    - **ecification:** The specification is **owned by the customer**, and they make decisions on required software changes.

Question: What is the difference between software engineering, computer science, and system engineering? **Answer:**

- **Computer science** focuses on theory and fundamentals.
- **Software engineering** is concerned with the practicalities of developing and delivering useful software.
- **System engineering** is concerned with all aspects of computer-based systems development, including hardware, software, and process engineering. Software engineering is a part of this more general process.

Question: What are the key challenges facing software engineering? **Answer:** The key challenges facing software engineering are:

- **Coping with increasing diversity:** Developing software for different platforms and types of systems.
- **Demands for reduced delivery times:** Developing and delivering software more quickly.
- **Developing trustworthy software:** Ensuring software is dependable and secure.

Question: Why is software engineering ethics important? List the eight principles of the ACM/IEEE Code of Ethics. **Answer:** Software engineering involves wider responsibilities than just technical skills. Engineers must behave in an honest and ethically responsible way to be respected as professionals. Their work can have a significant impact, and they have opportunities to do good or cause harm. The eight principles are :

1. **PUBLIC:** Act consistently with the public interest.
2. **CLIENT AND EMPLOYER:** Act in the best interests of their client and employer, consistent with the public interest.
3. **PRODUCT:** Ensure products and modifications meet the highest professional standards.
4. **JUDGMENT:** Maintain integrity and independence in professional judgment.
5. **MANAGEMENT:** Managers and leaders shall promote an ethical approach to software development.
6. **PROFESSION:** Advance the integrity and reputation of the profession.
7. **COLLEAGUES:** Be fair to and supportive of colleagues.
8. **SELF:** Participate in lifelong learning and promote an ethical approach to the practice.

Chapter 2: Software Process Models

Question: What is a software process model? Differentiate between plan-driven and agile processes. **Answer:** A software process model is an abstract representation of a software process. It presents a description of a process from a particular perspective.

- **Plan-driven processes:** All process activities are planned in advance, and progress is measured against this plan.
- **Agile processes:** Planning is incremental, and it is easier to change the process to reflect changing customer requirements.

Question: Describe the Waterfall model with a neat diagram. What are its main phases and drawbacks? **Answer:** The waterfall model is a plan-driven model with separate and distinct phases of specification and development. **Diagram: Phases:** The model includes the following separate phases:

1. Requirements analysis and definition
2. System and software design
3. Implementation and unit testing
4. Integration and system testing
5. Operation and maintenance **Drawbacks:** The main drawback is the **difficulty of accommodating change** after the process is underway. The inflexible partitioning into distinct stages makes it difficult to respond to changing customer requirements. This model is only appropriate when requirements are well-understood and stable.

Question: Illustrate the incremental development process model with a neat diagram. List its benefits and problems. **Answer:** In incremental development, specification, development, and validation are interleaved. The system is developed as a series of increments. **Diagram: Benefits:**

- **Reduced cost of change:** The cost of accommodating changing customer requirements is reduced. The amount of analysis and documentation to be redone is less than in the waterfall model.
- **Easier customer feedback:** It's easier to get customer feedback on the work that has been done.
- **Rapid delivery:** More rapid delivery and deployment of useful software to the customer is possible. **Problems:**

- **Process is not visible:** Managers need regular deliverables to measure progress. If systems are developed quickly, it's not cost-effective to produce documentation for every version.
- **System structure degrades:** The system structure tends to degrade as new increments are added. Unless time is spent on refactoring, regular change can corrupt the structure, making future changes difficult and costly.

Question: Which software process model—Waterfall, incremental, or Spiral—would you choose for an e-commerce project with evolving requirements? Justify your choice, describe the process model with a neat diagram, and list its advantages and disadvantages. **Answer:** For a project with **evolving requirements**, the **Incremental Development** model is the most suitable choice.

Justification: The primary drawback of the Waterfall model is its difficulty in accommodating change; it is only appropriate when requirements are stable and well-understood. An e-commerce project almost always has evolving requirements. The Incremental Development model is designed to handle this, as its fundamental benefit is **reducing the cost of accommodating changing customer requirements**.

Process Model Diagram and Description: (This is the same as the previous question)

- **Diagram:**
- **Description:** In this model, the activities of specification, development, and validation are interleaved. The system is developed and delivered as a series of increments, with each increment providing part of the required functionality.
- **Advantages:**
    - The cost of accommodating changing requirements is reduced.
    - It is easier to get customer feedback on the development work.
    - Rapid delivery and deployment of useful software is possible, allowing customers to gain value earlier.
- **Disadvantages:**
    - The process is not visible, as documentation is often not produced for every version, making it difficult for managers to measure progress.
    - System structure tends to degrade as new increments are added, requiring refactoring to avoid long-term maintenance problems.

Question: Explain why Boehm's spiral model is an adaptable model that can support both change avoidance and change tolerance activities. In practice, why has this model not been widely used? **Answer:** Boehm's spiral model represents the process as a spiral, where each loop represents a phase. It is adaptable because:

- **It supports Change Avoidance:** A key activity in each loop is **explicit risk assessment and reduction**. By identifying and managing risks (such as changing requirements) early, the process can anticipate possible changes before significant rework is needed, which is the goal of change avoidance.
- **It supports Change Tolerance:** The model is iterative, and a development model (like incremental development) is chosen for each loop. This incremental approach is the basis of change tolerance, as it's designed so that changes can be accommodated at a relatively low cost.

Why it is not widely used: The slides state that "in practice, however, the model is **rarely used as published** for practical software development" , although it has been influential in helping people think about iteration and risk-driven development.

Question: Explain the Reuse-Oriented Software Engineering model with a diagram. What are its main process stages? **Answer:** This model is based on systematic reuse, where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems. **Diagram:** **Process Stages:** The main stages of this process are:

1. **Component analysis:** Analyzing available components.
2. **Requirements modification:** Modifying requirements to match available components.
3. **System design with reuse:** Designing the system to integrate the components.
4. **Development and integration:** Integrating the components to create the system.

Question: What are the two approaches to reducing the costs of rework when coping with change? **Answer:** The two approaches to reducing the costs of rework are:

1. **Change avoidance:** The software process includes activities that can **anticipate possible changes** before significant rework is required. For example, developing a prototype to show key features to customers can help identify requirement issues early.
2. **Change tolerance:** The process is designed so that **changes can be accommodated at a relatively low cost**. This normally involves some form of incremental development, where changes can be implemented in future increments or only require altering a single increment.

Question: What is software prototyping? What are its benefits, and why should prototypes often be "throw-away"? **Answer:** A prototype is an initial version of a system used to demonstrate concepts and try out design options. **Benefits:**

- Improved system usability.
- A closer match to users' real needs.

- Improved design quality.
- Improved maintainability.
- Reduced development effort.

Why "Throw-away": Prototypes should often be discarded after development because they are not a good basis for a production system.

- They may be impossible to tune to meet **non-functional requirements** (like reliability or security).
- They are normally **undocumented**.
- The prototype structure is usually **degraded through rapid change**.
- They probably will not meet normal organizational **quality standards**.

Question: Explain the phases of the Rational Unified Process (RUP). **Answer:** The RUP is a modern generic process normally described from three perspectives. Its dynamic perspective shows four phases over time:

1. **Inception:** Establish the business case for the system.
2. **Elaboration:** Develop an understanding of the problem domain and the system architecture.
3. **Construction:** System design, programming, and testing.
4. **Transition:** Deploy the system in its operating environment. Each phase is iterative, and the whole set of phases may also be enacted incrementally.

Chapter 3: Agile Software Development

Question: What is the Agile Manifesto? List the five core principles of agile methods. **Answer:** The Agile Manifesto is a statement of values for "uncovering better ways of developing software". It states that while there is value in the items on the right, agile developers value the items on the left more:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

The five core principles are :

1. **Customer involvement:** Customers are closely involved to provide and prioritize requirements and evaluate iterations.

2. **Incremental delivery:** Software is developed in increments, with the customer specifying requirements for each.
3. **People not process:** The skills of the team are recognized and exploited; teams are left to develop their own ways of working.
4. **Embrace change:** System requirements are expected to change, so the system is designed to accommodate this.
5. **Maintain simplicity:** Focus on simplicity in the software and the process, actively working to eliminate complexity.

Question: With a suitable diagram, describe the extreme programming (XP) release cycle and discuss its key practices. **Answer:** Extreme Programming (XP) is an agile method that takes an 'extreme' approach to iterative development, with new versions built frequently and delivered to customers in two-week increments.

XP Release Cycle Diagram: The cycle consists of:

1. **Select story cards:** The customer chooses requirements (user stories) for the next release based on priority and time estimates.
2. **Plan release:** The team breaks stories into development "Tasks".
3. **Develop/Integrate/Test:** The software is developed, including running all tests for every new build. This involves practices like pair programming and test-first development.
4. **Release software:** A new, working version of the software is made available.
5. **Evaluate system:** The customer evaluates the new release.

Key XP Practices:

- **Incremental planning:** Requirements (stories) are prioritized by the customer for each release.
- **Small releases:** Frequent releases of the system incrementally add functionality.
- **Simple design:** Just enough design is done to meet the current requirements.
- **Test-first development:** Automated unit tests are written *before* the functionality is implemented.
- **Refactoring:** Developers continuously refactor (improve) the code to keep it simple and maintainable.
- **Pair programming:** Developers work in pairs to check each other's work.
- **Collective ownership:** All developers work on all areas of the system and take responsibility for all code.
- **Continuous integration:** As soon as a task is complete, it is integrated into the whole system. All tests must pass.
- **Sustainable pace:** Large amounts of overtime are not considered acceptable.
- **On-site customer:** A full-time customer representative is part of the team to provide requirements.

Question: Describe why test-first development helps the programmer to develop a better understanding of the system requirements. What are the potential difficulties with test-first development? **Answer: Better Understanding of Requirements:**

- Writing tests *before* writing the code **clarifies the requirements** that need to be implemented.
- The customer is directly involved in the process by helping to **develop acceptance tests** for the "stories" (requirements). This ensures the new code is validated against what the customer actually needs.

Potential Difficulties:

- Programmers may prefer programming to testing and **take shortcuts** , such as writing incomplete tests that don't check for all possible exceptions.
- Some tests can be **very difficult to write incrementally**, especially for complex user interfaces (e.g., display logic and workflow).
- It is difficult to **judge the completeness** of a set of tests. A large test set may not necessarily provide complete coverage.

Question: What is pair programming? Discuss its advantages. **Answer:** Pair programming is a practice used in XP where programmers work in pairs, sitting together at the same workstation to develop code. Pairs are created dynamically so all team members work with each other.

Advantages:

- It supports **collective ownership** and responsibility for the system; the team, not individuals, is responsible for problems.
- It acts as an **informal review process** because each line of code is looked at by at least two people.
- It helps support **refactoring** (code improvement), as the whole team benefits immediately from the improvements.
- It **spreads knowledge** across the team, which is important for reducing project risk when team members leave.
- Evidence suggests that a pair working together is **more efficient** than two programmers working separately.

Question: Illustrate the Agile Scrum development process model with a neat diagram and explain its significance. **Answer:** The Scrum approach is a general agile method focused on managing iterative development. **Diagram: Process Description:** The process consists of three phases:

1. **Initial Phase:** An outline planning phase to establish general objectives and design the software architecture.
2. **Sprint Cycles:** A series of "sprints" (fixed-length cycles of 2-4 weeks) where each cycle develops an increment of the system. This involves:
    - **Selection:** The team and customer select features from the **product backlog** (list of work).
    - **Development:** The team develops the software, isolated from distractions by the **Scrum master**.
    - **Review:** At the end of the sprint, the work is presented to stakeholders.
3. **Project Closure Phase:** This phase wraps up the project, completes documentation (like user manuals), and assesses lessons learned.

Significance (Benefits):

- The product is broken down into **manageable and understandable chunks**.
- **Unstable requirements do not hold up progress**.
- The **whole team has visibility** of everything, which improves communication.
- Customers see **on-time delivery of increments** and can provide feedback.
- It **establishes trust** between customers and developers.

Question: What is the difference between "scaling up" and "scaling out" agile methods? What are the challenges in scaling agile methods for large systems? **Answer:**

- **Scaling up:** Concerned with using agile methods for developing **large software systems** that cannot be developed by a small team.
- **Scaling out:** Concerned with how agile methods can be **introduced across a large organization** with many years of software development experience.

Challenges in Scaling for Large Systems:

- Large systems need **more up-front design and documentation**, which is counter to the "code-first" focus of some agile methods.
- **Cross-team communication** becomes difficult when teams are large, not co-located, and work in different time zones.
- **Continuous integration** of the whole system is often impossible; frequent builds are necessary but harder to manage.
- Large systems are often "brownfield systems" (interacting with existing systems), which **limits flexibility**.
- It is **impossible to involve all stakeholders** in the process, as large systems have a diverse set of them.

Chapter 4: Requirements Engineering

Question: What is requirements engineering? Differentiate between User Requirements and System Requirements with an example. **Answer:**

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

User vs. System Requirements:

- **User requirements:** Statements in natural language (plus diagrams) of the services the system provides and its operational constraints. They are written for customers.
- **System requirements:** A structured document setting out detailed descriptions of the system's functions, services, and operational constraints. It defines what should be implemented and may be part of a contract.

Example (from MHC-PMS):

- **User Requirement:** "The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month."
- **System Requirement (detailing the above):** "1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated." (and other details like 1.2, 1.3, etc.)

Question: Define and give examples of Functional, Non-functional, and Domain requirements. **Answer:**

- **Functional requirements:** Statements of services the system should provide, how it should react to inputs, or how it should behave in particular situations. They describe the system's functionality.
    - **Example (MHC-PMS):** "A user shall be able to search the appointments lists for all clinics."
- **Non-functional requirements:** Constraints on the services or functions offered by the system, such as timing constraints, standards, or constraints on the development process. They often apply to the system as a whole.
    - **Example (MHC-PMS):** "The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30)."
- **Domain requirements:** Constraints on the system that come from the domain of operation. If not satisfied, the system may be unworkable.
    - **Example (Train System):** A requirement defining the specific formula for train deceleration: "The deceleration of the train shall be computed as: Dtrain = Dcontrol + Dgradient..."

Question: How would you approach analyzing functional and non-functional requirements for a new hospital management system? Highlight key differences and provide examples for each.
Answer: Approach: The approach would be to use the **requirements elicitation and analysis process**. This involves:

1. **Requirements Discovery:** Interacting with stakeholders. For a hospital, this includes **patients, doctors, nurses, medical receptionists, and IT staff**. Techniques would include:
    - **Interviewing** stakeholders to understand their needs.
    - Developing **scenarios** for common tasks (e.g., admitting a patient, scheduling an appointment).
    - Creating **use cases** to identify all actors and their interactions with the system.
2. **Requirements Classification:** Grouping related requirements (e.g., all patient record requirements, all scheduling requirements).
3. **Prioritization and Negotiation:** Deciding which requirements are most important and resolving conflicts between stakeholders (e.g., a doctor's needs vs. a manager's needs).
4. **Requirements Specification:** Documenting the requirements.

Key Differences:

- **Functional requirements** define *what* the system should do (its services and functions).
- **Non-functional requirements** define *how well* the system should do it (its properties and constraints, like speed, reliability, or security).

Examples for a Hospital System (based on MHC-PMS):

- **Functional:**
    1. The system shall generate each day, for each clinic, a list of patients expected to attend appointments.
    2. Each staff member using the system shall be uniquely identified by their employee number.
    3. A user shall be able to search for a patient by family name.
- **Non-Functional:**
    1. **Availability:** The system shall be available during normal clinic working hours.
    2. **Security/Privacy:** The system shall implement patient privacy provisions as set out in relevant legislation.
    3. **Usability:** Medical staff shall be able to use all system functions after four hours of training.

Question: What is a Software Requirements Document (SRS)? Discuss its characteristics and structure. **Answer:** The **Software Requirements Document (SRS)** is the official statement of

what is required of the system developers. It should include both user requirements and a detailed specification of the system requirements.

Characteristics of a Good SRS: The requirements in an SRS should be checked for the following :

- **Validity:** Does it provide the functions that best support the customer's needs?
- **Consistency:** Are there any requirements conflicts?
- **Completeness:** Are all functions required by the customer included?
- **Realism:** Can the requirements be implemented given the available budget and technology?
- **Verifiability:** Can the requirements be checked (i.e., are they testable)?

Structure of an SRS: A typical SRS structure includes the following sections :

- **Preface:** Defines the expected readership and version history.
- **Introduction:** Describes the need for the system, its functions, and how it fits business objectives.
- **Glossary:** Defines technical terms used in the document.
- **User requirements definition:** Describes the services provided to the user (often in natural language).
- **System architecture:** A high-level overview of the anticipated architecture.
- **System requirements specification:** Detailed descriptions of the functional and non-functional requirements.
- **System models:** Graphical models (like use case diagrams) showing relationships.
- **System evolution:** Assumptions and anticipated changes.
- **Appendices:** Detailed information (e.g., hardware or database descriptions).
- **Index:** An index to help navigate the document.

Question: What is requirements discovery? Explain it using the MHC-PMS use case diagram.
Answer:

Requirements discovery is the process of gathering information about the required and existing systems and distilling the user and system requirements from this information. It involves interacting with a range of system **stakeholders**.

MHC-PMS Use Case Diagram Example: A use case diagram is a technique used in requirements discovery. The MHC-PMS use case diagram shows the system's boundaries, the actors (stakeholders) who interact with it, and the functions they perform.

- **Actors (Stakeholders) Identified:** The diagram identifies key stakeholders, such as:
    - **Medical Receptionist**
    - **Nurse**
    - **Doctor**

    - **Patient**
- **Functions (Requirements) Discovered:** The diagram shows the tasks each actor performs, which represent functional requirements:
    - A **Medical Receptionist** can "Register patient" and "View patient information".
    - A **Nurse** can "View patient information" and "Record patient consultation".
    - A **Doctor** can "Record patient consultation," "View patient history," and "Transfer data" (presumably to other systems or clinics). This diagram helps stakeholders and developers understand the scope of the system and all the interactions that must be supported.

Question: What is requirements validation? What five key checks are performed during this process? **Answer:**

Requirements validation is the process of demonstrating that the requirements define the system that the customer really wants. This is a critical process because fixing requirements errors after delivery can be extremely expensive.

The five key checks performed during validation are :

1. **Validity:** Does the system provide the functions that best support the customer's needs?
2. **Consistency:** Are there any requirements conflicts?
3. **Completeness:** Are all functions required by the customer included?
4. **Realism:** Can the requirements be implemented given the available budget and technology?
5. **Verifiability:** Can the requirements be checked (i.e., are they testable)?

Question: What is requirements management? With a neat diagram, describe the process of requirements change management. **Answer:**

Requirements management is the process of managing changing requirements during the requirements engineering process and system development. New requirements always emerge, and it's essential to keep track of individual requirements, maintain links between dependent requirements, and assess the impact of changes.

Requirements Change Management Process: This is the set of activities that assess the impact and cost of proposed changes. **Diagram: Process Steps:**

1. **Problem analysis and change specification:** The proposed change is analyzed to check if it is valid. This analysis is fed back to the requestor.
2. **Change analysis and costing:** The effect of the proposed change is assessed using traceability information. The cost of the change is estimated, and a decision is made on whether to proceed.

3. **Change implementation:** If the change is accepted, the requirements document, system design, and implementation are modified.

Chapter 4: Requirements Engineering (Additional)

Question: Discuss the generic requirement engineering activities with a neat diagram that are common to all process models. **Answer:** The Requirements Engineering (RE) process includes several generic activities that are common to all software process models. In practice, these activities are interleaved in an iterative, spiral-like process.

The generic activities are:

- **Requirements elicitation:** Interacting with stakeholders to discover their requirements.
- **Requirements analysis:** (Often grouped with elicitation) Involves classifying, organizing, prioritizing, and negotiating requirements.
- **Requirements validation:** Demonstrating that the requirements define the system the customer really wants.
- **Requirements management:** The process of managing changing requirements during development.

Diagram: This iterative nature, where the activities are repeated as understanding deepens, can be shown with a spiral view.

- The process starts with a feasibility study.
- It then cycles through the main activities:
    - Requirements elicitation/discovery.
    - Requirements classification and organization.
    - Requirements prioritization and negotiation.
    - Requirements specification (documenting the requirements).
- This spiral continues, producing documents from business requirements to user requirements, and finally to system requirements and specification.

Question: An online pharmacy application needs to be developed. Assuming the role of a requirement analyst, write the functional requirements for the above application. **Answer:** Functional requirements state the services the system should provide or how it should react to particular inputs. Based on the examples from the lecture slides , here are sample functional requirements for an online pharmacy:

1. A user **shall be able to** search for medicines by name, brand, or medical condition.
2. The system **shall require** all users to create an account or log in before placing an order.
3. A user **shall be able to** upload a valid doctor's prescription to purchase prescription-only medication.

4. The system **shall** verify the authenticity of an uploaded prescription before confirming an order.
5. The system **shall** maintain a shopping cart, allowing users to add, remove, and modify quantities of items.
6. The system **shall** provide an online payment gateway for processing credit card and other digital payments.
7. The system **shall** generate an order confirmation and send it to the user's registered email address.
8. A user **shall be able to** view their order history and track the status of current orders.
9. Each staff member (pharmacist, admin) **shall be uniquely identified** by their employee ID and password.
10. A pharmacist **shall be able to** review and approve orders containing prescription medication.