AMOGH University (Education System) needs their own website and mobile app for better connectivity with their stakeholders. Suggest a suitable SDLC process model, explain the same in detail, and give the justification for the same.

To address the needs of AMOGH University for a website and mobile app, here's a suitable SDLC process model, explanation, and justification:

Suitable SDLC Process Model: Agile Unified Process (AUP)

Explanation of AUP:

The Agile Unified Process (AUP) is an agile development approach that uses elements of the Unified Process (UP). AUP adopts a "serial in the large" and "iterative in the small" philosophy.

AUP incorporates the classic UP phased activities:

- **Inception**: Communication and planning activities, collaborating with stakeholders to identify business requirements and propose a rough architecture.
- **Elaboration**: Refines the architecture and plans project iterations.
- **Construction**: Develops and tests software increments.
- **Transition**: Delivers the software to end-users.

AUP emphasizes customer communication, streamlined methods, and an iterative and incremental process flow. UML is used for modeling during requirements and design. AUP integrates agile modeling philosophy and can adapt to the needs of the agile team.

How AUP Addresses AMOGH University's Needs:

- **Stakeholder Involvement:** AUP recognizes the importance of customer communication. In the context of AMOGH University, stakeholders may include students, faculty, administrative staff, and alumni. Involving these stakeholders ensures the website and mobile app meet their needs.
- **Iterative and Incremental Development:** AUP's iterative nature allows AMOGH University to release the website and mobile app in increments, gathering feedback and making necessary adjustments along the way. This is particularly useful when requirements are not fully defined at the outset.
- **Use Case Driven:** AUP uses use cases to describe the customer's view of the system. The software team can have each stakeholder write use cases describing how they will use the software. For AMOGH University, use cases could include students registering for courses, faculty posting announcements, or alumni making donations.
- **Architecture-Centric:** AUP emphasizes the role of software architecture. The Unified Process helps the architect focus on understandability, resilience to

future changes, and reuse. This ensures the website and mobile app are built on a solid foundation that can evolve with the university's needs.

- **Flexibility and Adaptability:** AUP can be adapted to the specific needs of the project and team. This is important for AMOGH University, as the development team may need to adjust their approach based on feedback, changing requirements, or technical challenges.
- **Agile Modeling:** Agile modeling (AM) suggests core and supplementary modeling principles including modelling with a purpose and adapting locally.
- **Testing:** AUP incorporates testing throughout the development lifecycle. The test team may have conflicting arguments about the quality of the product while analyzing the requirements before delivery. The product may fail to meet the expectations of the users. Therefore, it is essential to have an unambiguous representation of the requirements and have it made available in a centralized place so that all the stakeholders have the same interpretation of the requirements.

Justification for Choosing AUP:

- **Addresses Complex Requirements:** AMOGH University's website and mobile app will likely have a wide range of features and functionality. AUP provides a structured approach for managing this complexity.
- **Promotes Collaboration:** AUP emphasizes customer communication and collaboration. It is essential to collaborate well internally and communicate well externally to form a self-directed team.
- **Supports Evolutionary Development:** The website and mobile app will need to evolve over time to meet changing needs. AUP's iterative and incremental nature supports this evolution.
- **Reduces Risk:** By delivering the website and mobile app in increments, AMOGH University can reduce the risk of delivering a final product that does not meet stakeholder needs.
- **UML Integration:** UML is a standard language for writing software blueprints. UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system.

AUP is a suitable choice for AMOGH University because it balances the need for structure and planning with the flexibility and adaptability of agile development. It ensures stakeholder involvement, promotes collaboration, and supports the evolutionary development of the website and mobile app.

AMOGH University has decided to complete the automation of their entire education system, initially with all the basic processes of academic and non academic departments within six months of time, with reference to the following BASIC processes Admission management system, Examination management system, Placement management system, Teaching- Learning processes, Library management system, and Finance management system. Also university is planning to extend the automation for all other processes like Purchase management system, HR management system, Sports and Cultural management system, Infrastructure management system, etc in near future. 1. Suggest the process model which can be used to design and develope the online education system with proper jutification. 2. Identify different stakeholders for the proposed system with their roles and responsibilities. 3. Explain the identified process model in detail with respect to development of online education system.

To automate AMOGH University's education system, the following process model, stakeholder identification, and explanations are recommended based on the sources:

1. Suggested Process Model: Incremental Process Model

- The **incremental process model** is recommended because it allows for the delivery of functional software in increments, which aligns well with AMOGH University's phased automation approach.
- The university aims to automate basic academic and non-academic departments within six months and then extend automation to other processes later. The incremental model supports this staged rollout, providing value early and continuously.
- The incremental model is an **evolutionary process model**. This means that each increment builds upon the previous ones, allowing the system to grow and adapt over time. This is important for AMOGH University, as their needs and priorities may change as they move through the automation process.
- The model allows **incorporating changes and feedback** at the end of each increment before moving to the next one.
- **Each increment can be tested and validated** independently, ensuring that the system meets the university's requirements and quality standards.
- The incremental model's iterative nature allows for continuous improvement and refinement of the system.

Justification:

- **Phased Delivery:** The incremental model allows AMOGH University to deliver functional modules of the online education system in stages. This allows the university to realize the benefits of automation early on while continuing to develop and integrate additional modules.

- **Risk Mitigation:** By delivering the system in increments, AMOGH University can identify and address potential problems early in the development process. This reduces the risk of delivering a final product that does not meet the university's needs.
- **Flexibility:** The incremental model is flexible and adaptable to changing requirements. This is important for AMOGH University, as their needs may evolve as they move through the automation process.
- **Stakeholder Involvement:** The incremental model allows for continuous stakeholder involvement throughout the development process. This ensures that the system meets the needs of all stakeholders, including students, faculty, and administrative staff.

2. Stakeholders, Roles, and Responsibilities:

- **Students:**
    - *Role:* Users of the online education system.
    - *Responsibilities:* Provide feedback on the system's usability and functionality, participate in user acceptance testing, and use the system for learning and academic activities.
- **Faculty:**
    - *Role:* Instructors and content creators for the online education system.
    - *Responsibilities:* Develop and deliver online courses, provide feedback on the system's features and functionality, and use the system to manage their courses and interact with students.
- **Administrative Staff:**
    - *Role:* System administrators and support staff.
    - *Responsibilities:* Manage the system's infrastructure, provide technical support to users, and ensure the system's security and reliability.
- **University Management:**
    - *Role:* Project sponsors and decision-makers.
    - *Responsibilities:* Provide resources and support for the project, define the project's goals and objectives, and make decisions about the system's scope and functionality.
- **IT Department:**
    - *Role:* System developers and integrators.

    - *Responsibilities:* Design, develop, and test the online education system, integrate it with existing university systems, and provide ongoing maintenance and support.

- **Software Quality Assurance Team:**
    - *Role:* Ensure the quality of the system.
    - *Responsibilities:* Plan and execute system testing, track and analyze defects, and ensure that the system meets the university's quality standards.

3. Explanation of the Incremental Process Model for Online Education System Development:

- **Increment 1: Admission Management System**
    - *Communication:* Gather requirements from the admissions department, IT staff, and prospective students.
    - *Planning:* Develop a plan for designing, developing, and testing the admission management system.
    - *Modeling:* Create models of the system's data, processes, and user interfaces.
    - *Construction:* Develop and test the admission management system.
    - *Deployment:* Deploy the admission management system and gather feedback from users.
- **Increment 2: Examination Management System**
    - *Communication:* Gather requirements from the examination department, faculty, and students.
    - *Planning:* Develop a plan for designing, developing, and testing the examination management system.
    - *Modeling:* Create models of the system's data, processes, and user interfaces.
    - *Construction:* Develop and test the examination management system.
    - *Deployment:* Deploy the examination management system and gather feedback from users.
- **Increment 3: Placement Management System**
    - *Communication:* Gather requirements from the placement department, students, and employers.

    - *Planning:* Develop a plan for designing, developing, and testing the placement management system.
    - *Modeling:* Create models of the system's data, processes, and user interfaces.
    - *Construction:* Develop and test the placement management system.
    - *Deployment:* Deploy the placement management system and gather feedback from users.
- **Increment 4: Teaching-Learning Processes**
    - *Communication:* Gather requirements from faculty and students.
    - *Planning:* Develop a plan for designing, developing, and testing the teaching-learning processes.
    - *Modeling:* Create models of the system's data, processes, and user interfaces.
    - *Construction:* Develop and test the teaching-learning processes.
    - *Deployment:* Deploy the teaching-learning processes and gather feedback from users.
- **Increment 5: Library Management System**
    - *Communication:* Gather requirements from the library staff and students.
    - *Planning:* Develop a plan for designing, developing, and testing the library management system.
    - *Modeling:* Create models of the system's data, processes, and user interfaces.
    - *Construction:* Develop and test the library management system.
    - *Deployment:* Deploy the library management system and gather feedback from users.
- **Increment 6: Finance Management System**
    - *Communication:* Gather requirements from the finance department and administrative staff.
    - *Planning:* Develop a plan for designing, developing, and testing the finance management system.
    - *Modeling:* Create models of the system's data, processes, and user interfaces.
    - *Construction:* Develop and test the finance management system.

- *Deployment:* Deploy the finance management system and gather feedback from users.

This incremental approach allows AMOGH University to deliver a comprehensive online education system in a phased manner, ensuring that each module meets the needs of its stakeholders and integrates seamlessly with the existing university infrastructure.

State the significance of requirement analysis. Elaborate elements of requirement analysis with neat and clean diagram.

The significance of requirements analysis lies in ensuring that a software system meets the actual needs of its users and stakeholders. It is crucial to understand what the customer wants before designing and building a system. **Designing an elegant program that solves the wrong problem is of no use**.

Elements of requirements analysis include:

- **Understanding user interaction:** Identifying what user interaction occurs in a particular circumstance.
- **Defining objects:** Determining what objects the system manipulates.
- **Specifying functions:** Identifying what functions the system must perform.
- **Describing behaviors:** Defining what behaviors the system exhibits.
- **Defining interfaces:** Identifying what interfaces are defined.
- **Identifying constraints:** Determining what constraints apply.

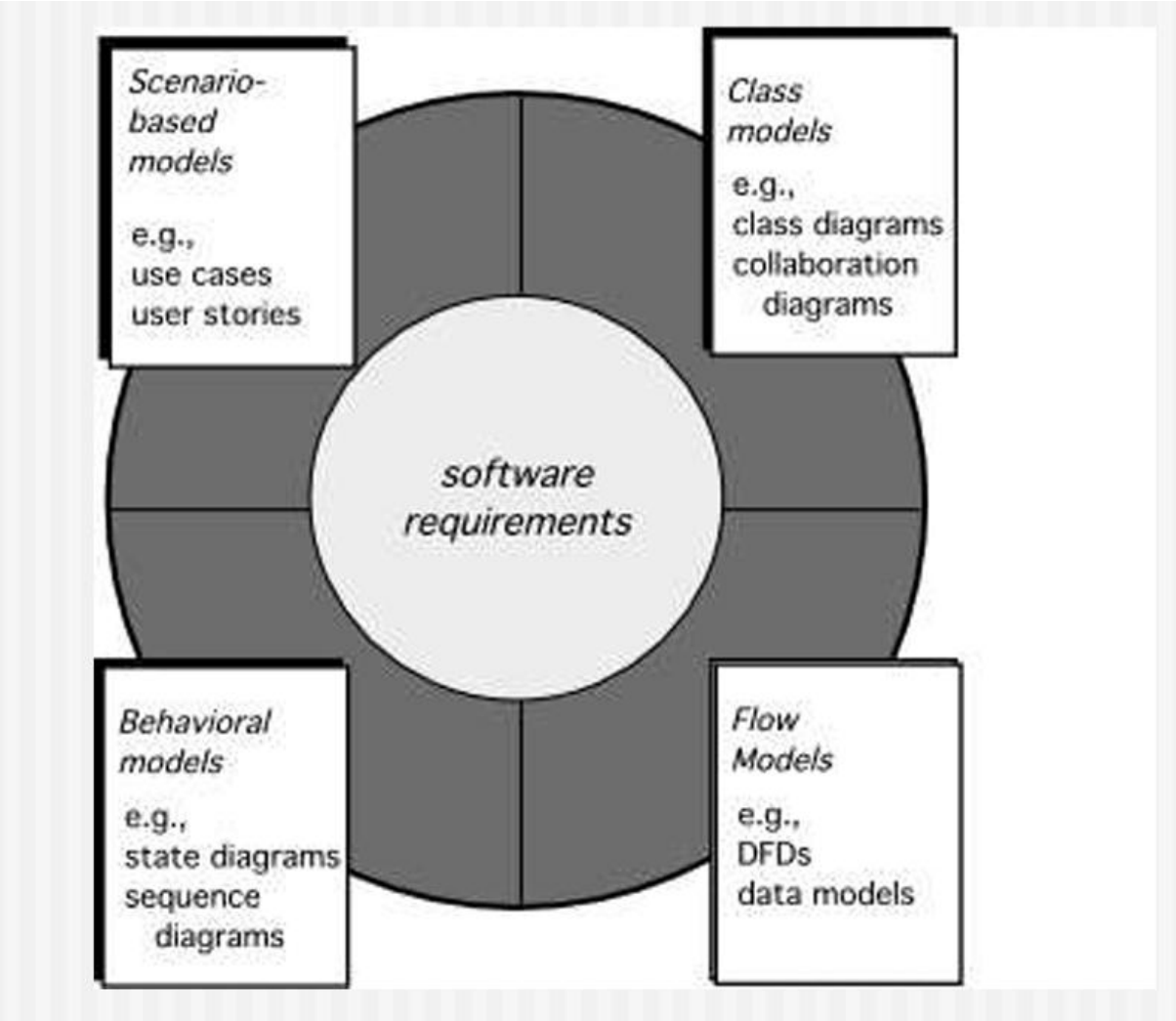
The overall objectives and philosophy of requirements analysis emphasize understanding 'what' rather than 'how'. This means focusing on the essential needs and goals of the system without getting bogged down in implementation details. The analysis model should describe what the customer wants, establish a basis for design, and provide a target for validation.

To achieve these objectives, the analysis model incorporates several key elements:

- **Scenario-based elements** Use cases are created to describe how users and other actors interact with the system. User scenarios are refined to extract analysis classes, which represent business domain entities visible to the end user.
- **Class-based elements** Analysis classes are defined with their attributes and required services. Relationships and collaborations between classes are identified. Nouns in use-case scripts can be candidate classes.

- **Behavioral elements** The requirements model depicts the behavior of the system.
- **Flow-oriented elements** Data flow diagrams can be created.

The requirements model acts as a bridge between the system description and the design model, ensuring that all elements are traceable to the design. Some design invariably occurs as part of analysis, and some analysis will be conducted during design.



What are the significances of use cases. With reference to software engineering lab-mini project, draw a use case diagram. Also, write at least 4 use case descriptions in detail.

Significance of Use Cases in Software Engineering

Use cases are crucial in software engineering as they define **functional requirements** by describing interactions between **users (actors) and the system**. Their significance includes:

1. **Understanding Requirements:** Helps stakeholders visualize system behavior.
2. **Better Communication:** Provides a clear description of system interactions for developers, testers, and business analysts.
3. **Facilitates Design and Testing:** Helps in designing system modules and test cases.
4. **Enhances User Experience:** Ensures the system meets user needs efficiently.
5. **Risk Reduction:** Identifies missing or unclear requirements early in the development process.

Use Case Diagram for a Mini Project (Learning Management System - LMS)

Actors:

- **Admin**
- **Tutor**
- **Student**

Main Use Cases:

- **User Registration & Login**
- **Course Management (Add/Remove Courses)**
- **Assignment Submission**
- **Performance Tracking**
- **Doubt Resolution**
- **Leaderboard Management**
- **Email Notifications**

```
actor "Admin" as Admin
actor "Tutor" as Tutor
actor "Student" as Student

rectangle "Learning Management System" {
  Admin --> (Manage Users)
  Admin --> (Manage Courses)
  Admin --> (View Reports)

  Tutor --> (Create Assignments)
  Tutor --> (Evaluate Submissions)
  Tutor --> (Respond to Doubts)

  Student --> (Register/Login)
  Student --> (Enroll in Course)
  Student --> (Submit Assignment)
  Student --> (Raise Doubt)
  Student --> (View Performance)

  (Register/Login) <--> (Email Notification) : "Triggers"
}
@enduml
```

Use Case Descriptions

1. Use Case: Register/Login

- **Actor:** Student
- **Description:** Allows users to register for an account and log in to access the LMS.
- **Preconditions:** The user must not have an existing account for registration.
- **Steps:**

1. The user enters login credentials.
2. The system verifies credentials with the database.
3. If correct, the user is granted access; else, an error is shown.

- **Postconditions:** The user is logged in or receives an error message.

---

2. Use Case: Enroll in Course

- **Actor:** Student
- **Description:** Students can enroll in courses of their choice.
- **Preconditions:** The user must be logged in.
- **Steps:**
    1. The student selects a course from the available list.
    2. The system verifies eligibility and enrollment capacity.
    3. If valid, enrollment is confirmed, and access is granted.
    4. Confirmation is displayed.
- **Postconditions:** The student is enrolled in the course.

---

3. Use Case: Create Assignments

- **Actor:** Tutor
- **Description:** Tutors can create assignments for students.
- **Preconditions:** The tutor must be logged in and assigned to a course.
- **Steps:**
    1. The tutor navigates to the assignment section.
    2. They enter assignment details (title, description, deadline).
    3. The system saves the assignment in the database.
    4. The assignment is published and visible to students.
- **Postconditions:** The assignment is available for students to complete.

---

4. Use Case: Submit Assignment

- **Actor:** Student

- **Description:** Students submit their completed assignments.
- **Preconditions:** The student must be enrolled in the course and the assignment must be active.
- **Steps:**
    1. The student selects an assignment.
    2. They upload their solution.
    3. The system validates and stores the submission.
    4. A success message is displayed.
- **Postconditions:** The assignment is submitted and recorded.

Differentiate between Scenario based and Behavioural based software modelling with suitable example.

- **Scenario-based modeling** represents the system from the **user's point of view**. It uses narratives or templates to describe interactions between an actor and the software to define key steps for a specific function or interaction. Use cases, activity diagrams, and swimlane diagrams are elements of scenario-based modeling.
- **Behavioral modeling** depicts the **dynamic behavior** of the system or product. It represents how software responds to external events or stimuli. **State diagrams** and **sequence diagrams** are the notation used for behavioral modeling.

Key Differences:

- **Perspective:** Scenario-based modeling focuses on user interaction, while behavioral modeling focuses on the system's response to events.
- **Elements:** Scenario-based modeling uses use cases and activity diagrams. Behavioral modeling uses state diagrams and sequence diagrams.
- **Focus:** Scenario-based modeling describes what the user does with the system. Behavioral modeling describes how the system changes state in response.

Examples:

- **Scenario-Based Modeling:** A use case for a digital music file application might describe the steps a user takes to download an MP3 music file and store it in the application’s library.

- **Behavioral Modeling:** A state diagram for a ControlPanel object in a security system shows the active states for each class and the events that trigger changes between states. For example, the transitions from the Idle state can occur if the system is reset, activated, or powered off.

In summary, scenario-based modeling illustrates *how* users interact with the system, while behavioral modeling illustrates *how* the system reacts to those interactions and other events. Both provide different viewpoints for understanding software requirements.

Explain Class-Responsibility-Collaborator (CRC modelling, and draw CRC card with suitable example (Software Engineering Lab - Mini Project)

Class-Responsibility-Collaborator (CRC) Modelling in Software Engineering

CRC (Class-Responsibility-Collaborator) modelling is a technique used in object-oriented software design to identify **classes**, their **responsibilities**, and their **collaborations** within a system.

Key Components of CRC Modeling:

1. **Class:** Represents an entity in the system.
2. **Responsibilities:** Defines what the class does (methods or actions).
3. **Collaborators:** Other classes that work together to fulfill the responsibility.

Significance of CRC Modeling

- Helps in **object-oriented design and analysis**.
- Encourages **team collaboration** in designing system components.
- Helps in understanding **class interactions** before coding.
- Enhances **maintainability and modularity**.

CRC Card Example for a Learning Management System (LMS)

Let's create **CRC cards** for a **mini-project (LMS system)** that includes **Admin, Student, Tutor, and Course Management**.

Example CRC Cards

1. CRC Card for User Class

Class: User

Responsibilities

- Register/Login

- Update profile

- View Courses

2. CRC Card for Course Class

Class: Course

Responsibilities

- Store course details

- Allow enrollment

- Assign Tutor

3. CRC Card for Tutor Class

Class: Tutor

Responsibilities

- Create assignments

- Evaluate submissions

- Answer doubts

4. CRC Card for Student Class

Class: Student

Responsibilities

- Enroll in a course

- Submit assignments

- Raise doubts

- View performance

CRC Diagram Representation (PlantUML)

Here is a **CRC diagram** using PlantUML:

```
@startuml

class User {
  + Register()
  + Login()
  + UpdateProfile()
  + ViewCourses()
}
User --> AuthenticationService
User --> CourseManager

class Course {
  + StoreDetails()
  + AllowEnrollment()
  + AssignTutor()
}
Course --> User
Course --> Admin

class Tutor {
  + CreateAssignment()
  + EvaluateSubmission()
  + AnswerDoubts()
}
Tutor --> Course
Tutor --> Student

class Student {
```

```
 + EnrollInCourse()
 + SubmitAssignment()
 + RaiseDoubt()
 + ViewPerformance()
}
Student --> Course
Student --> Tutor
Student --> ReportManager

@enduml
```

Conclusion

- **CRC modeling** helps in structuring **object-oriented systems**.
- It defines **clear responsibilities** and **class interactions**.
- The **LMS example** shows how classes collaborate in a real-world project.

Why waterfall model of the software engineering is not an accurate reflection of software development activities? What are the issues addressed by Umbrella Activities

The waterfall model is not always an accurate reflection of software development activities because real projects rarely follow the sequential flow that the model proposes. The waterfall model is a plan-driven process where process activities are planned and scheduled before work begins. It suggests a systematic, sequential approach to software development.

Issues with the Waterfall Model:

- **Rigidity** Real software projects rarely follow the sequential flow of the waterfall model. Although the linear model can accommodate iteration, it does so indirectly, which can cause confusion as the project team proceeds.
- **Inflexibility** The waterfall model requires that all requirements be explicitly stated by the customer, which is often difficult. The model has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.

- **Delayed Feedback** A working version of the program is not available until late in the project timeline. If a major issue is not detected until the working program is reviewed, it can be disastrous.
- **Inability to accommodate changes** Modern applications often change before they are presented to the end user, and then after the first version is in use. The waterfall model is often inappropriate for such work.

Umbrella activities address project management, tracking, and control throughout the software process. They are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. According to the sources, umbrella activities include:

- Project tracking and control
- Risk management
- Software quality assurance
- Configuration management
- Technical reviews
- Measurement
- Reporting
- Production of work products like models, documents, logs, forms and lists
- Change control mechanisms
- Defined approaches for customer communication
- Established methods for representing user requirements
- Compliance with software development standards

Explain the DevOps concepts and process for implementation of educational ERP system.

DevOps is a practice that allows a single team to manage the entire application development life cycle. It aims to shorten the system's development life cycle while frequently delivering features, fixes, and updates in close alignment with business objectives. It is an approach through which superior quality software can be developed quickly and with more reliability.

The DevOps life cycle includes stages such as **continuous development, continuous integration, continuous testing, continuous deployment, and continuous monitoring**.

- **Continuous Development** This phase involves the planning and coding of the software. During the planning phase, the vision of the project is decided, and the developers begin developing the code for the application. Version control tools such as Git maintain the code. Branching is an important aspect of version control, especially when multiple developers are working in parallel.
- **Continuous Testing** In this stage, the developed software is continuously tested for bugs. Automation testing tools like Selenium, TestNG, and JUnit are used to test multiple code bases thoroughly in parallel. Docker Containers can be used for simulating the test DevOps.
- **Continuous Integration** This stage is about integrating the code which is developed by multiple developers. Continuous integration is possible because of tools like Jenkins.
- **Continuous Deployment** This phase is where the code is deployed to the production environment. Continuous deployment can be achieved with the help of configuration management tools like Chef, Puppet, SaltStack, and Ansible. Docker again plays a vital role.
- **Continuous Monitoring** This is a crucial stage in the DevOps lifecycle. The operation teams will monitor the behavior of the software in the production environment.

Explain the incremental development process model with neat block diagram. List its benefits and problems.

The **incremental model** combines aspects of linear and parallel processing. It applies linear sequences in a staggered fashion, where each sequence produces deliverable increments of the software. The process flow for any increment can incorporate prototyping.

Process:

- The incremental model delivers a series of releases or increments, providing more functionality with each delivery.
- The first increment is often a core product, addressing basic requirements, with supplementary features remaining undelivered.
- The core product is used or evaluated by the customer, and a plan is developed for the next increment based on feedback.
- The plan addresses modifications to the core product and the delivery of additional features and functionality.
- This process repeats until the complete product is produced.

Benefits of the Incremental Model:

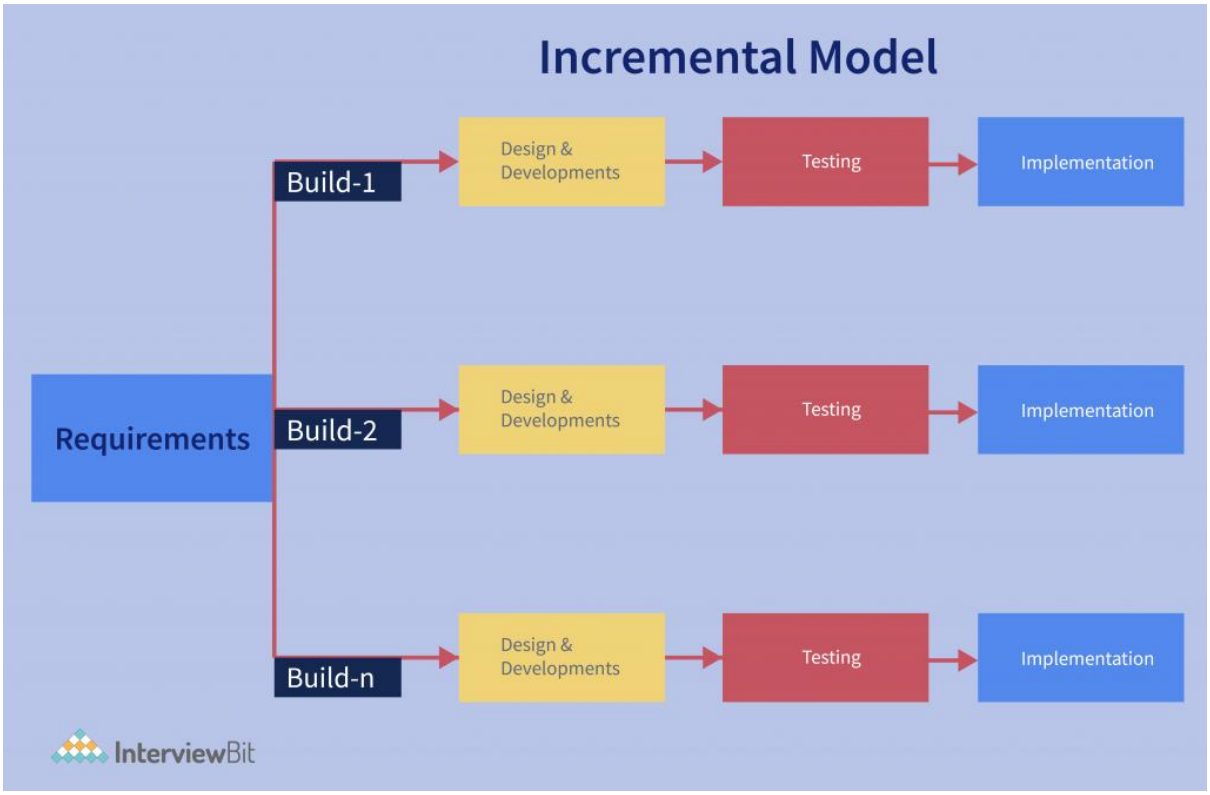
- Working versions of software are produced rapidly.
- It can be easier to manage change and is more flexible, making it less costly to change scope and requirements.
- Testing and debugging are easier during a smaller iteration.
- The customer can respond to each build.
- It lowers initial delivery costs.
- Risk management is easier because risky elements are identified and handled during their specific iteration.
- The cost of accommodating changing customer requirements is reduced because less analysis and documentation has to be redone compared to the waterfall model.
- It is easier to get customer feedback on the development work because they can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible, even if all of the functionality has not been included, allowing them to use and gain value from the software earlier than with a waterfall process.

Problems with the Incremental Model:

- Needs good planning and design.
- A clear and complete definition of the whole system is needed before it can be broken down and built incrementally.
- The total cost may be higher than with the waterfall model.
- The process is not visible, and managers need regular deliverables to measure progress.
- System structure can degrade as new increments are added unless time and money are spent on refactoring to improve the software; regular change tends to degrade the structure.
- It can be hard to identify common facilities needed by all increments, and can be difficult when a replacement system is being developed because users want all the functionality of the old system.
- Bureaucratic procedures in large organizations may conflict with a more informal iterative or agile process.

The incremental model is suitable when the requirements of the complete system are clearly defined and understood, major requirements are defined but some details can evolve, and there is a need to get software functionality to users quickly. It is particularly

useful when staffing is unavailable for a complete implementation by the business deadline.



What is the need of Software Requirement Specification? Why Requirement Elicitation is difficult? What are the problems in requirement elicitation?

Need for Software Requirements Specification (SRS):

- An SRS is a detailed description of all aspects of the software that needs to be built before a project starts.
- **SRS is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence**.
- It establishes a solid base for design and construction, and without it, the resulting software has a high probability of not meeting the customer's needs.
- The requirements model should describe what the customer wants, establish a basis for design, and establish a target for validation.
- A software requirements document (also called software requirements specification) is an official statement of what the system developers should implement.

- It is important to have an unambiguous representation of the requirements and have it made available in a centralized place so that all the stakeholders have the same interpretation of the requirements.

However, the sources also point out some caveats regarding SRS:

- A formal SRS is not always written.
- There are instances where effort expended on an SRS might be better spent in other software engineering activities.
- When software is to be developed by a third party, when a lack of specification would create severe business issues, or when a system is extremely complex or business-critical, an SRS may be justified.

Why Requirement Elicitation is Difficult:

- Understanding the requirements of a problem is among the most difficult tasks that face a software engineer.
- **Eliciting and understanding requirements from system stakeholders is a difficult process**.
- Stakeholders often don’t know what they want from a computer system except in the most general terms; they may find it difficult to articulate what they want.
- The information obtained from the customer during inception and elicitation is expanded and refined during elaboration.
- Software developers don’t pay enough attention to requirements engineering.

Problems in Requirement Elicitation:

- **Problems of scope**: The boundary of the system is ill-defined, or customers specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.
- **Problems of understanding**: Customers aren't completely aware of what is needed, have trouble communicating needs to the system engineer, omit information that is believed to be “obvious," specify requirements that conflict with the needs of other customers/users, or specify requirements that are ambiguous or untestable.
- **Problems of volatility**: The requirements change over time.
- All application specialists use terminology and jargon specific to a domain, and domain knowledge is so familiar to stakeholders that they find it hard to articulate or think it isn't worth mentioning.
- The flexibility of natural language often causes problems. There is scope for writing unclear requirements, and readers (the designers) may misinterpret requirements because they have a different background to the user. It is easy to

amalgamate several requirements into a single sentence, and structuring natural language requirements can be difficult.

- During the software process, the stakeholders' understanding of the problem is constantly changing.
- Stakeholders may have limited technical knowledge and time to interact with the requirements engineer.

Explain activities and the steps used for negotiating software requirements for canteen management system.

Activities and steps for negotiating software requirements for a canteen management system, based on the sources:

- **Identification of key stakeholders** Identify the key stakeholders who will be involved in the negotiation process. This includes the system or subsystem's key stakeholders.
- **Determine stakeholders' "win conditions"** Ascertain each stakeholder's "win conditions".
- **Negotiation of "win-win" conditions** Reconcile the stakeholders' win conditions to create a set of win-win conditions for all parties involved, including the software team. Successful completion of these initial steps achieves a win-win result, which becomes the key criterion for proceeding to subsequent software engineering activities.
- **Collaborative requirements gathering** Conduct meetings attended by software engineers and stakeholders. Establish rules for preparation and participation. Suggest an agenda formal enough to cover all important points but informal enough to encourage the free flow of ideas. A facilitator controls the meeting, and a definition mechanism is used. The goal is to identify the problem, propose solution elements, and negotiate different approaches.
- **Address conflicting ideas** When encountering stakeholders with conflicting ideas about the software, employ a process pattern to address this problem and suggest an effective approach.
- **Elicitation and analysis** During requirements elicitation and analysis, software engineers collaborate with customers and end-users to gather information about the application domain, system services, performance requirements, and constraints.
- **Requirements Specification** Document the requirements and input them into the next round. Produce formal or informal requirements documents.

- **Requirements Validation** Check the requirements for validity, consistency, completeness, realism, and verifiability.
- **Considerations for COTS** In cases of COTS (Commercial Off-The-Shelf) product reuse, recognize the need to trade off specific requirements against rapid development and reuse and design a system architecture that allows the COTS systems to operate together.
- **Adaptation and Change Tuning:** Understand that proposed process changes may not be effective immediately and that a tuning phase may be needed to identify and address minor problems. This includes a plan for implementing the measures.
- **Requirements Management:** Implement requirements management to manage changes, ensure proper analysis, and track changes throughout the system.

These steps align with establishing a common understanding between the customer and developers, guided by a shared view of customer requirements.

Explain the IEEE standard requirement document with its structure.

The sources discuss the IEEE standard for software requirements documents and their structure. IEEE standards provide a generic basis for developing more detailed organizational standards.

Key aspects of the IEEE standard requirement document and its structure:

- **Official Statement:** The software requirements document, sometimes called the software requirements specification (SRS), is an official statement of what the system developers should implement.
- **Content**: It should include both the user requirements for a system and a detailed specification of the system requirements.
- **Flexibility:** The IEEE standard is generic and can be adapted to specific uses.
- **Diverse Audience:** The requirements document has a diverse set of users, so the document needs to be a compromise.

A possible organization for a requirements document based on an IEEE standard:

- **Introduction:** Describes the need for the system, its functions, and how it works with other systems. Explains how the system fits into the organization's business or strategic objectives.
- **Glossary:** Defines the technical terms used in the document, without assuming the reader's expertise.
- **User Requirements Definition:** Describes the services provided to the user and the non-functional system requirements. This description may use natural

language, diagrams, or other notations understandable to customers. Specifies product and process standards that must be followed.

- **System Architecture:** Presents a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules and highlighting reused architectural components.
- **System Requirements Specification:** Describes the functional and non-functional requirements in more detail and defines interfaces to other systems.
- **System Models:** Includes graphical system models showing the relationships between system components, the system, and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
- **System Evolution:** Describes anticipated changes to the system and helps maintainers and designers include support for likely changes.
- **Index:** Includes an alphabetic index, an index of diagrams, and an index of functions to navigate the document.

For long documents, it is important to include a comprehensive table of contents and document index.

Explain in detail any three types of process models used for software development with one real time example / system / application.

Three types of process models used for software development, with real-time examples:

1. Waterfall Model

- The waterfall model represents the fundamental process activities of specification, development, validation, and evolution as separate process phases.
- It takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing, and so on.
- The waterfall model is an example of a **plan-driven process**. In principle, all of the process activities are planned and scheduled before starting work on them.
- **Applicability**: The Waterfall model is useful in situations where requirements are well defined and stable.
- **Real-time example**: Developing critical control software for aircraft. The requirements are collected from software engineers and external stakeholders

(such as the regulatory certification authority), and the system is developed using a plan-driven approach.

2. Incremental Model

- **The incremental model** combines elements of linear and parallel process flows.
- It applies linear sequences in a staggered fashion as calendar time progresses.
- Each linear sequence produces deliverable "increments" of the software.
- In the incremental model the whole requirement is divided into various builds.
- Multiple development cycles take place here, making the life cycle a “multi-waterfall” cycle.
- Cycles are divided up into smaller, more easily managed modules.
- Each module passes through the requirements, design, implementation, and testing phases.
- A working version of software is produced during the first module, to have working software early on during the software life cycle.
- **Applicability**: The incremental model is suitable for projects where the details of product or system extensions have yet to be defined.
- **Real-time example**: Agile development model is also a type of Incremental model. Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly tested to ensure software quality is maintained. It is used for time critical applications.

3. Agile Process Model (XP and Scrum)

- **Agile process models** emphasize project “agility” and follow a set of principles that lead to a more informal approach to software process.
- Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads, and producing high-quality code. They involve the customer directly in the development process.
- Agile development model is also a type of Incremental model.
- **Applicability**: Agile process models are appropriate for many types of projects and are particularly useful when Web applications are engineered.
- **Real-time example:** Extreme Programming (XP).
    - XP model gives high importance on testing and considers it to be the primary factor to develop a fault-free software.

- Basic activities followed in software development are:
  - Coding
  - Testing
  - Listening

Differentiate between Extream Programming (XP) and Scrum agile models

Extreme Programming (XP) and Scrum are both agile process models, but they differ in their approach and focus.

Extreme Programming (XP):

- **Focus**: XP is primarily a technical approach to agile software development. It emphasizes good programming practices and is designed to improve software quality and responsiveness to customer requirements.
- **Practices**: XP integrates a range of good programming practices such as frequent releases, continuous software improvement, and customer participation. Key practices include test-first development, pair programming, and continuous integration.
- **Values**: XP is based on five values: communication, simplicity, feedback, courage, and respect.
- **Requirements**: In XP, requirements are expressed as user stories, which are implemented directly as a series of tasks.
- **Testing**: XP places a high importance on testing and considers it a primary factor in developing fault-free software. Automated tests are developed before a program feature is created, and all tests must successfully execute when an increment is integrated into the system.
- **Release Cycle**: New software versions may be built several times a day, with releases delivered to customers roughly every two weeks.
- **Planning**: XP uses a "planning game" involving the whole development team and customer representatives.
- **Spikes**: XP may use "spikes," which are prototyping or trial development efforts to understand problems and solutions. Spikes are simple programs constructed to explore the suitability of a proposed solution and are similar to prototypes.
- **Change Management**: In XP, any developer can improve any code without getting external approval.
- **Team Structure**: XP uses small, self-organizing teams.
- **Suitable Projects**: XP is suitable for small projects.

Scrum:

- **Focus**: Scrum is primarily a project management framework for managing iterative development. It focuses on how to manage iterative development rather than specific technical approaches to agile software engineering.
- **Management**: Scrum provides a management framework for the project, with an emphasis on team empowerment and decision-making.
- **Framework Activities**: Scrum principles guide development activities within a process that incorporates the following framework activities: requirements, analysis, design, evolution, and delivery.
- **Roles**: Scrum defines specific roles such as the Product Owner, Scrum Master, and Development Team.
- **Sprints**: Scrum is centered around sprints, which are fixed time periods when a system increment is developed. Sprints are fixed length, normally 2-4 weeks.
- **Planning**: Planning is based on prioritizing a backlog of work and selecting the highest-priority tasks for a sprint.
- **Meetings**: Short daily meetings involving all team members are held to review progress and reprioritize work if necessary.
- **Suitable Projects**: Scrum has proven effective for projects with tight timelines, changing requirements, and business criticality. It has been used on projects with 500+ people.
- **Sprint Cycle**: A Scrum sprint is a planning unit in which the work to be done is assessed, features are selected for development, and the software is implemented. Key characteristics of this process are as follows: sprints are fixed length, normally 2–4 weeks.
- **Sprint planning**: Decide how to achieve sprint goal (design) and create sprint backlog (tasks) from product backlog items (user stories / features).
- **Daily Scrum Meeting**: Daily, ~15 minutes, Stand-up, whole team participates.

Key Differences Summarized:

- XP is a technical approach focused on coding and testing practices. Scrum is a management framework focused on iterative development and team organization.
- XP prescribes specific development practices like pair programming and test-first development. Scrum does not prescribe specific programming practices but can be used with technical agile approaches like XP.
- The innovative feature of Scrum is its central phase, namely the sprint cycles. The 'Scrum master' is a facilitator who arranges daily meetings, tracks the

backlog of work to be done, records decisions, measures progress against the backlog, and communicates with customers and management outside of the team.

Integration:

- Scrum can be used with more technical agile approaches, such as XP, to provide a management framework for the project. Scrum does not prescribe the use of programming practices such as pair programming and test-first development. It can therefore be used with more technical agile approaches, such as XP, to provide a management framework for the project.
- ASD concepts of collaboration and self-organizing teams can be adapted to a combined process model.

Explain Water Fall model model with example

The **waterfall model** is a sequential software development approach where progress flows steadily downwards through distinct phases. It is also referred to as a linear-sequential life cycle model.

Key Aspects of the Waterfall Model:

- **Sequential Phases:** Each phase must be completed before the next one begins. These phases include requirements analysis and definition, system and software design, implementation and unit testing, and integration and system testing.
- **Plan-Driven:** The waterfall model is a **plan-driven process**, requiring all activities to be planned and scheduled upfront.
- **Reviews:** At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project.
- **Documentation:** Produces documentation that intended to communicate the system needs of the customer to the system developers.

Applicability:

- The waterfall model is best suited for projects with well-understood and stable requirements.
- It can be used when the requirements are very well known, clear and fixed.
- Technology is understood.
- Product definition is stable.
- There are no ambiguous requirements.
- Ample resources with required expertise are available freely.

- The project is short.

Advantages:

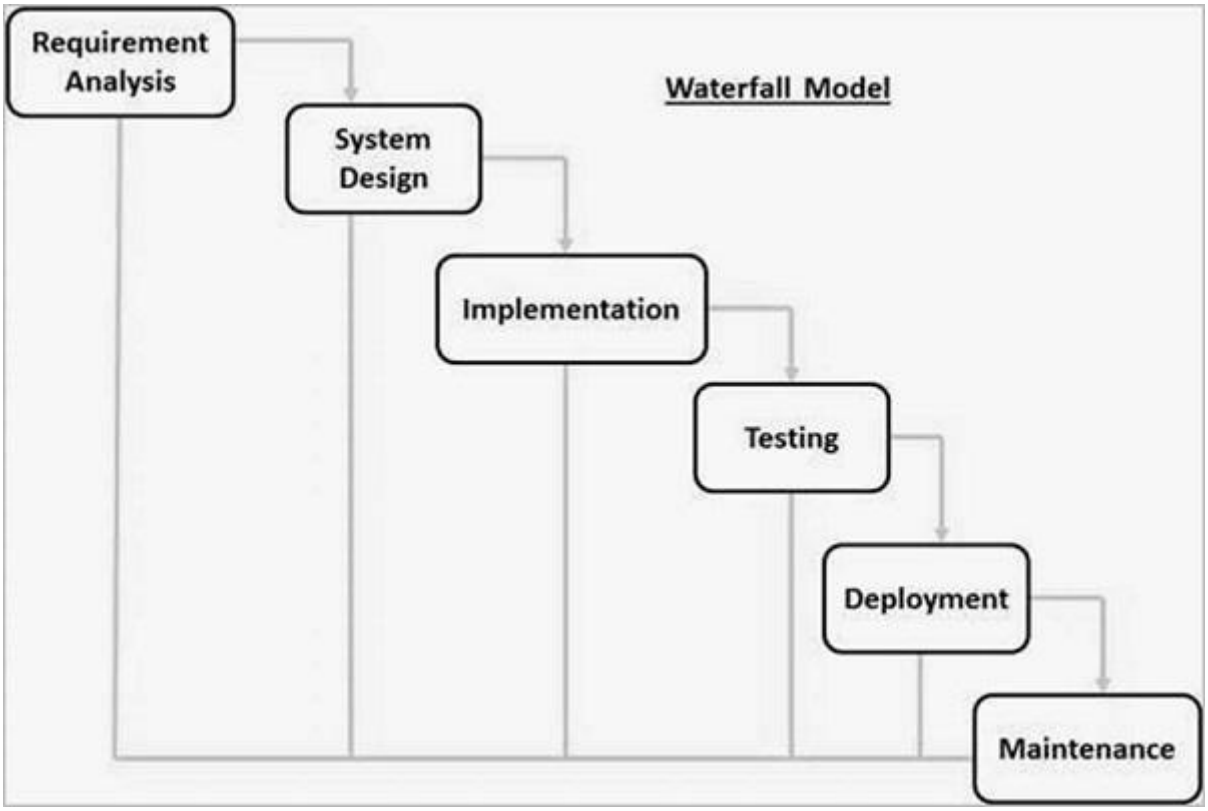
- Simple, easy to understand and use.
- Easy to manage due to the rigidity of the model.
- Phases are processed and completed one at a time.

Disadvantages:

- Difficult to accommodate changes after the testing stage.
- No working software is produced until late in the life cycle.
- High amounts of risk and uncertainty.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

Example:

- Developing critical control software for aircraft, where requirements are collected from software engineers and external stakeholders, and the system is developed using a plan-driven approach.
- Used to develop enterprise applications.



Explain working principles of Agile model. Write Advantages and disadvantages of Agile process model.

The **agile model** is a type of incremental model that emphasizes iterative and rapid cycles in software development. Agile methods focus on delivering working software quickly, reducing process overheads, and responding to changing customer requirements.

Working Principles of the Agile Model:

- **Incremental Development**: Software is developed in small increments, with each release building on previous functionality.
- **Rapid Cycles**: Agile development uses rapid cycles to produce frequent releases.
- **Customer Satisfaction**: Delivering useful software continuously ensures customer satisfaction.
- **Adaptability**: Agile methods welcome changing requirements, even late in development.
- **Collaboration**: Customers, developers, and testers constantly interact with each other. Business people and developers work together daily throughout the project.
- **Communication**: Face-to-face conversation is the best form of communication.
- **Technical Excellence**: Agile emphasizes continuous attention to technical excellence and good design.
- **Simplicity**: The work should emphasize economy of action by keeping the technical approach as simple as possible.
- **Self-Organizing Teams**: Agile relies on self-organizing teams that have control over the work they perform.
- **Process Adaptation**: The modeling approach should be adapted to the needs of the agile team.

Advantages of the Agile Model:

- **Customer Satisfaction**: Rapid and continuous delivery of useful software leads to high customer satisfaction.
- **Adaptability**: Agile welcomes changing requirements, even late in development.
- **Frequent Delivery**: Working software is delivered frequently, typically in weeks rather than months.

- **Collaboration and Communication**: Emphasizes people and interactions, with close, daily cooperation between business people and developers and face-to-face conversation.
- **Technical Excellence**: Continuous attention to technical excellence and good design.
- **Responsiveness**: Agile processes respond appropriately to changes.
- **Limited Planning**: Very limited planning is required to get started with the project.
- **Flexibility**: New features can be easily added or removed based on feedback, giving the customer the finished system they want or need.
- **Risk Management**: Easier to manage risk because risky pieces are identified and handled during iteration.
- **Productivity**: Increases productivity.
- **Reduced Time to Benefits**: Reduces time to benefits.

Disadvantages of the Agile Model:

- **Effort Estimation**: Difficult to assess the effort required at the beginning of the software development life cycle, especially for large deliverables.
- **Lack of Documentation**: There is a lack of emphasis on necessary designing and documentation.
- **Potential for Scope Creep**: The project can easily get taken off track if the customer representative is not clear on the final outcome.
- **Demands Senior Programmers**: Only senior programmers are capable of taking the kind of decisions required during the development process.
- **Not Suited for Newbie Programmers**: Has no place for newbie programmers unless combined with experienced resources.
- **Requires Customer Involvement**: Effective collaboration with your customer will only occur if you jettison any “us and them” attitudes.

Explain requirement engineering Evaluation Scheme: Iception, Elicitaion , Elaboration and negotiation

Requirements engineering is the process of defining what services are required from a system, along with identifying the constraints on its operation and development. It involves understanding and documenting customer needs to communicate them effectively to system developers.

The main activities in the requirements engineering process include feasibility study, requirements elicitation and analysis, requirements specification, and requirements validation.

Requirements engineering encompasses seven distinct tasks:

- **Inception**: Establishes the scope and nature of the problem to be solved.
- **Elicitation**: Helps stakeholders define what is required.
- **Elaboration**: Refines and modifies basic requirements.
- **Negotiation**: Addresses conflicting priorities to reach a reasonable solution.
- **Specification**: Documents the requirements in a written document, graphical models, or prototype.
- **Validation**: Ensures the work products meet quality standards and stakeholder expectations.
- **Management**: Manages the requirements as they are transformed into an operational system.

Requirements engineering builds a bridge to design and construction.

Evaluation Scheme

Requirements engineering involves distinct tasks such as inception, elicitation, elaboration, and negotiation.

- **Inception**:
    - Defines the scope and nature of the problem.
    - Establishes a basic understanding of the problem, the stakeholders, the solution's nature, and the effectiveness of communication between the customer and developer.
    - During inception, the scope of the problem and the overall perception of a solution are established through basic questions and answers.
    - It defines the scope and nature of the problem to be solved.
    - Aims to identify stakeholders, recognize multiple viewpoints, work toward collaboration, and ask initial questions to understand the work request, solution users, economic benefits, and potential alternative solutions.
    - It Includes initial requirements capture, cost-benefit analysis, risk analysis, project scope definition, candidate architecture, disposable prototype development, and a preliminary use case model.
    - Context-free questions can be asked to stakeholders during inception.
- **Elicitation**:

    - A task that helps stakeholders define what is required.
    - It obtains requirements from all stakeholders.
    - Combines problem-solving, elaboration, negotiation, and specification.
    - Meetings are conducted with software engineers and customers, establishing rules for preparation and participation.
    - It uses collaborative requirements gathering, quality function deployment, and usage scenarios.
    - The stakeholders work together to identify the problem and propose elements of the solution and negotiate different approaches.
    - Elicitation involves gathering information about the system and existing systems and distilling user and system requirements.
    - Eliciting requirements may involve interviews, scenarios, use cases, and ethnographic analysis.
- **Elaboration**:
    - Where basic requirements are refined and modified.
    - It creates an analysis model that identifies data, function, and behavioral requirements.
    - Expands and refines the information obtained during inception and elicitation, focusing on developing a refined requirements model.
    - Driven by creating and refining user scenarios that describe how end-users will interact with the system.
    - Basic use cases evolve into more elaborate template-based use cases, serving as input for creating elements of the requirements model.
    - Includes requirements analysis and capture, use case analysis, scenarios, and various diagrams such as sequence, collaboration, class, activity, component, and state diagrams.
- **Negotiation**:
    - Addresses conflicting priorities, and assesses costs and risks to reach a reasonable solution.
    - Aims to agree on a deliverable system that is realistic for developers and customers.
    - Prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, so that each party achieves some measure of satisfaction.

- Stakeholder negotiations should be organized so that compromises can be reached.
- The software team and project stakeholders negotiate the priority, availability, and relative cost of each requirement, with the intent to develop a realistic project plan.