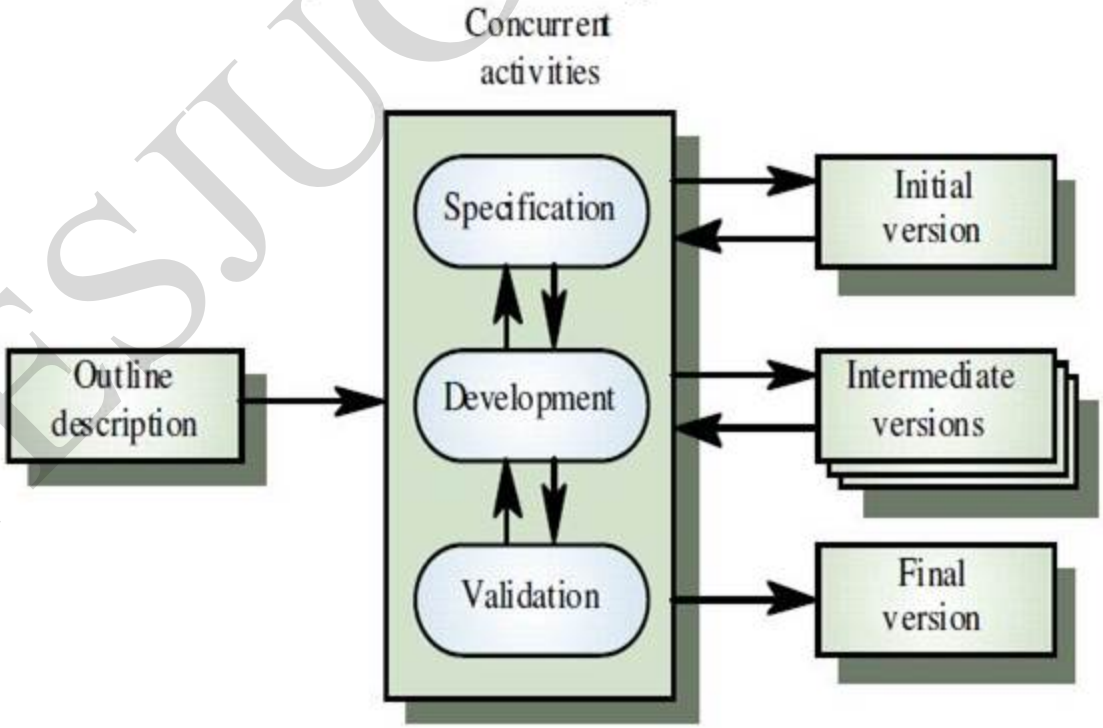
Disadvantages of Prototype Model:

- Since the customer is involved in every phase, the customer can change the requirement of the end product which increases the complexity of the scope and may increase the delivery time of the product.

EVOLUTIONARY MODEL

- The Evolutionary development model ***divides the development cycle into smaller, incremental waterfall models*** in which users are able to get access to the product at the end of each cycle.
- ***Feedback is provided by the users*** on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan or process. Therefore, the software product evolves with time.

- They are iterative in Process.
- They enable the software developer to develop increasingly more Complex versions of the software.
- Like all Complex systems, software involve over period of the time and hence evolutionary models are more suited to software development.



EVOLUTIONARY MODEL

Application of Evolutionary Model:

- It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
- Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Advantages:

- In evolutionary model, a user gets a chance to experiment partially developed system.
- It reduces the error because the core modules get tested thoroughly.

Disadvantages:

- Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

SPIRAL MODEL

- The Evolutionary (Spiral) Model emphasizes **risk analysis**, and thus requires customers to accept this analysis and act on it.
- This requires both trust in the developer as well as the willingness to spend more to fix the issues. If the implementation of risk analysis will greatly affect the profits of the project, the spiral model should not be used. Software developers have to actively look for possible risks, and analyse it accurately for the spiral model to work.

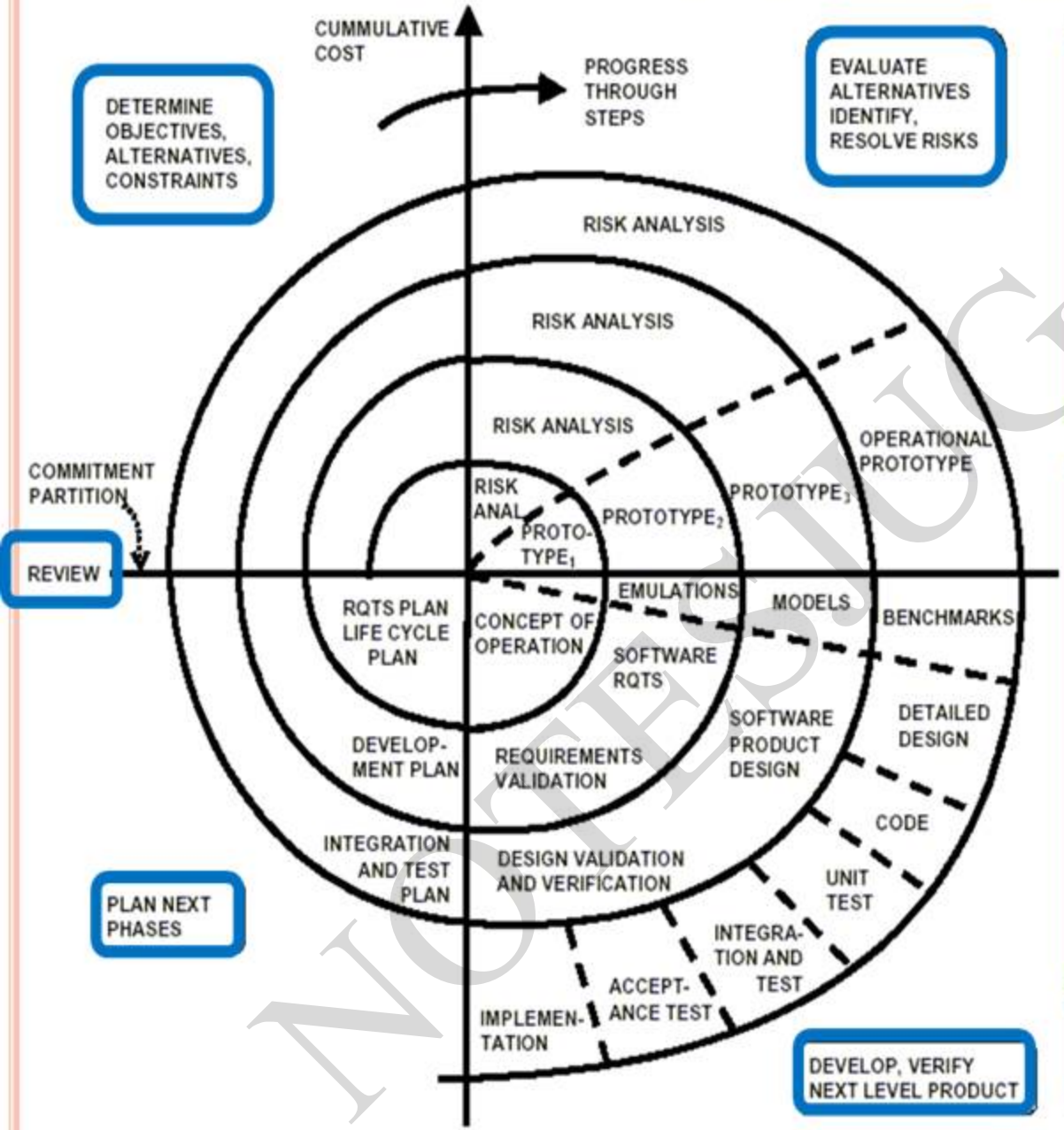
Strengths of the Spiral Model

- Provides early indication of intractable risks, without much cost.
- Users see the system early because of rapid prototyping tools.
- Critical high-risk functions are developed first.
- The design does not have to be perfect.
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

Weaknesses of the Spiral Model

- Time spent for evaluating risks too large for small or low-risk projects.
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive.
- The model is complex.
- Risk assessment expertise is required Spiral may continue indefinitely.
- Developers must be reassigned during non-development phase activities.
- May be hard to define objective, verifiable milestones that indicate readiness to move to the next iteration.

SPIRAL MODEL



- Formulate plans to: identify software targets, implement the program, clarify the project development restrictions
- Risk analysis: an analytical assessment of selected programs, to consider how to identify and eliminate risk
- Implementation of the project: the implementation of software development and verification

- *The spiral model emphasizes risk analysis, and thus requires customers to accept this analysis and act on it.*
- *This requires both trust in the developer as well as the willingness to spend more to fix the issues.*
- *If the implementation of risk analysis will greatly affect the profits of the project, the spiral model should not be used.*
- *Software developers have to actively look for possible risks, and analyse it accurately for the spiral model to work.*

Major Issues

Selection of a Life Cycle Model

Selection of a model is based on:

a) Requirements

b) Development team

c) Users

d) Project type and associated risk

Based On Characteristics Of Requirements

Requirements	Waterfall	Prototype	Evolutionary Development	Spiral
Are requirements easily understandable and defined?	Yes	No	No	No
Do we change requirements quite often?	No	Yes	No	Yes
Can we define requirements early in the cycle?	Yes	No	Yes	No
Requirements are indicating a complex system to be built	No	Yes	Yes	Yes

Based On Status of Development Team

Requirements	Waterfall	Prototype	Evolutionary Development	Spiral
Less experience on similar projects?	No	Yes	No	Yes
Less domain knowledge (new to technology)	Yes	No	Yes	Yes
Less experience on tools to be used	Yes	No	No	Yes
Availability of training if required	No	No	Yes	No

BASED ON USER'S PARTICIPATION

Requirements	Waterfall	Prototype	Evolutionary Development	Spiral
User involvement in all phases	No	Yes	No	No
Limited user participation	Yes	No	Yes	Yes
User have no previous experience of participation in similar projects	No	Yes	Yes	Yes
Users are experts of problem domain	No	Yes	Yes	No

BASED ON TYPE OF PROJECT WITH ASSOCIATED RISK

Requirements	Waterfall	Prototype	Evolutionary Development	Spiral
Project is the enhancement of the existing system	No	No	Yes	No
Funding is stable for the project	Yes	Yes	No	No
High reliability requirements	No	No	Yes	Yes
Tight project schedule	No	Yes	Yes	Yes
Use of reusable components	No	Yes	No	Yes
Are resources (time, money, people etc.) scare?	No	Yes	No	Yes

REFERENCE

Book

Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007

Website

- https://www.geeksforgeeks.org/software-engineering-software-crisis/
- https://www.tutorialspoint.com/software_engineering/software_engineering_overview.htm
- http://ecomputernotes.com/software-engineering/explain-software-process-characteristics
- https://er.yuvayana.org/software-process-improvement-activities-attributes-and-characteristics/
- https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/
- https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc
- https://www.guru99.com/what-is-sdlc-or-waterfall-model.html

SOFTWARE ENGINEERING

Unit 1: Software Requirements Analysis & Specifications

Shree Harsh Attri
(Assistant Professor)
Department: Bachelor of Computer Applications

JIMS Engineering Management Technical Campus
Greater Noida (U.P)

jims Greater Noida

Requirements describe

What not How

Produces one large document written in natural language contains a description of what the system will do without describing how it will do it.

Crucial process steps

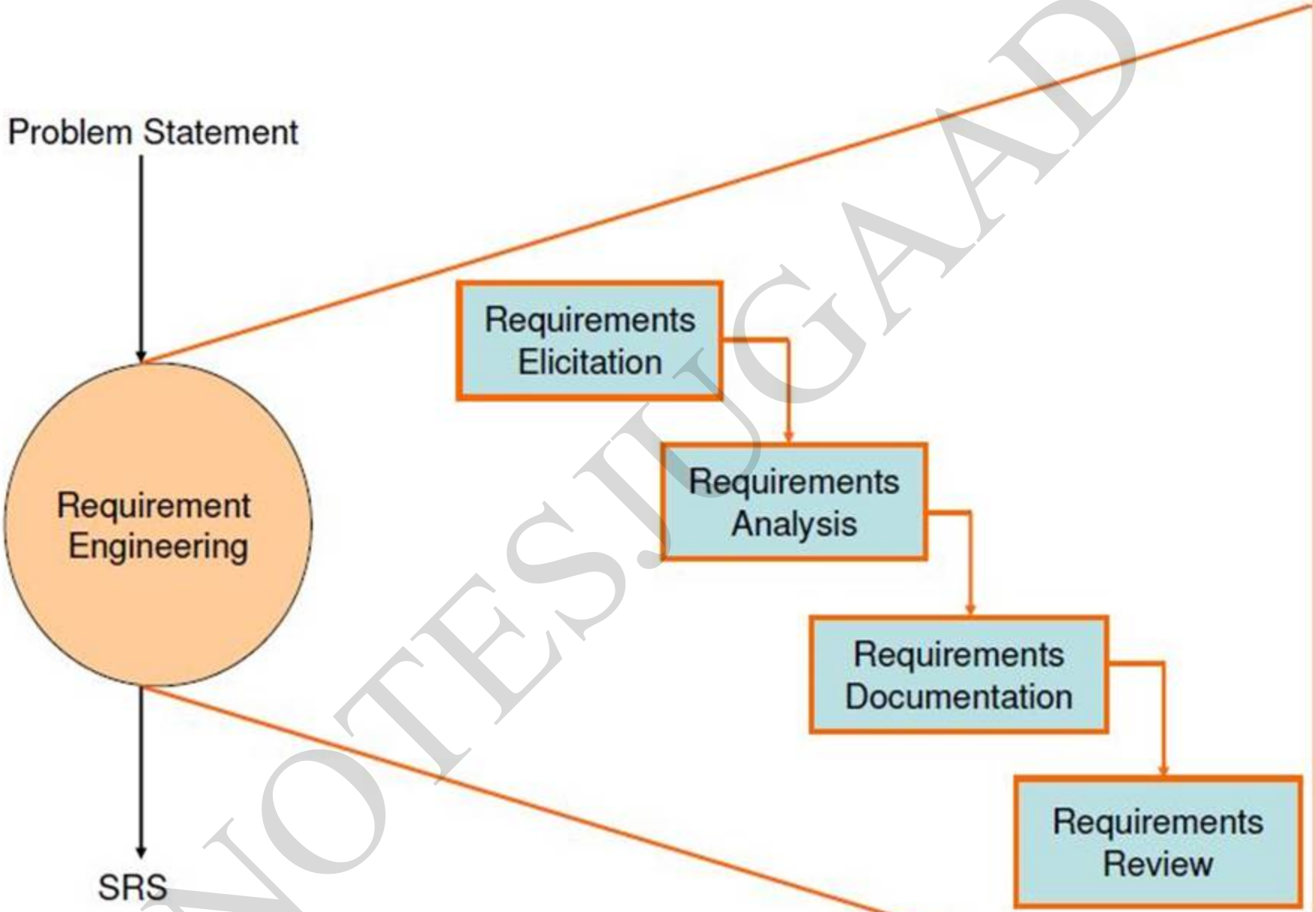
Quality of product → Process that creates it

Without well written document

-- Developers do not know what to build

-- Customers do not know what to expect

-- What to validate



Crucial Process Steps of requirement engineering

Requirement Engineering

The software requirements are description of **features and functionalities of the target system**. Requirements convey the expectations of users from the software product. The requirements can be **obvious or hidden, known or unknown, expected or unexpected** from client's point of view.

Requirement Engineering

- The process to gather the software requirements from client, analyze and document them is known as requirement engineering.
- **The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification (SRS)' document.**

Requirement Engineering Process:

It is a four step process, which includes :–

1. Feasibility Study
2. Requirement Gathering
3. Software Requirement Specification
4. Software Requirement Validation

Requirement Engineering

jims Greater Noida

1. **Feasibility Study:** This feasibility study is focused towards goal of the organization. It explores **technical aspects** of the project and product such as ***usability, maintainability, productivity and integration ability***. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization.
2. **Requirement Gathering:** After getting the positive feasibility report, analyst and engineers starts with gathering requirements from the user like *what features user want to be in software etc.*
3. **Software Requirement Specification:** SRS is a document created by system analyst after the requirements are collected from various stakeholders. SRS defines how the intended software will ***interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations*** etc.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

4. **Software Requirement Validation:**

Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

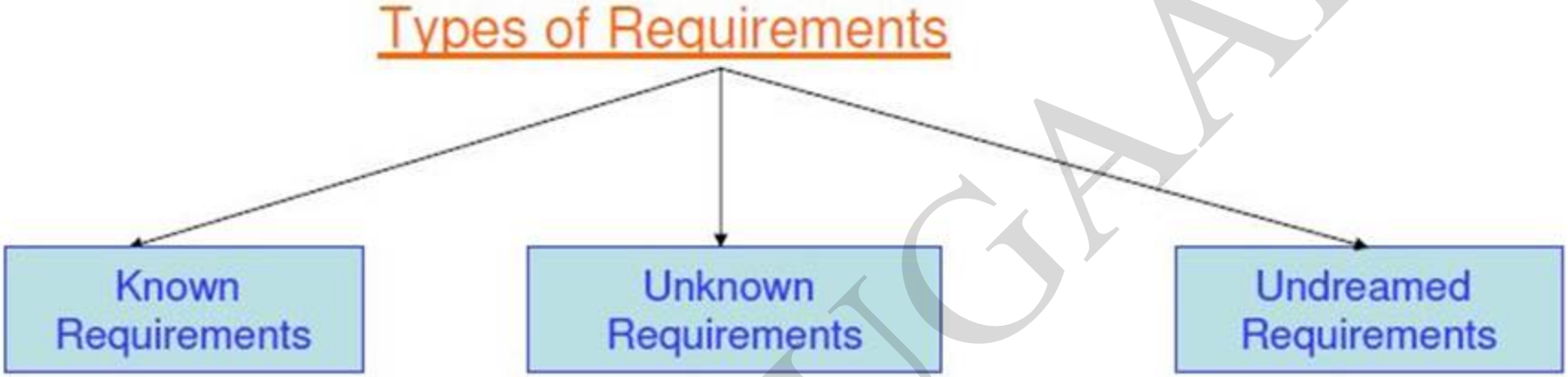
Example

A University wish to develop a software system for the student result management of its M.Tech. Programme.

A problem statement is to be prepared for the software development company.

The problem statement may give an overview of the existing system and broad expectations from the new software system.

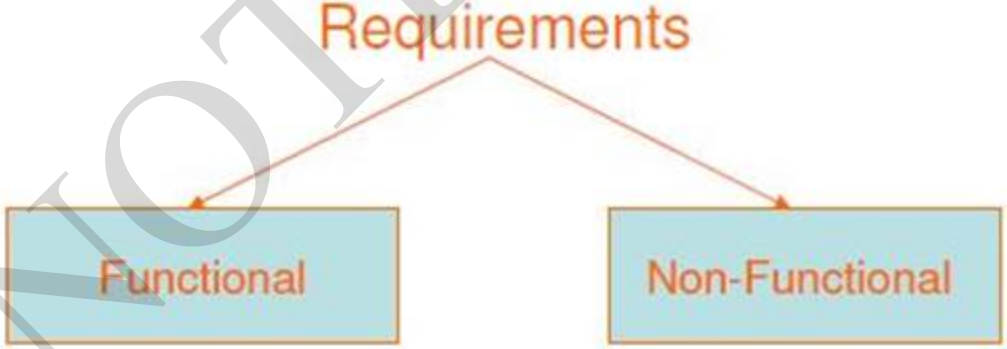
Types of Requirements



Stakeholder: Anyone who should have some direct or indirect influence on the system requirements.

--- User

--- Affected persons



Types of Requirements

Functional requirements describe what the software has to do. They are often called product features.

Non Functional requirements are mostly quality requirements. That stipulate how well the software does, what it has to do.

- Availability
- Reliability
- Usability
- Flexibility

} For Users

- Maintainability
- Portability
- Testability

} For Developers

Requirements Elicitation

jims Greater Noida

Perhaps

- Most difficult
- Most critical
- Most error prone
- Most communication intensive

Succeed

→ effective customer developer partnership

Selection of any method

1. It is the only method that we know
2. It is our favorite method for all situations
3. We understand intuitively that the method is effective in the present circumstances.

REQUIREMENTS ELICITATION TECHNIQUES

Following are the number of requirements elicitation methods:-

1. **Interviews**
2. **Brainstorming Sessions**
3. **Facilitated Application Specification Technique (FAST)**
4. **Quality Function Deployment (QFD)**
5. **Use Case Approach**

Requirements Elicitation

1. Interviews

Both parties have a common goal → Success of the project

--- open ended
--- structured
} Interview

Selection of stakeholder

1. Entry level personnel
2. Middle level stakeholder
3. Managers
4. Users of the software (Most important)

Requirements Elicitation

Types of questions.

- Any problems with existing system
- Any Calculation errors
- Possible reasons for malfunctioning
- No. of Student Enrolled

5. Possible benefits
6. Satisfied with current policies
7. How are you maintaining the records of previous students?
8. Any requirement of data from other system
9. Any specific problems
10. Any additional functionality
11. Most important goal of the proposed development

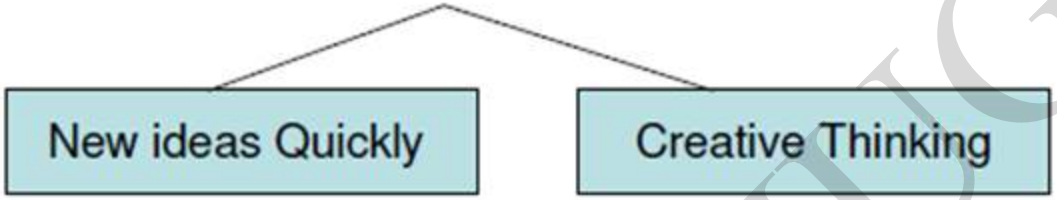
Requirements Elicitation

2. Brainstorming Sessions

It is a group technique

↓

group discussions



Prepare long list of requirements

Categorized
Prioritized
Pruned

*Idea is to generate views ,not to vet them.

Groups

1. Users 2. Middle Level managers 3. Total Stakeholders

Requirements Elicitation

jims Greater Noida

A Facilitator may handle group bias, conflicts carefully.

-- Facilitator may follow a published agenda

-- Every idea will be documented in a way that everyone can see it.

--A detailed report is prepared.

3. Facilitated Application specification Techniques (FAST)

-- Similar to brainstorming sessions.

-- Team oriented approach

-- Creation of joint team of customers and developers.

Requirements Elicitation

jims Greater Noida

Guidelines

1. Arrange a meeting at a neutral site.
2. Establish rules for participation.
3. Informal agenda to encourage free flow of ideas.
4. Appoint a facilitator.
5. Prepare definition mechanism board, worksheets, wall stickier.
6. Participants should not criticize or debate.

FAST session Preparations

Each attendee is asked to make a list of objects that are:

Requirements Elicitation

1. Part of environment that surrounds the system.
2. Produced by the system.
3. Used by the system.

A. List of constraints
B. Functions
C. Performance criteria

Activities of FAST session

1. Every participant presents his/her list
2. Combine list for each topic
3. Discussion
4. Consensus list
5. Sub teams for mini specifications
6. Presentations of mini-specifications
7. Validation criteria
8. A sub team to draft specifications

Requirements Elicitation

4. Quality Function Deployment (QFD):

In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.

3 types of requirements are identified –

Normal Requirements – In this the objective and goals of the proposed software are discussed with the customer.
Example – normal requirements for a result management system may be *entry of marks, calculation of results* etc.

Expected Requirements – These requirements are so obvious that the customer need not explicitly state them.
Example – *protection from unauthorised access*.

Exciting Requirements – It includes features that are beyond customer's expectations and prove to be very satisfying when present.
Example – *when an unauthorised access is detected, it should backup and shutdown all processes*.

Requirements Elicitation

The major steps involved in QFD are –

1. Identify all the stakeholders, eg. Users, developers, customers etc.
2. List out all requirements from customer.
3. A value indicating degree of importance is assigned to each requirement.
4. In the end the final list of requirements is categorised as –
   - It is possible to achieve
   - It should be deferred and the reason for it
   - It is impossible to achieve and should be dropped off

5. Use Case Approach:

This technique combines **text and pictures** to provide a better understanding of the requirements.

The use cases describe the **'what',** of a system and not **'how'.** Hence they only give a **functional view** of the system.

The components of the use case design includes three major things –

1. **Actor**
2. **Use cases**
3. **Use case diagram**

Requirements Elicitation

1. **Actor –** It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a **person, machine, information system** etc. It is represented as a **stick figure**. *Actors can be primary actors or secondary actors.*
   - **Primary actors** – It requires assistance from the system to achieve a goal.
   - **Secondary actor** – It is an actor from which the system needs assistance.
2. **Use cases –** They describe the sequence of interactions between actors and the system. They capture ***who(actors) do what(interaction) with the system.*** A complete set of use cases specifies all possible ways to use the system.
3. **Use case diagram –** A use case diagram graphically represents ***what happens when an actor interacts with a system***. It captures the functional aspect of the system.
   - A ***stick figure*** is used to represent an actor.
   - An ***oval*** is used to represent a use case.
   - A ***line*** is used to represent a relationship between an actor and a use case.

Requirements Elicitation

Use case Diagrams

-- represents what happens when actor interacts with a system.

-- captures functional aspect of the system.

Actor

Use Case

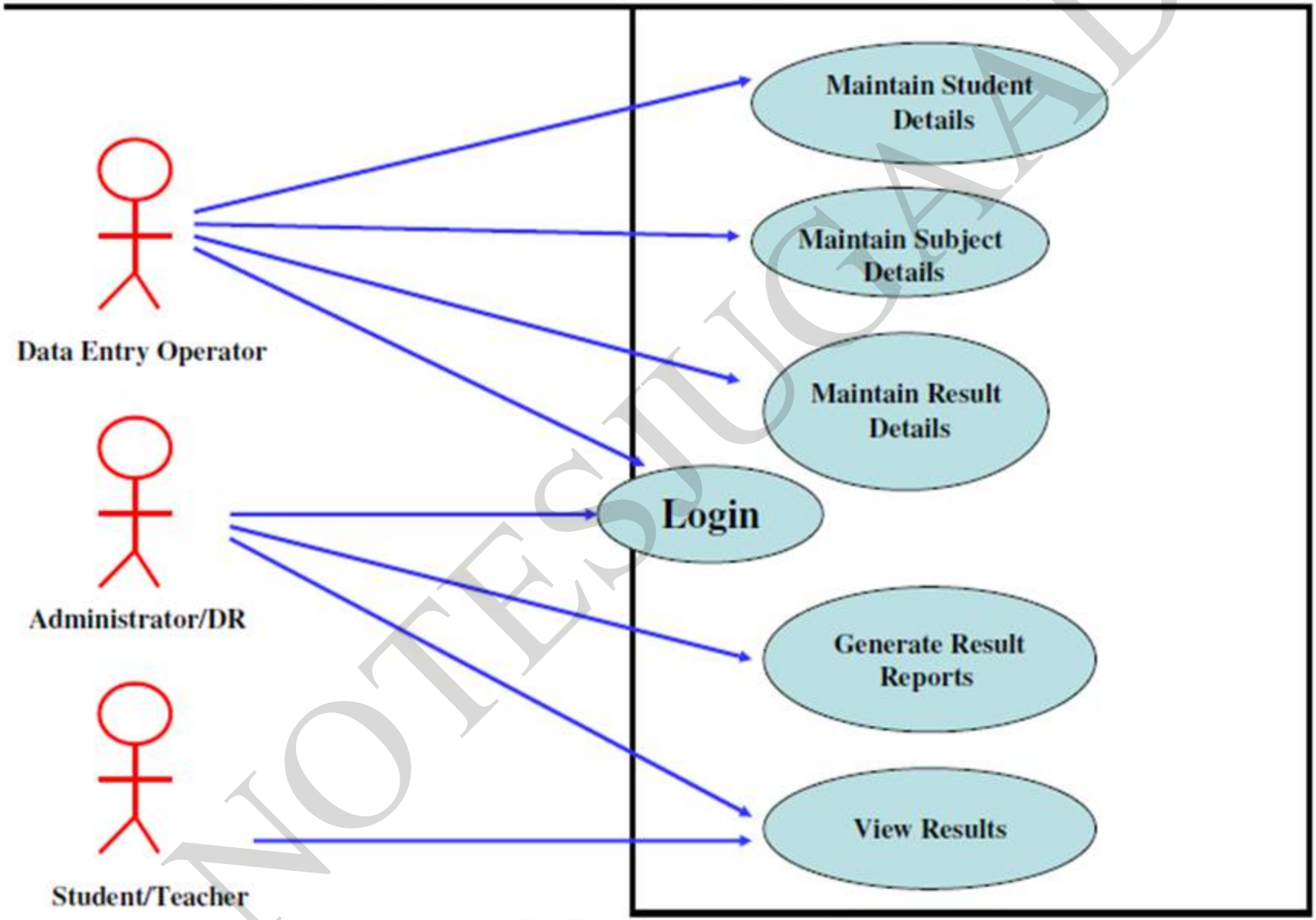
Relationship between actors and use case and/or between the use cases.

-- Actors appear outside the rectangle.

--Use cases within rectangle providing functionality.

--Relationship association is a solid line between actor & use cases.

Use case diagram for Result Management System



Requirements Elicitation

jims Greater Noida

The Template Elements for writing Use Cases as follows:

1. **Introduction:** Describe a quick background of the use case.
2. **Actors:** List the actors that interact and participate in the use cases.
3. **Pre Conditions:** Pre conditions that need to be satisfied for the use case to perform.
4. **Post Conditions:** Define the different states in which we expect the system to be in, after the use case executes.
5. **Flow of Event:**
   a) **Basic Flow:** List the primary events that will occur when this use case is executed.
   b) **Alternative Flows:** Any Subsidiary events that can occur in the use case should be separately listed. List each such event as an alternative flow. A use case can have many alternative flows as required.
6. **Special Requirements:** Business rules should be listed for basic & information flows as special requirements in the use case narration .These rules will also be used for writing test cases. Both success and failures scenarios should be described.
7. **Use Case Relationships:** For Complex systems it is recommended to document the relationships between use cases. Listing the relationships between use cases also provides a mechanism for traceability.

Requirements Elicitation

1. Maintain student Details

   Add/Modify/update students details like name, address.

2. Maintain subject Details

   Add/Modify/Update Subject information semester wise

3. Maintain Result Details

   Include entry of marks and assignment of credit points for each paper.

4. Login

   Use to Provide way to enter through user id & password.

5. Generate Result Report

   Use to print various reports

6. View Result
   (i) According to course code
   (ii) According to Enrollment number/roll number

Requirements Elicitation (Use Case)

jims Greater Noida

Login

1.1 Introduction : This use case describes how a user logs into the Result Management System.

1.2 Actors : (i) Data Entry Operator
(ii) Administrator/Deputy Registrar

1.3 Pre Conditions : None

1.4 Post Conditions : If the use case is successful, the actor is logged into the system. If not, the system state is unchanged.

Requirements Elicitation (Use Case)

jims Greater Noida

1.5 Basic Flow : This use case starts when the actor wishes to login to the Result Management system.

(i) System requests that the actor enter his/her name and password.

(ii) The actor enters his/her name & password.

(iii) System validates name & password, and if finds correct allow the actor to logs into the system.

1.6 Alternate Flows

1.6.1 Invalid name & password

If in the basic flow, the actor enters an invalid name and/or password, the system displays an error message. The actor can choose to either return to the beginning of the basic flow or cancel the login, at that point, the use case ends.

1.7 Special Requirements:

None

1.8 Use case Relationships:

None

2.Maintain student details

2.1 Introduction : Allow DEO to maintain student details. This includes adding, changing and deleting student information

2.2 Actors : DEO

2.3 Pre-Conditions: DEO must be logged onto the system before this use case begins.

Use Cases

2.4 Post-conditions : If use case is successful, student information is added/updated/deleted from the system. Otherwise, the system state is unchanged.

2.5 Basic Flow : Starts when DEO wishes to add/modify/update/delete Student information.

(i) The system requests the DEO to specify the function, he/she would like to perform (Add/update/delete)

(ii) One of the sub flow will execute after getting the information.

- If DEO selects "Add a student", "Add a student" sub flow will be executed.
- If DEO selects "update a student", "update a student" sub flow will be executed.
- If DEO selects "delete a student", "delete a student" sub flow will be executed.

2.5.1 Add a student

(i) The system requests the DEO to enter:
- Name
- Address
- Roll No
- Phone No
- Date of admission

(ii) System generates unique id

2.5.2 Update a student

(i) System requires the DEO to enter student-id.

(ii) DEO enters the student_id. The system retrieves and display the student information.

(iii) DEO makes the desired changes to the student information.

(iv) After changes, the system updates the student record with changed information.

Use Cases

2.5.3 Delete a student

(i) The system requests the DEO to specify the student-id.

(ii) DEO enters the student-id. The system retrieves and displays the student information.

(iii) The system prompts the DEO to confirm the deletion of the student.

(iv) The DEO confirms the deletion.

(v) The system marks the student record for deletion.

2.6 Alternative flows

2.6.1 Student not found

If in the update a student or delete a student sub flows, a student with specified_id does not exist, the system displays an error message .The DEO may enter a different id or cancel the operation. At this point ,Use case ends.

2.6.2 Update Cancelled

If in the update a student sub-flow, the data entry operator decides not to update the student information, the update is cancelled and the basic flow is restarted at the begin.

2.6.3 Delete cancelled

If in the delete a student sub flows, DEO decides not to delete student record ,the delete is cancelled and the basic flow is re-started at the beginning.

2.7 Special requirements

None

2.8 Use case relationships

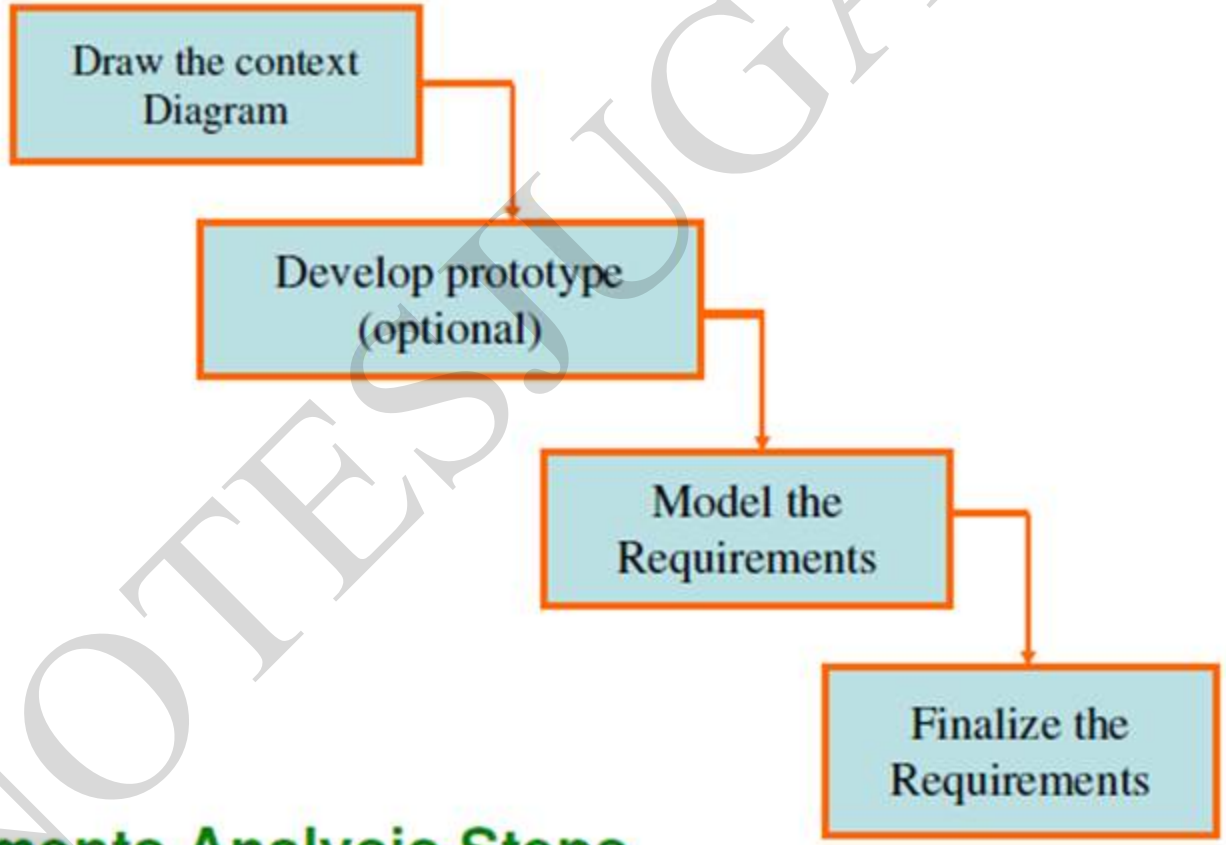
None

REQUIREMENT ANALYSIS

Requirements Analysis

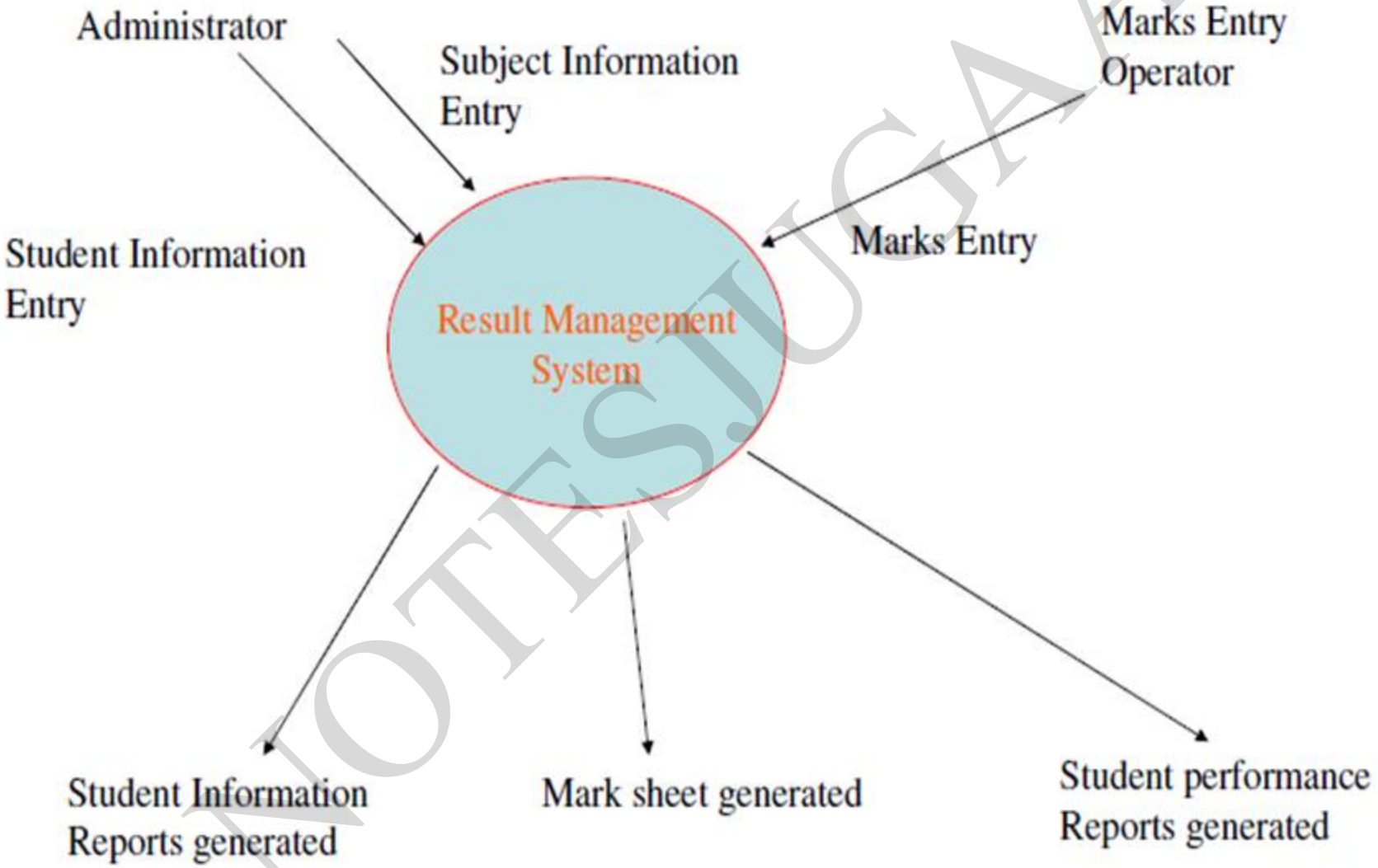
We analyze, refine and scrutinize requirements to make consistent & unambiguous requirements.

Steps



Requirements Analysis Steps

Requirements Analysis



Data Flow Diagram

Data flow diagram is graphical representation of **flow of data** in an information system. It is capable of depicting ***incoming data flow, outgoing data flow and stored data***.

The four different components of DFD Model are as follows:

Entities - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.

Process - Activities and action taken on the data are represented by Circle or Round-edged rectangles.

Data Storage - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.

Data Flow - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

0 – Level DFD

It is also known as ***fundamental system model***, which represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows Fig 1.

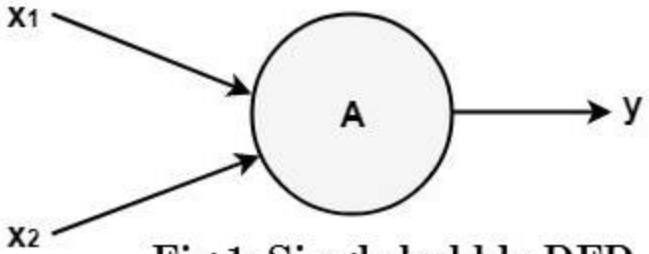


Fig 1: Single bubble DFD

The Level-0 DFD, also called ***context diagram*** in which the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed Fig 2.

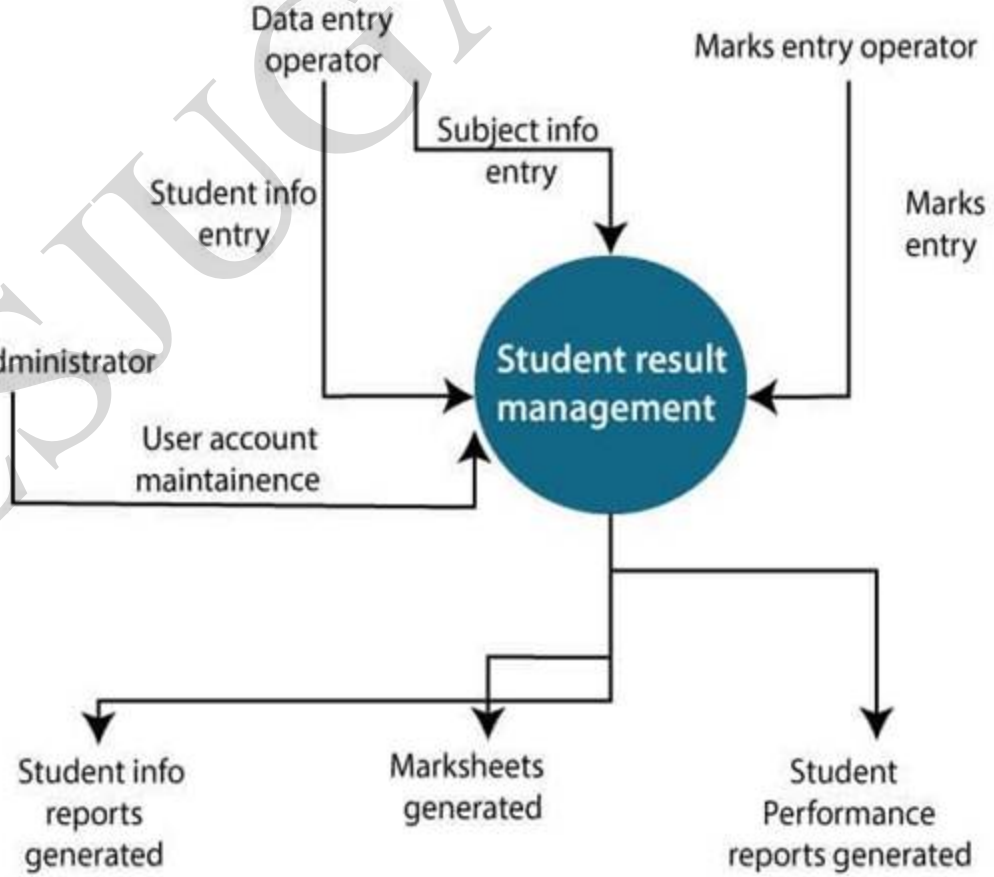


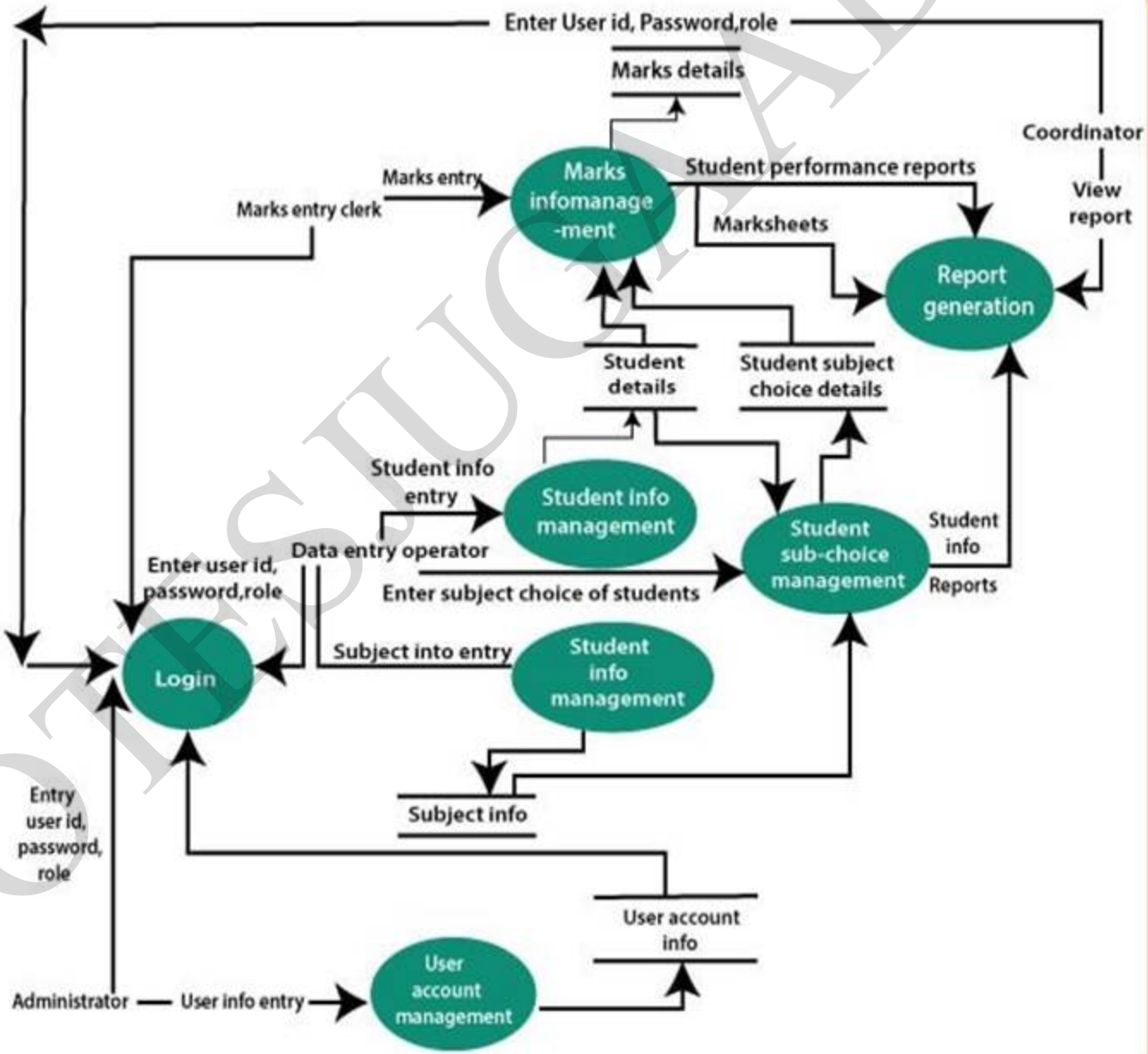
Fig 2: DFD of Result Mgmt. System

Levels of DFD

1 – Level DFD

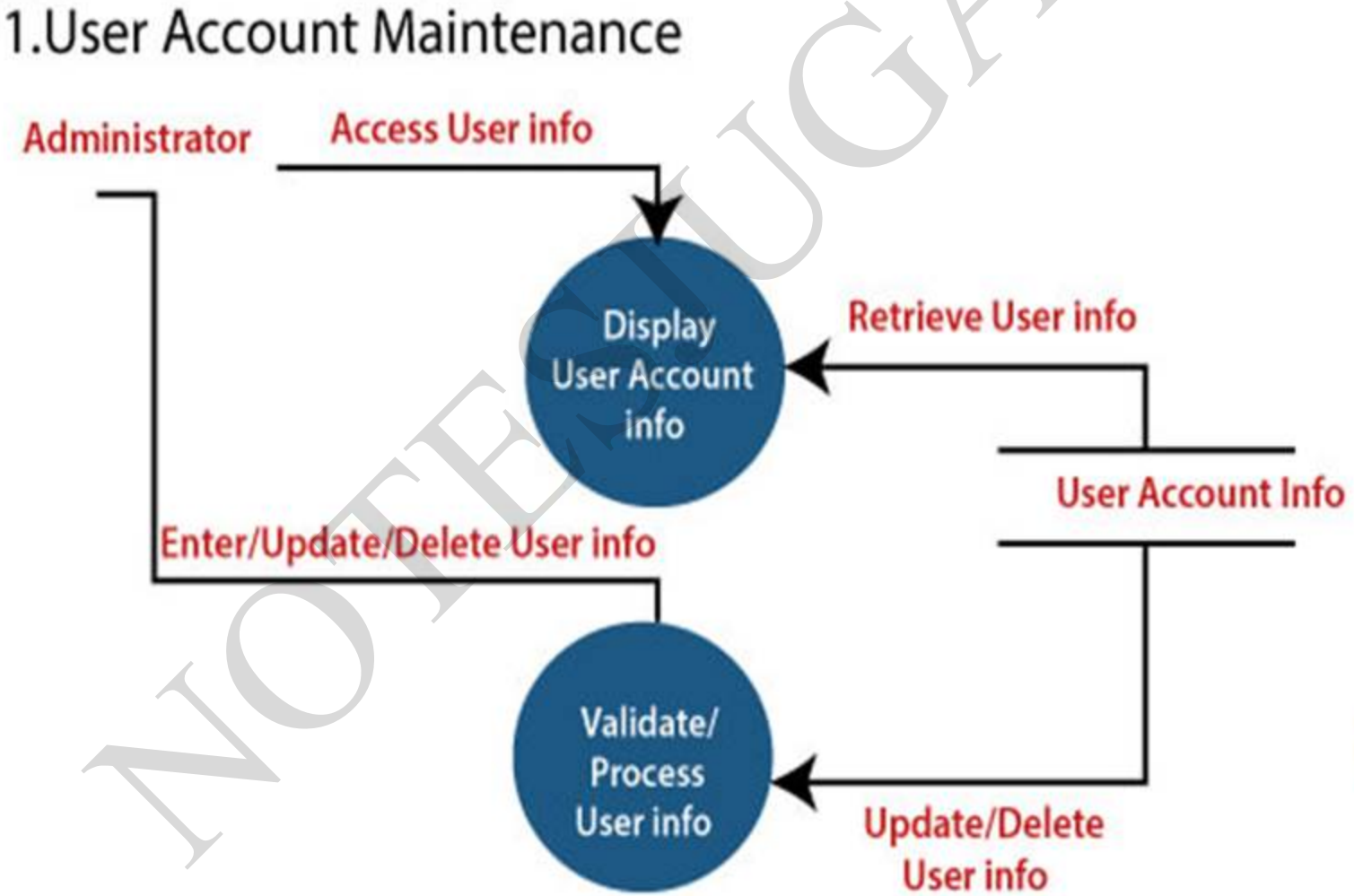
In 1-level DFD, a context diagram is decomposed into multiple bubbles / processes.

In this level, the main objectives of the system and breakdown of the high-level process of 0-level DFD into sub-processes are done.



2 – Level DFD

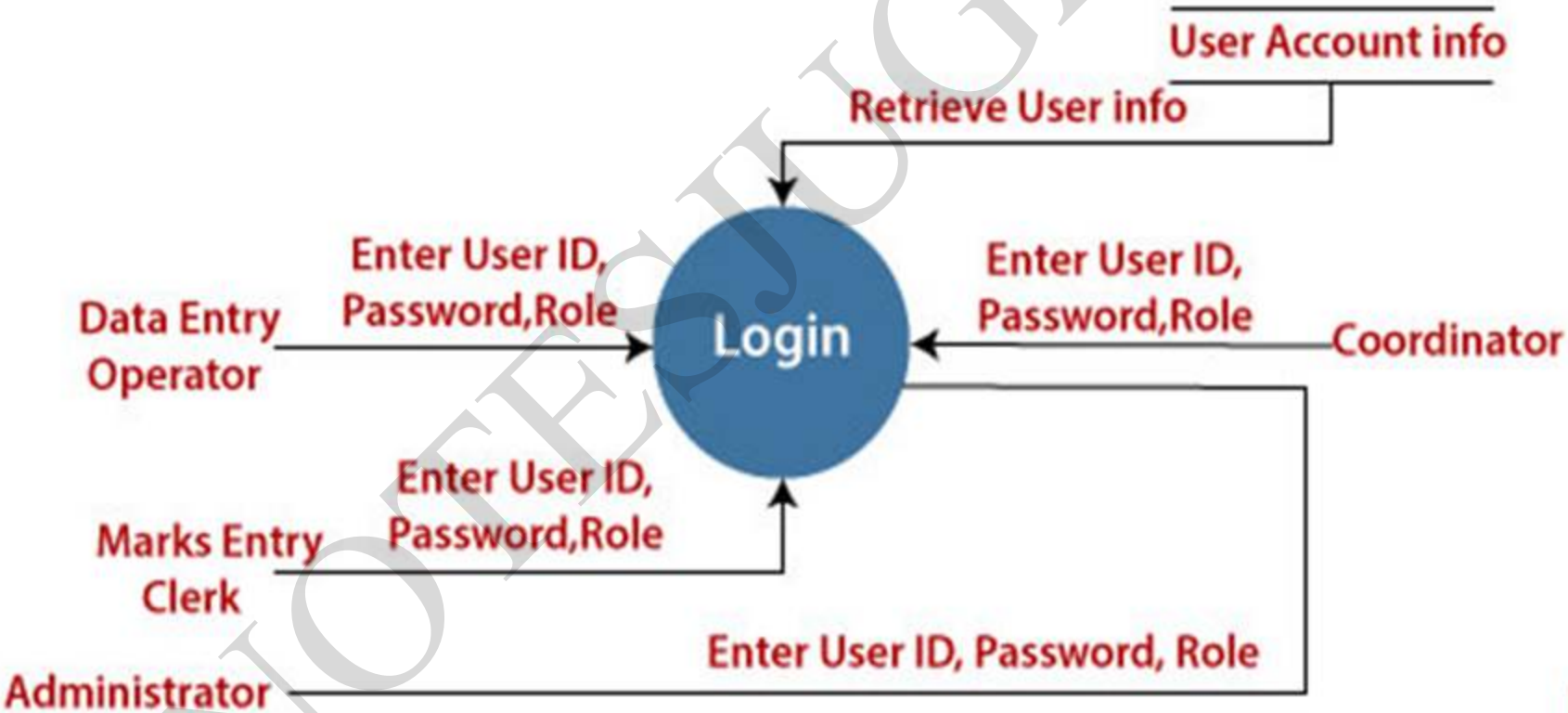
2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.



2 – Level DFD

2. Login

The level 2 DFD of this process is given below:



Data Dictionaries

A data dictionary is a file or a set of files that includes a database's metadata. The data dictionary hold records about other objects in the database, such as *data ownership, data relationships to other objects, and other data*.

The data dictionary, includes information about the following:

1. **Name of the data item:** self explanatory
2. **Aliases:** include other names by which this data item is called DEO for Data Entry Operator and DR for Deputy Registrar
3. **Description/Purpose:** is a textual description of what the data item is used for or why it exists.
4. **Related Data Items:** capture relationships between data items e.g., *total_marks* must always equal to *internal_marks plus external_marks*.
5. **Range of Values:** records all possible values, e.g. *total marks must be positive and between 0 to 100.*
6. **Data Structure Definition/Forms:** Data flows capture the name of processes that generate or receive the data items. If the data item is primitive, then data structure form captures the physical structures of the data item. If the data is itself a data aggregate, then data structure form capture the composition of the data items in terms of other data items.

Data Dictionaries

jims Greater Noida

Notation	Meaning
x= a+b	x consists of data element a & b
x={a/b}	x consists of either a or b
x=(a)	x consists of an optional data element a
x= y{a}	x consists of y or more occurrences
x={a}z	x consists of z or fewer occurrences of a
x=y{a}z	x consists of between y & z occurrences of a{

Requirements Analysis ENTITY RELATIONSHIP DIAGRAM

- ER-modeling is a data modeling method used in software engineering to produce a ***conceptual data model of an information*** system. Diagrams created using this *ER-modeling* method are called ***Entity-Relationship Diagrams*** or ER diagrams or ERDs.

Purpose of ERD

1. The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.
2. The ERD serves as a documentation tool.
3. Finally, the ERD is used to connect the ***logical structure of the database to users***. In particular, the ERD effectively communicates the logic of the database to users.

Components of ERD

1. Entity
2. Attributes
3. Relationships

Requirements Analysis ENTITY RELATIONSHIP DIAGRAM

1. **Entity:** An entity is a **real-world object**. It is denoted as a **rectangle** in an ER diagram. For example, in a school database, ***students, teachers, classes, and courses offered*** can be treated as entities. All these entities have some attributes or properties that give them their identity.

Entity Set: An entity set is a **collection of related types of entities**. An entity set may include entities with attribute sharing similar values. For example, a Student set may contain all the students of a school; likewise, a Teacher set may include all the teachers of a school from all departments. Entity set need not be disjoint.

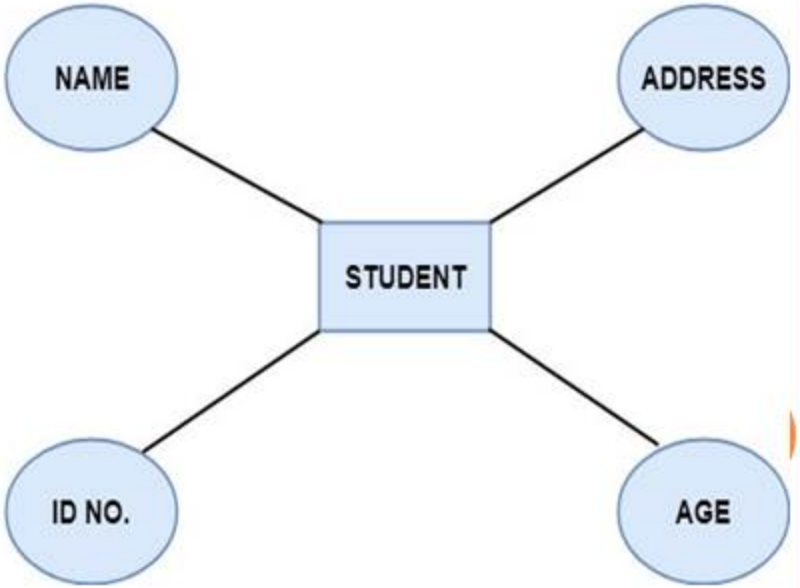
2. **Attributes:** Entities are denoted ***utilizing their properties***, known as attributes. All attributes have values. For example, a ***student entity may have name, class, and age as attributes***.

There exists a domain or range of values that can be assigned to attributes. For example,

- ✓ a student's name cannot be a numeric value. It has to be alphabetic.
- ✓ A student's age cannot be negative.

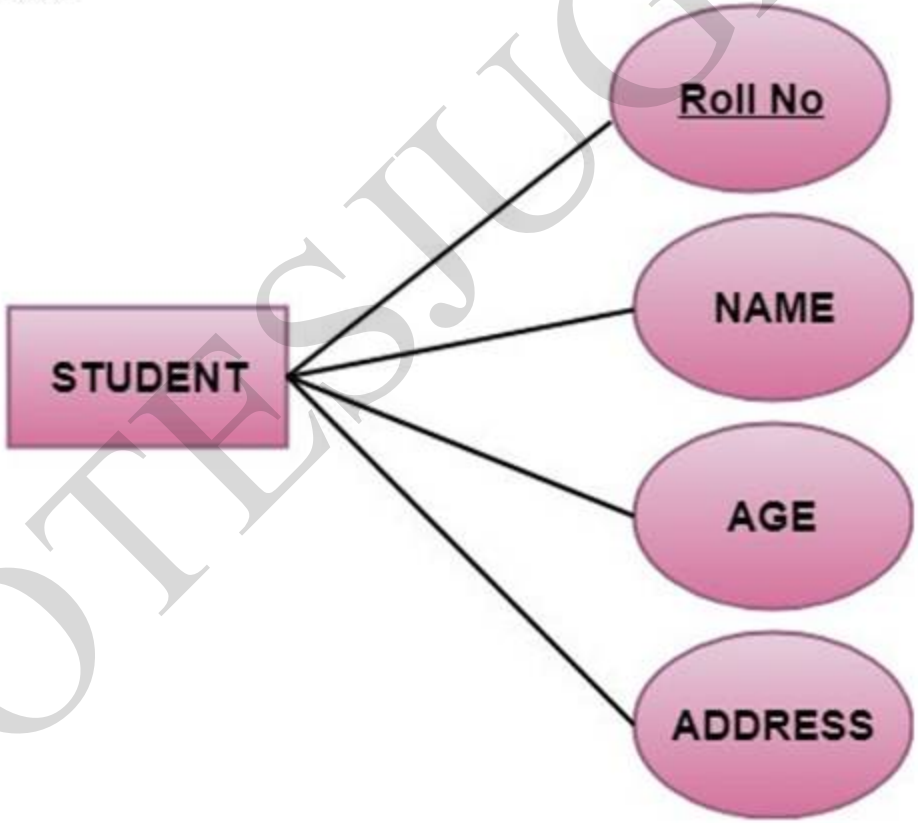
There are four types of Attributes:

a) Key attribute
b) Composite attribute
c) Single-valued attribute
d) Multi-valued attribute
e) Derived attribute



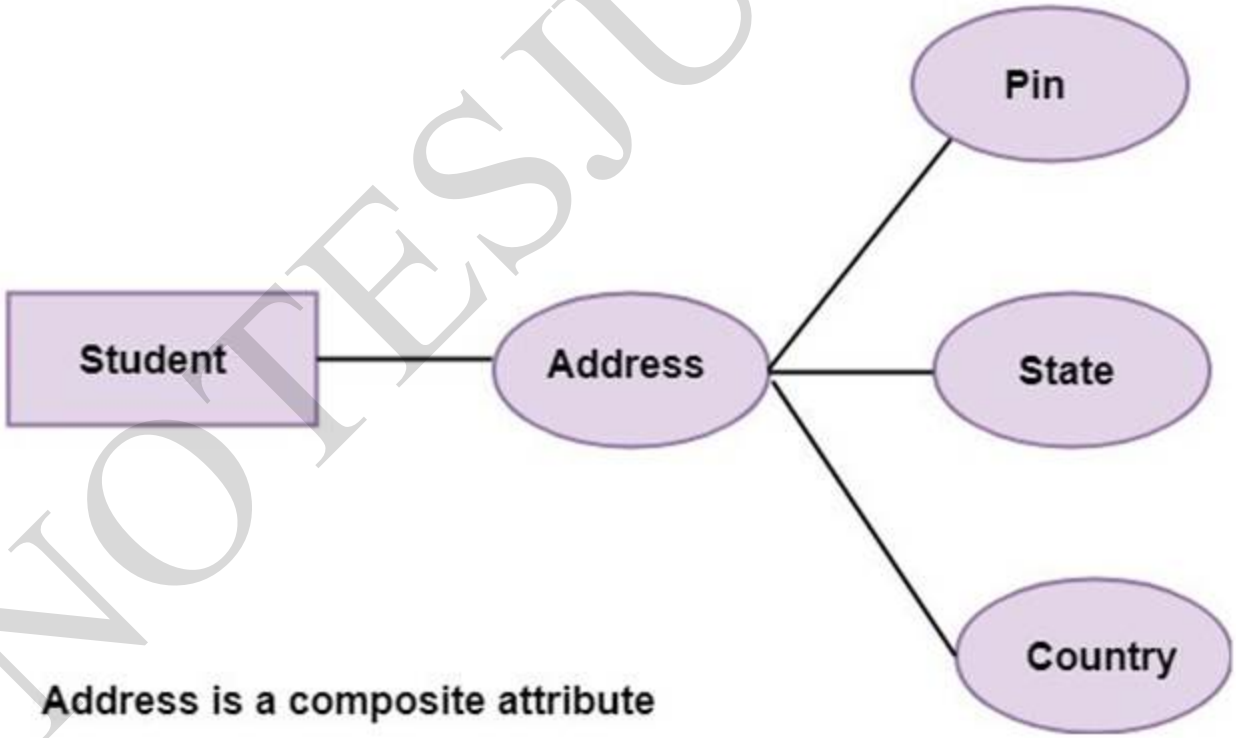
TYPES OF ATTRIBUTES

a) **Key attribute:** Key is an attribute or ***collection of attributes that uniquely identifies an entity*** among the entity set. For example, the roll_number of a student makes him identifiable among students.



TYPES OF ATTRIBUTES

b) **Composite attribute:** An attribute that is a ***combination of other attributes*** is called a composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other characteristics such as pin code, state, country.

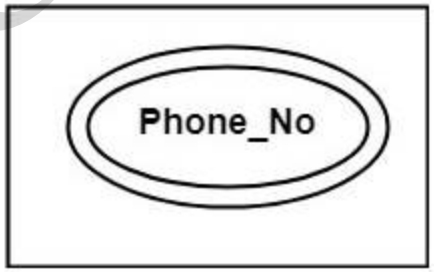


Address is a composite attribute

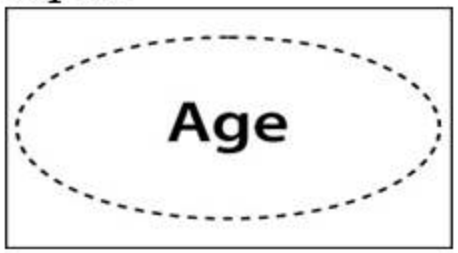
TYPES OF ATTRIBUTES

c) **Single-valued Attribute:** Single-valued attribute contain a single value. For example, Social_Security_Number.

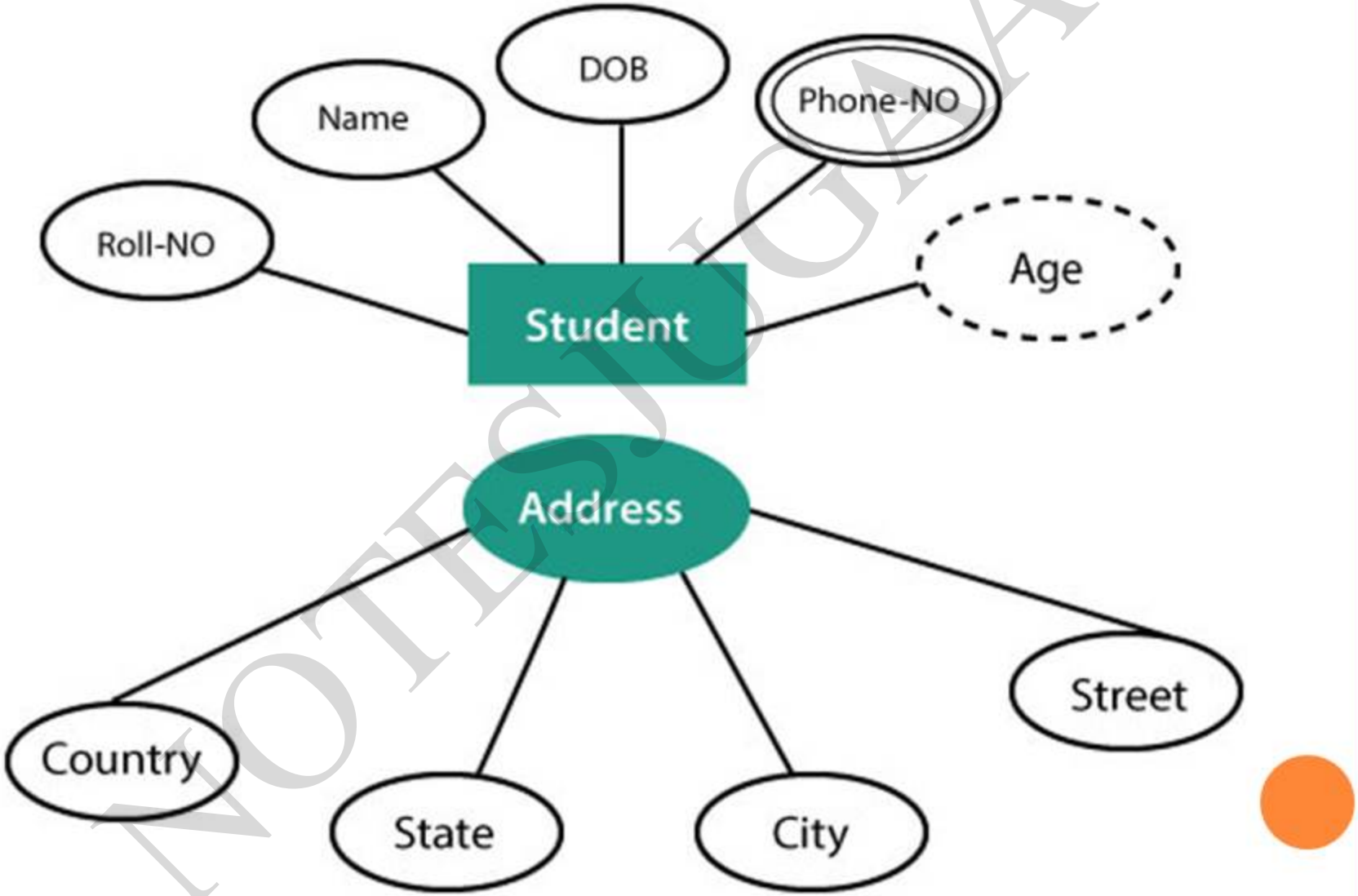
d) **Multi-valued Attribute:** If an attribute can have more than one value, it is known as a multi-valued attribute. Multi-valued attributes are depicted by the ***double ellipse***. For example, a person can have more than one phone number, email-address, etc.



e) **Derived Attribute:** Derived attributes are the attribute that ***does not exist in the physical database***, but their ***values are derived from other attributes present in the database***. For example, age can be derived from date_of_birth. In the ER diagram, Derived attributes are depicted by the dashed ellipse.



The Complete entity type Student with its attributes can be represented as:



3. **Relationship:** The association among entities is known as relationship. Relationships are represented by the ***diamond-shaped box***.

Fig: Relationships in ERD

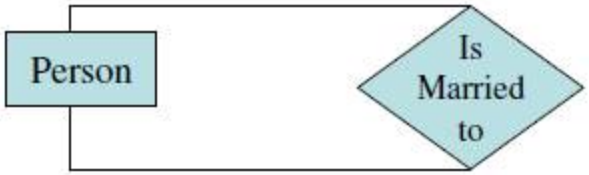
Degree of a Relationship Set:

The number of participating entities in a relationship describes the degree of the relationship.

The three most common relationships in E-R models are:

a) **Unary (degree1):** This is also called recursive relationships. It is a relationship between the instances of one entity type. *For example, one person is married to only one person.*
b) **Binary (degree2):** It is a relationship between the instances of two entity types. *For example, the Teacher teaches the subject.*
c) **Ternary (degree3):** It is a relationship amongst instances of three entity types. *For Example: TEACHER, STUDENT, and SUBJECT*

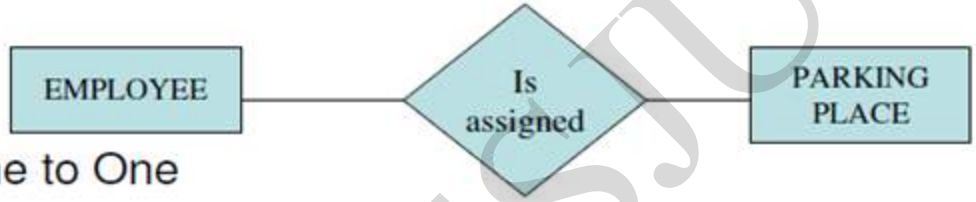
Unary relationship



One to One

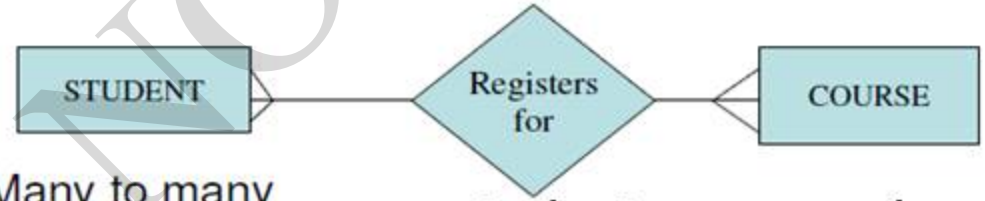
One to many

Binary Relationship



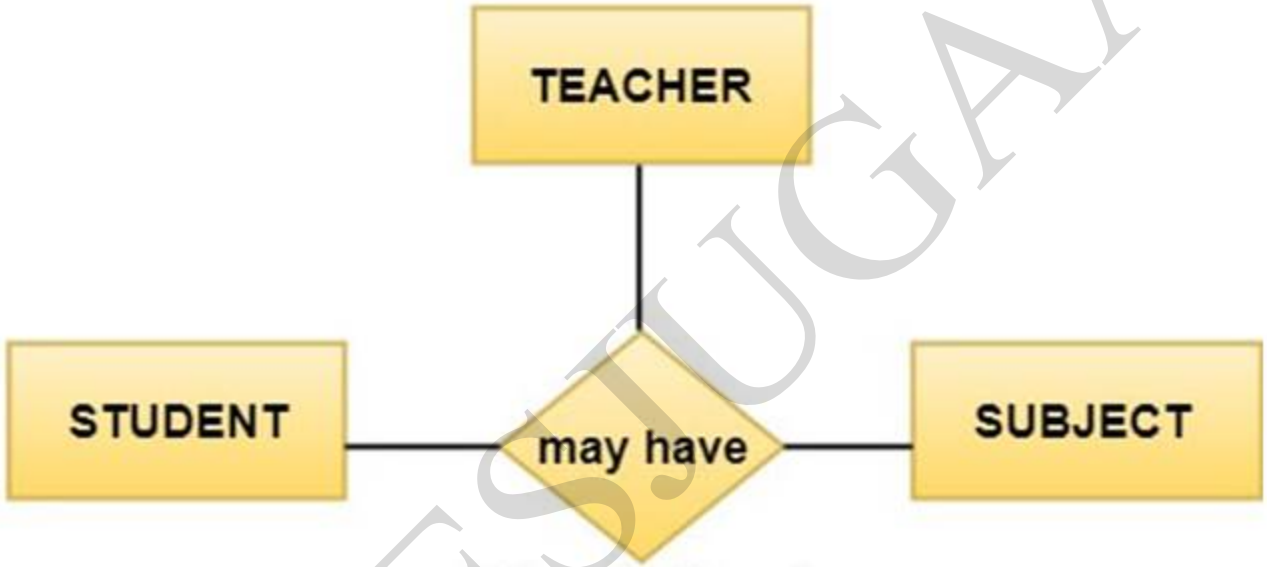
One to One

One to many



Many to many

Ternary Relationship



(Many to Many)

Fig: Ternary Relationship

Requirements Documentation

Requirements Documentation

This is the way of representing requirements in a consistent format

SRS serves many purpose depending upon who is writing it.

-- written by customer
-- written by developer

Serves as contract between customer & developer.

A **System Requirements Specification (SRS)** (also known as a Software Requirements Specification) is a document or set of documentation that describes the features and behavior of a system or software application.

PAYROLL SYSTEM REQUIREMENTS SAMPLE

<insert your logo> **REQUIREMENTS SPECIFICATION for**

HR and Payroll Software
for <Organisation name, City>

Reference: <insert reference number>
Issued on: <insert date>
Issued by: <insert contact name>

Ref	Functional Requirements	C	W	Vendor Response	S	WS
6.6	**Performance appraisal**					
6.6.1	Support multiple types of appraisals and performance assessments eg: annual, 6 monthly, self appraisal, 180/360 degree, peer-to-peer, induction	N	0		0	0
6.6.2	Support multiple appraisals of each employee, determined by appraiser or organisational needs	N	0		0	0
6.6.3	Flexible next appraisal date eg 6 monthly, annual, project completion	N	0		0	0
6.6.4	Set appraisals for individual employees or groups / departments	N	0		0	0
6.6.5	Configurable templates for appraisals including: self assessment, competencies, development plans, 360 degree evaluations	N	0		0	0
6.6.6	Configurable appraisal form language / terminology, evaluation formats	N	0		0	0
6.6.7	Support different appraisal plans and assessments for different roles, responsibilities, groups of employees	N	0		0	0
6.6.8	Configurable appraisal process workflows, appraisal scoring, approval steps	N	0		0	0
6.6.9	Unlimited appraisal criteria, performance measures, objectives (qualitative and quantitative) and competencies	N	0		0	0
6.6.10	Automatic generation of forms, schedules and issue to recipients	N	0		0	0
6.6.11	Choice of appraisers / appraisees including:	N	0		0	0
6.6.11.1	- automatic assignment of appraisers (managers) and appraisees (employees)	N	0		0	0
6.6.11.2	- manual (HR / management) choice	N	0		0	0
6.6.11.3	- appraisee (employee) choice of feedback from subordinates, peers, seniors, external contacts, other raters (for 360 degree appraisals)	N	0		0	0
6.6.11.4	- nomination of employees for appraisal	N	0		0	0
6.6.12	Appraiser and appraisee access to historical appraisals, goals, targets and current appraisal forms	N	0		0	0
	Initial appraiser and appraisee discussion and performance points input					

Cover | TOC | Requirements Specification | Evaluation Scores

C - to note the function is currently used. The default is N(o), change to Y(es) as appropriate.
W - for importance weighting of a future requirement. Various numerical ranges can be used - an easy-to-use range is:

0 - not needed
1 - a nice to have requirement
2 - a desirable requirement
3 - an essential requirement

Requirements Documentation

Nature of SRS

Basic Issues

- Functionality
- External Interfaces
- Performance
- Attributes
- Design constraints imposed on an Implementation

Requirements Documentation

jims Greater Noida

SRS Should

-- Correctly define all requirements

-- not describe any design details

-- not impose any additional constraints

Characteristics of a good SRS

An SRS Should be

- ✓ Correct
- ✓ Unambiguous
- ✓ Complete
- ✓ Consistent
- ✓ Ranked for important and/or stability
- ✓ Verifiable
- ✓ Modifiable
- ✓ Traceable

Requirements Documentation

Correct

An SRS is correct if and only if every requirement stated therein is one that the software shall meet.

Unambiguous

An SRS is unambiguous if and only if, every requirement stated therein has only one interpretation.

Complete

An SRS is complete if and only if, it includes the following elements

(i) All significant requirements, whether related to functionality, performance, design constraints, attributes or external interfaces.

(ii) Responses to both valid & invalid inputs.

(iii) Full Label and references to all figures, tables and diagrams in the SRS and definition of all terms and units of measure.

Consistent

An SRS is consistent if and only if, no subset of individual requirements described in it conflict.

Ranked for importance and/or Stability

If an identifier is attached to every requirement to indicate either the importance or stability of that particular requirement.

Verifiable

An SRS is verifiable, if and only if, every requirement stated therein is verifiable.

Modifiable

An SRS is modifiable, if and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining structure and style.

Traceable

An SRS is traceable, if the origin of each of the requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.

ORGANIZATION OF SRS

श्री हर्ष अत्रि (सहायक आचार्य)

jims Greater Noida

Organization of the SRS

IEEE has published guidelines and standards to organize an SRS.
First two sections are same. The specific tailoring occurs in section-3.

1. Introduction
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Definition, Acronyms and abbreviations
    - 1.4 References
    - 1.5 Overview

Requirements Documentation

jims Greater Noida

2. The Overall Description

- 2.1 Product Perspective
  - 2.1.1 System Interfaces
  - 2.1.2 Interfaces
  - 2.1.3 Hardware Interfaces
  - 2.1.4 Software Interfaces
  - 2.1.5 Communication Interfaces
  - 2.1.6 Memory Constraints
  - 2.1.7 Operations
  - 2.1.8 Site Adaptation Requirements
- 2.2 Product Functions
- 2.3 User Characteristics
- 2.4 Constraints
- 2.5 Assumptions for dependencies
- 2.6 Apportioning of requirements

jims Greater Noida

3. Specific Requirements

3.1 External Interfaces
3.2 Functions
3.3 Performance requirements
3.4 Logical database requirements
3.5 Design Constraints
3.6 Software System attributes
3.7 Organization of specific requirements
3.8 Additional Comments.

REQUIREMENTS VALIDATION

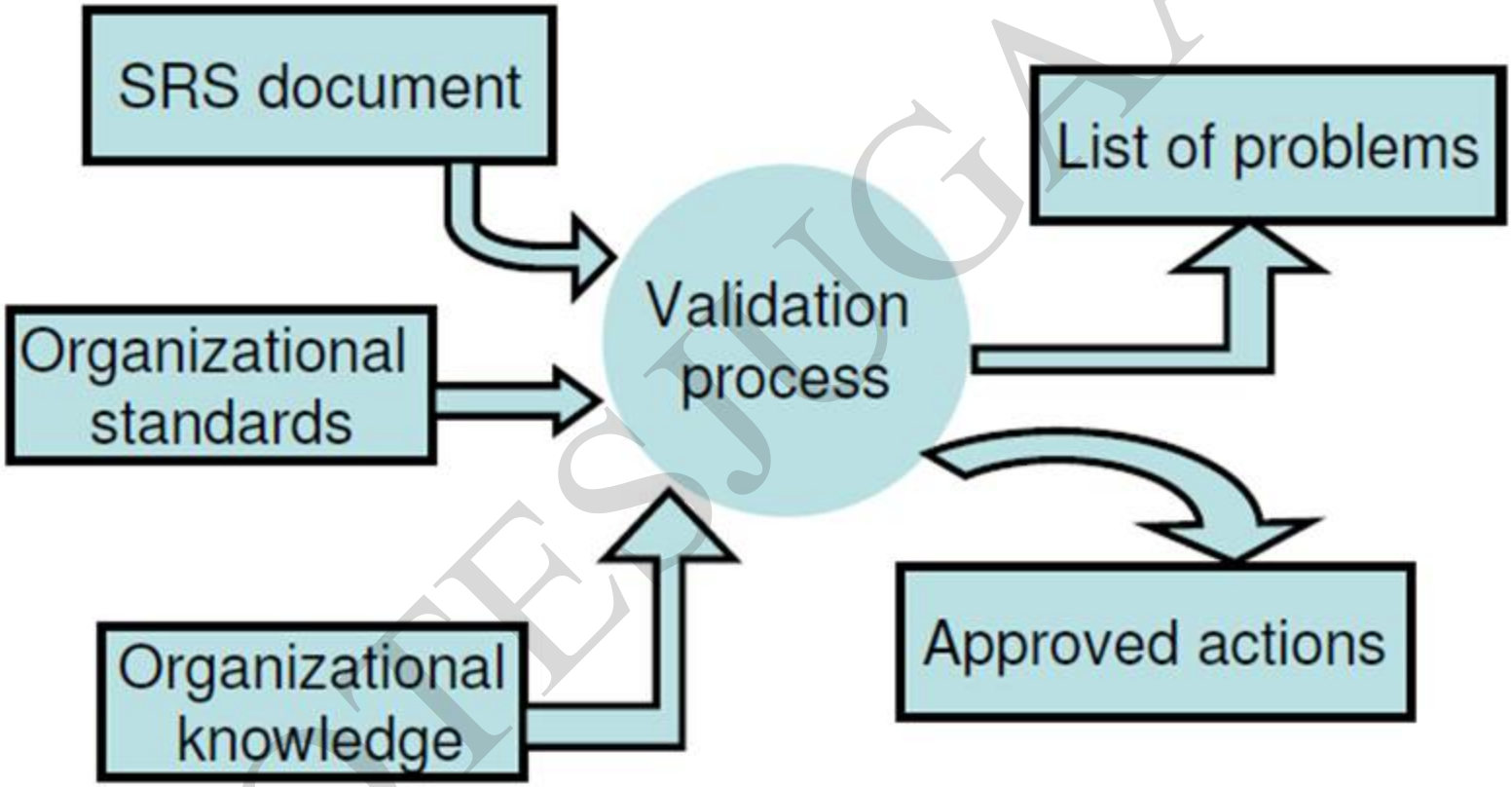
श्री हर्ष अत्रि (सहायक आचार्य)

Requirements Validation

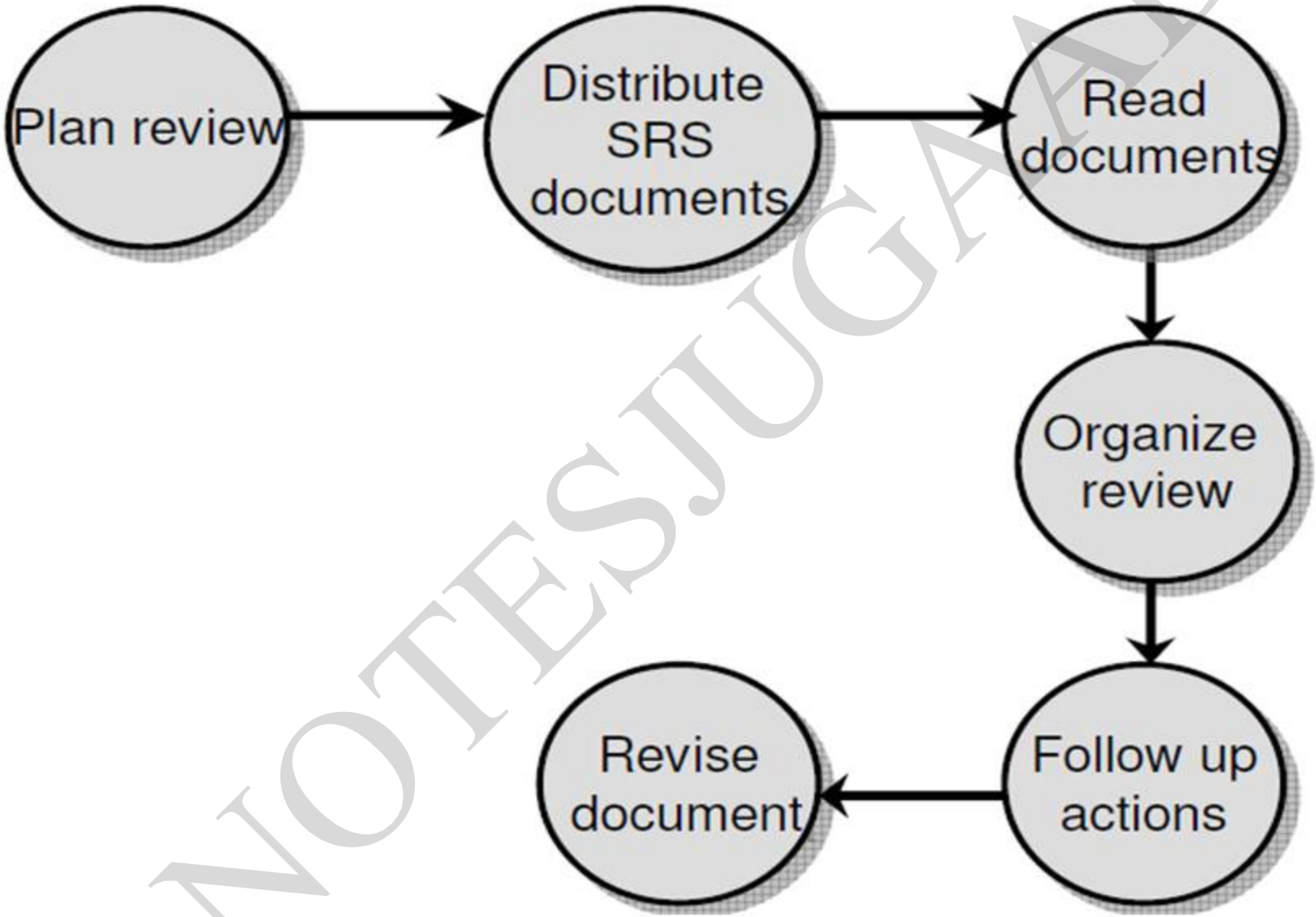
Check the document for:

- ✓ Completeness & consistency
- ✓ Conformance to standards
- ✓ Requirements conflicts
- ✓ Technical errors
- ✓ Ambiguous requirements

Requirements Validation



Requirements Review Process



Requirements Validation

jims Greater Noida

Problem actions

- Requirements clarification
- Missing information
  - find this information from stakeholders
- Requirements conflicts
  - Stakeholders must negotiate to resolve this conflict
- Unrealistic requirements
  - Stakeholders must be consulted
- Security issues
  - Review the system in accordance to security standards

Review Checklists

- ✓ Redundancy
- ✓ Completeness
- ✓ Ambiguity
- ✓ Consistency
- ✓ Organization
- ✓ Conformance
- ✓ Traceability

REFERENCE

Book:

Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007

Website Link:

https://www.geeksforgeeks.org/software-engineering-requirements-elicitation/

https://www.tutorialspoint.com/software_engineering/software_requirements.htm

https://www.slideshare.net/AdilAslam4/data-flow-diagram-in-software-engineering?from_action=save

https://www.javatpoint.com/software-engineering-data-flow-diagrams

https://www.inflectra.com/ideas/topic/requirements-definition.aspx

https://www.axia-consulting.co.uk/html/human_resource_sample.html