SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture 1: Introduction to Software Engineering, Ethics & SDLC Overview

Week 1 | IT2030 - Software Engineering

Lesson Learning Outcomes

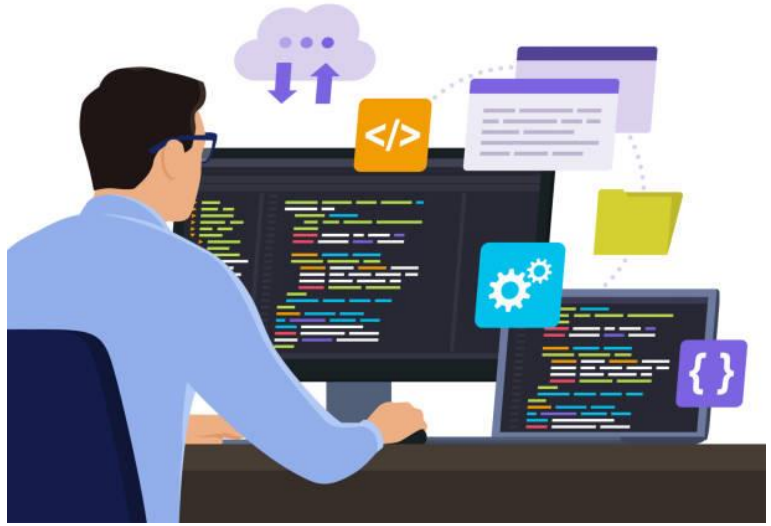
By the end of this lecture, students will be able to:

- Define software engineering and explain how it differs from traditional programming.
- Describe the main phases of the Software Development Life Cycle (SDLC).
- Explain the purpose and types of feasibility studies in software projects.
- Identify key software requirements and differentiate between functional, non-functional, and constraints.
- Recognize the importance of ethics in software engineering and the role of professional codes (IEEE/ACM).

What is Software Engineering (SE)?

Software Engineering (SE) is about applying **engineering principles** to build software that is:

- **Reliable**
- **Efficient**
- **Easy to maintain**



It focuses on:

- Following **structured processes**
- Ensuring **quality at every step**
- Planning across the entire **software lifecycle**

Why Software Engineering Matters

Good software engineering helps to:

- **Save time and money** in the long run
- **Protect users and ensure their satisfaction**
- **Avoid major failures** in critical systems

Real-world examples:

- Banking apps
- Hospital systems
- Flight control software

These systems **must** be built with care using proper engineering practices.

SE vs Programming

Aspect	Programming	Software Engineering
Focus	Writing code for specific tasks or features	Developing complete, reliable, and maintainable systems
Scope	Narrow, task-oriented	Broad, includes the whole software lifecycle
Activities	Coding, debugging	Analysis, design, architecture, coding, testing, deployment, maintenance
Teamwork	Often done individually	Requires collaboration and coordination in teams
Process	May be ad hoc or informal	Follows structured, well-defined processes (e.g., SDLC)
End Goal	Working code	Working, reliable, scalable, and user-validated software
Real-World Use	Useful for quick scripts or prototypes	Critical for large systems (e.g., banking, healthcare, aviation)

The Software Crisis

The Software Crisis (1960s–1990s)

During this time, many software projects were failed due to:

- **Over budget**
- **Missed deadlines**
- **System crashes and disasters**

Failure	What Happened	Why it Failed
Therac-25	Patients received massive radiation overdoses	Poor testing, missing safety checks
Ariane 5 Rocket	Exploded 40 seconds after launch	Reused code without proper adaptation
Mariner 1 (NASA)	Veered off course and self-destructed	A **missing hyphen** in code!
Knight Capital	Lost $440 million in 30 minutes	Wrong code update in stock trading bot

What is SDLC?

The **Software Development Life Cycle (SDLC)** is like a **roadmap** for building software the right way!

It helps teams:

- **Meet user needs** and expectations
- **Save time and cost** by planning ahead
- **Avoid risks** and delivery delays

Why Use It?

- Would you build a house without a blueprint? Same goes for software — **SDLC gives structure** to every phase from planning to maintenance.

Understanding SDLC Through Real Life

"Ordering a customized chair is like building a software – it all starts with understanding the need."

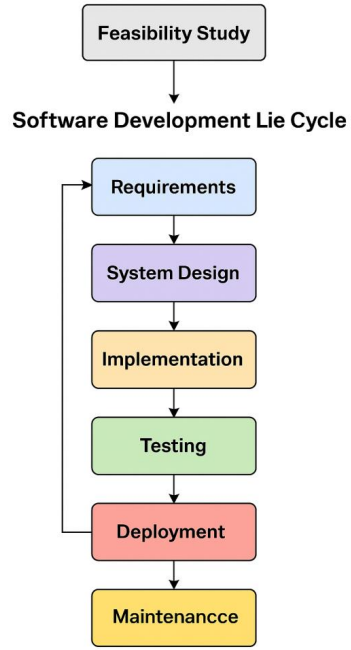
Feasibility Study

"Actually, I'm still on life support.
just came by to do a feasibility study."

What is a Feasibility Study?

A **feasibility study** is done **before starting** the actual software development process.

- It helps answer one big question:
  "Should we build this system?"



Purpose

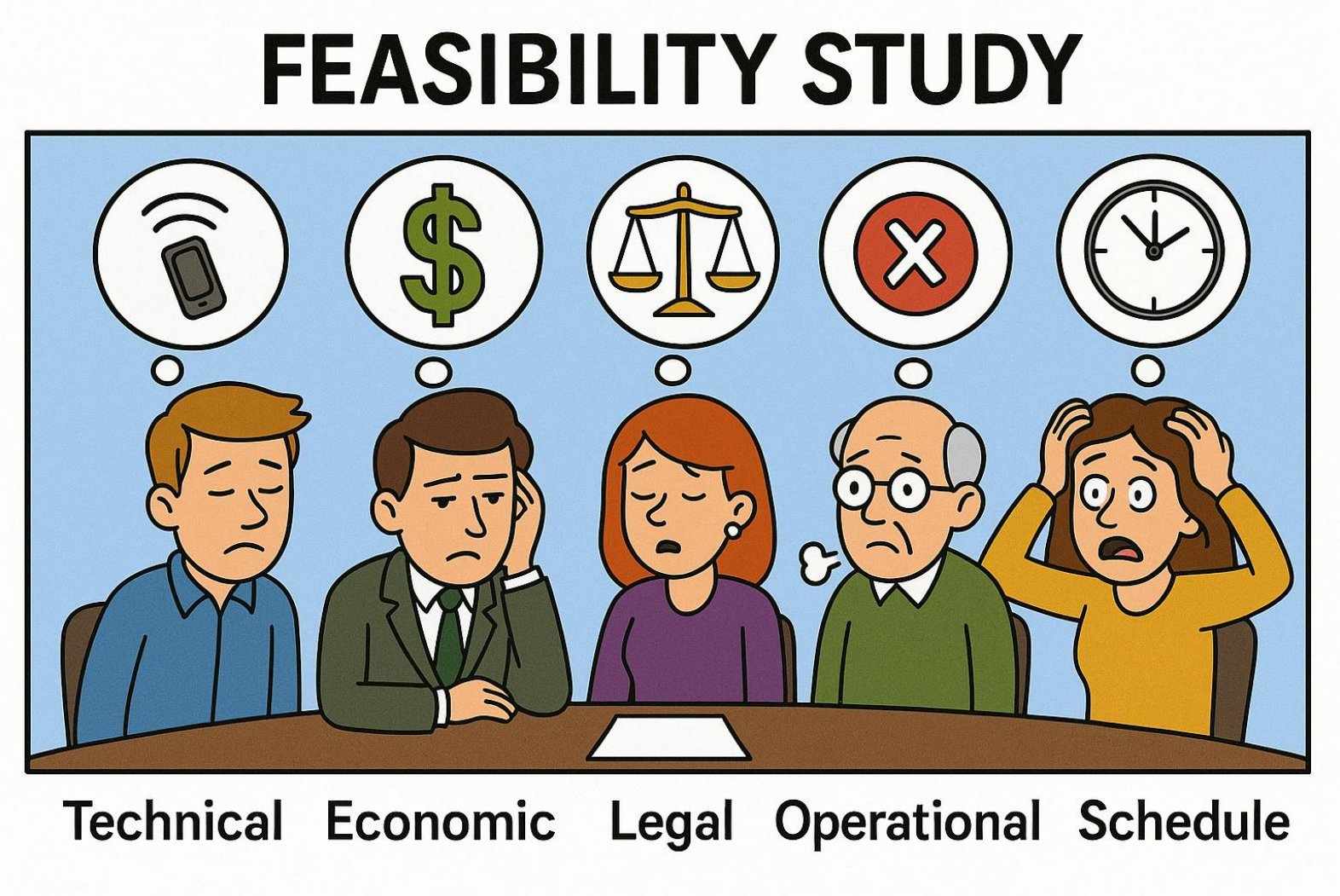
To decide whether the project is:

- **Technically possible**
- **Financially affordable**
- **Realistic**

Output:

- A **Feasibility Report** – used by managers or stakeholders to **approve or reject** the project.

Types of Feasibility



Types of Feasibility

1 Technical Feasibility

Can we build it with the technology we have?

- Do we have the required hardware, software, and skills?

Example: Does our current network and server setup support the new system?

2 Economic Feasibility

Is it worth the cost?

- Will the **benefits** outweigh the **expenses**?

Example: Will the time and money spent bring long-term value to the company?

Types of Feasibility

3 Legal Feasibility

Is the project legally safe?

- Are there any laws, policies, or regulations to follow?

Example: Does the system comply with **GDPR (**General Data Protection Regulation) or **data protection laws**?

4 Operational Feasibility

Will users actually use it?

- Will the system work in the real world?
- Will people accept and use it?

Example: Will staff feel confident adapting to the new system?

Types of Feasibility

5 Schedule Feasibility

Can we deliver it on time?

- Is the timeline realistic?
- Can we meet deadlines with available resources?

Example: Can the system go live **before the new school year** begins?

Activity: Feasibility Analysis

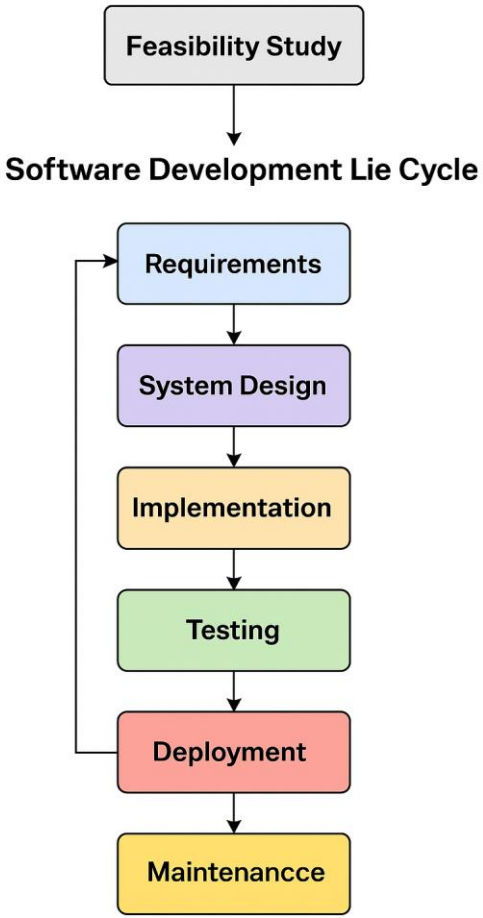
Scenario: Your team is designing a **mobile app** to help **rural farmers** access weather updates, market prices, and farming advices.

- For each type of feasibility, identify **one real concern** related to this project:

Introduction to SDLC Phases

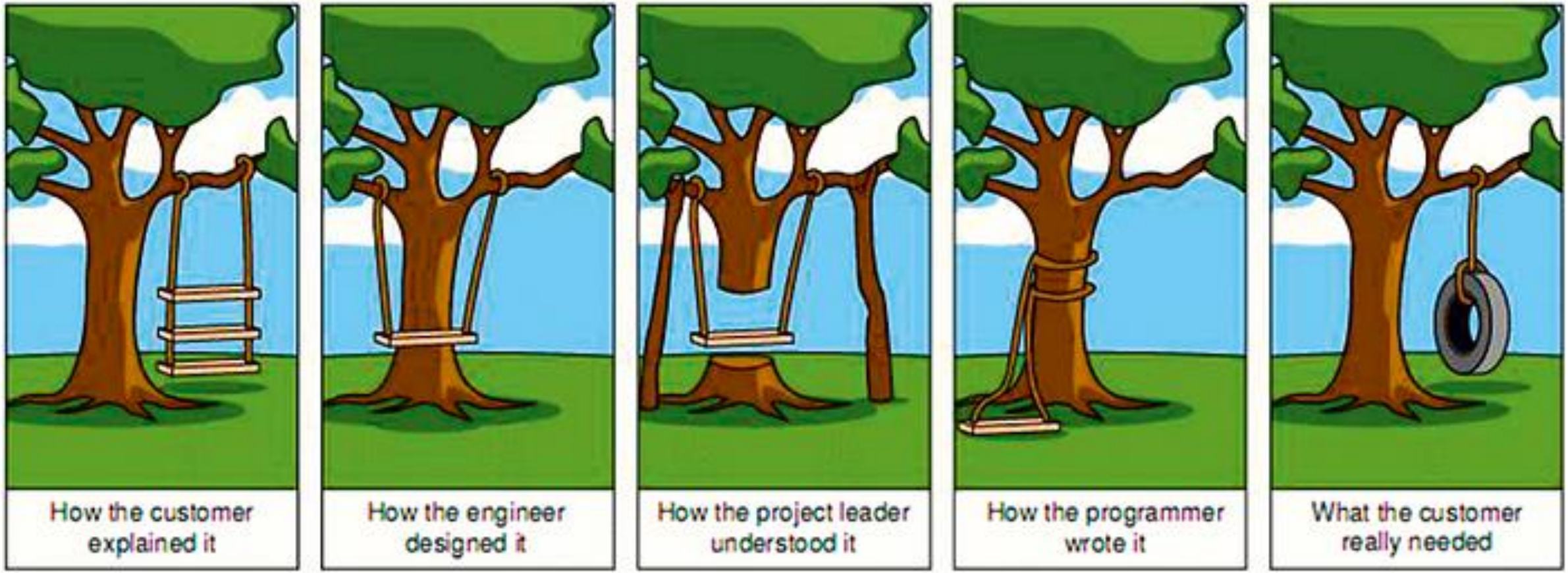
Key Phases:

1. Requirements gathering and Analysis
2. System design
3. Implementation
4. Testing
5. Deployment
6. Maintenance



Phase 1: Requirements Gathering & Analysis

- What is the story behind this image???



Phase 1: Requirements Gathering & Analysis

Requirements Gathering

- Find out what the **user or client** wants the system to do
- Write it down clearly in a **Software Requirements Specification (SRS)**

Who's Involved?

- Clients
- Users
- Business analysts

Example:

For a **school management system**, gather input from:

- Admin staff
- Teachers
- Students

Phase 1: Requirements Gathering & Analysis

Role of Requirements Gathering

- It happens **right after the feasibility study**
- Provides the **foundation for design, implementation, and testing**
- Acts like a **contract** between the client and the development team outlining what will (and won't) be built

Common Techniques for Gathering Requirements

- **Talking to users (interviews)**

One-on-one or small group conversations to understand what users need and expect from the system.

- **Asking questions (questionnaires)**

Use forms or online surveys to collect feedback or preferences from a larger group of users.

- **Watching how tasks are done (observations)**

Observe users doing their daily work to find pain points, inefficiencies, or system improvement needs.

- **Reviewing existing documents (document analysis)**

Study current reports, forms, or system manuals to understand existing workflows and rules.

- **Group discussions (workshops or focus groups)**

Bring users together to discuss their needs, agree on priorities, and clarify conflicts.

- **Showing examples (prototyping)**

Create mockups or sample screens to help users visualize the system and give early feedback.

- **Generating ideas (brainstorming)**

Collaborate with stakeholders to quickly come up with ideas, features, or solutions.

Types of Requirements

- **Functional Requirements**
  Describe **what** the system should do

Example: “The system should allow users to log in.”

- **Non-Functional Requirements**
  Describe **how** the system should perform

Example: “The system should load within 3 seconds.”

- **Limitations (Constraints)**
  Rules or boundaries the system must follow

Example: “The system must run on Android only.”

Example Breakdown – Student Attendance Management System

Example Breakdown – Student Attendance Management System

- **Functional Requirement:**
  The system should allow teachers to mark daily attendance for each student.
- **Non-Functional Requirement:**
  The system should be accessible on both desktop and mobile devices.
- **Limitation (Constraint):**
  The system must be deployed within the university's local network only.

Who are the Stakeholders?

What is a Stakeholder?

- A **stakeholder** is **any person or group** who has an interest in the software system —
  they may use it, manage it, affect it, or be affected by it.

Types of Stakeholders

End Users
People who directly use the system
Example: Students using a school app, customers using an e-commerce site
Internal Users
Staff within the organization who support or manage the system
Example: Admin staff, teachers, HR officers
External Partners
Third parties that the system interacts with
Example: Payment gateways, suppliers, delivery services
Regulatory Bodies
Authorities that set rules or standards the system must follow
Example: Education ministry, data protection authorities (GDPR)
Project Sponsor / Client
The person or group funding or owning the system
Example: University board, a government agency, company CEO

Best Practices in Requirement Gathering

- Involve all key stakeholders early in the process.

- Use clear, simple language (avoid technical jargon).

- Validate requirements with stakeholders using reviews or walkthroughs.

A **walkthrough** is a **peer review meeting** where the team goes through the software requirements, design, or code **step by step** to check for issues.

- Clearly document any assumptions and limitations.

- Maintain traceability to track requirements through design, development, and testing.

Common Mistakes in Requirement Gathering

- Guessing stakeholder needs

Assuming what users want without asking or confirming

- Using too much technical jargon

Writing requirements that non-technical users can't understand

- Not documenting assumptions or limitations

Leaving out important conditions or system boundaries

- Ignoring non-functional requirements

Forgetting things like speed, security, or usability

- Failing to update requirements

Not adjusting the requirements when project changes happen

Activity – Identify Requirements (5 minutes)

Think about a School Management System

Q1. Identify 2 Functional Requirements - What should the system do?

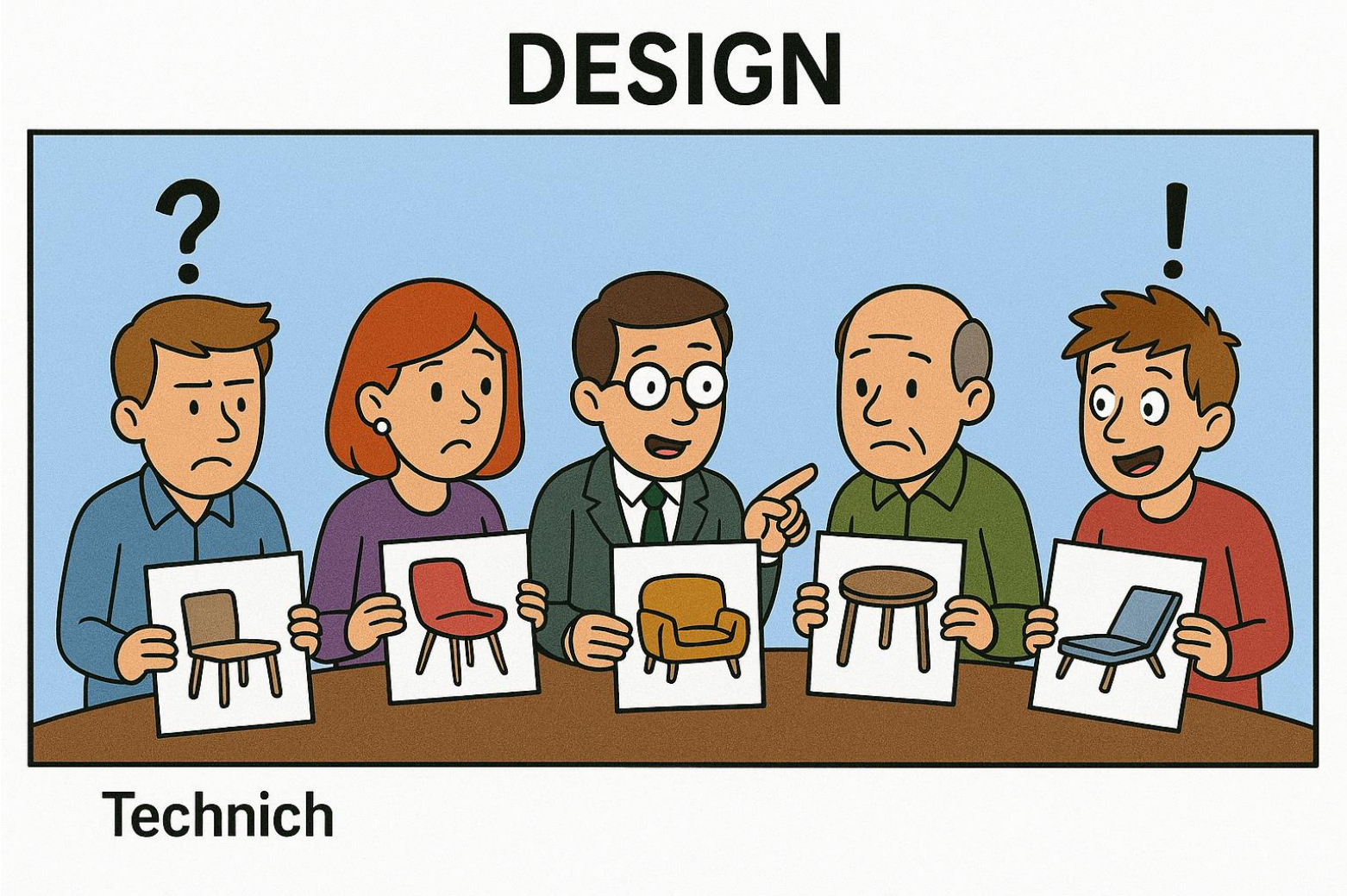
Q2. Identify 2 Non-Functional Requirements - How should the system behave?

Q3. List 2 Limitations (Constraints) - Any restrictions or boundaries?

Q4. Identify Stakeholders - Who is involved or affected?

Q5. Choose a Requirement Gathering Technique - Which method would help you collect this information?

Phase 2: System Design



Phase 2: System Design

What Happens in the Design Phase?

•This is where you **translate the SRS** (Software Requirements Specification) into a **system blueprint**

•You break down what the system should do into **design components**

What's Included:

•**UI Design** – Layout of screens and user flow

•**Database Schema** – Tables, relationships, and data structure

•**System Architecture** – How components (front-end, back-end, APIs) work together

Output:

•High-level design (overall structure)

•Low-level design (detailed logic and components)

Example:

Designing the **screens, database**, and **system structure** for a student registration portal

Phase 3: Implementation (Coding)

Phase 3: Implementation (Coding)

What Happens in the Implementation Phase?

- **Developers write the actual code** based on the design documents
- Follow **coding standards**, use the right **tools and technologies**
- The system is built **module by module** (e.g., login, registration, reports)

Each module is tested individually

- This is called **unit testing** — making sure each small part of the system works correctly on its own.

Example:

Coding the **login page**, including:

- Username/password fields
- Input validation (e.g., required fields, correct format)
- Error messages for invalid logins

Phase 4: Testing

Phase 4: Testing

What Happens in the Testing Phase?

- The team tests both **individual parts** (modules) and the **entire system**
- The goal is to **find and fix bugs**, and make sure the software works as expected based on the requirements

Types of Testing:

1. **Unit Testing** – Check if each small piece of code (like a function or module) works correctly
2. **Integration Testing** – Make sure different modules work well together
3. **System Testing** – Test the whole system to see if it behaves correctly
4. **Acceptance Testing** – Check if the system meets the **client's requirements**

Phase 4: Testing

Example:

Test the **student registration form** by entering:

- Valid details (e.g., correct name, ID, email)
- Invalid inputs (e.g., missing fields, wrong format)
  Check if the system responds properly.

Phase 5: Deployment

Phase 5: Deployment

What Happens in the Deployment Phase?

- The finished software is **installed in the real environment** where users will use it
- This step may include a **pilot release** (testing with a small group first)
- Users are **trained**, and **user manuals or help documents** are prepared

Support is set up:

- Help desks, technical support, or FAQs are provided to assist users during the transition

Example:

- Install the school management system on the **school's network server** so that admin staff and teachers can start using it

Phase 6: Maintenance

Phase 6: Maintenance

What is Software Maintenance?

- Maintenance begins **after the software is deployed**
- It includes **bug fixes**, **feature updates**, and **system improvements**
- The goal is to keep the system **functional, secure, and up to date**

Why It Matters:

Even good software needs changes over time due to:

- New user needs
- Technology updates
- Security risks

Example:

- Fixing a login issue reported by users, or adding a new report feature for admin staff.

Phase 6: Maintenance

- **Types of Software Maintenance**

1 Corrective Maintenance

- Fixes **bugs or errors** found after the system is deployed
  Example: Fixing a crash that happens when users try to log in

2 Adaptive Maintenance

- Changes the software to work in a **new environment**
  Example: Updating the app to support a new version of Android or iOS

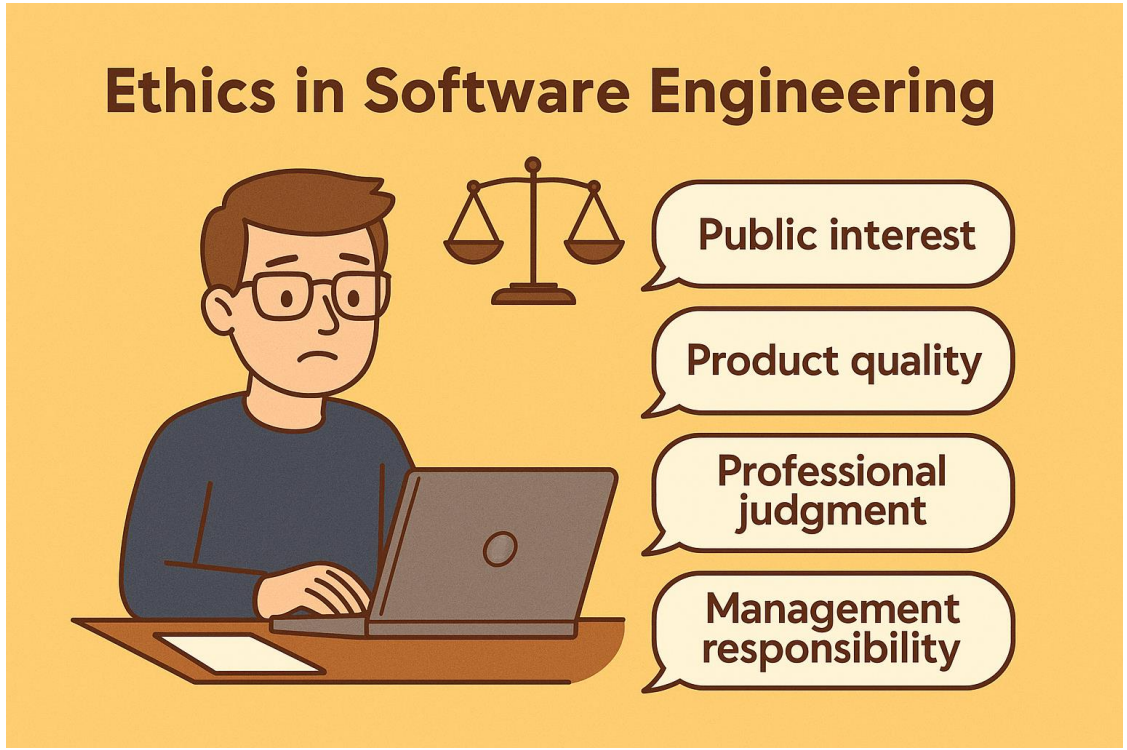
3 Perfective Maintenance

- Improves the system by **adding new features** or enhancing performance
  Example: Adding an online payment option to the school system

Maintenance keeps software **useful, updated, and problem-free** even after launch!

What is Ethics in SE?

Being a software engineer isn't just about writing code — it's about being responsible, fair, and professional in everything we do.



What is Ethics in SE?

What is Ethics in Software Engineering?

Ethics means the set of **moral values** that help us decide what is **right or wrong** in our actions.

In software engineering, ethics guide how professionals:

- Design
- Develop
- Test
- And use software systems

Key Ethical Values in SE:

- **Responsibility** – Take ownership of your work and its impact
- **Transparency** – Be honest about how the software works
- **Privacy** – Protect users' personal information
- **Fairness** – Avoid discrimination or bias in your system

What is Ethics in SE?

Example:

- Even if you can access user data, you **should not** share it without their **permission**. That's an ethical boundary, not just a technical one.

Codes of Ethics:

- International organizations like **ACM** and **IEEE** provide formal **Codes of Ethics** to help software engineers act responsibly.

IEEE Code of Ethics

IEEE (Institute of Electrical and Electronics Engineers)

- Global body for engineering, computing, and electronics
- Publishes technical standards (e.g., IEEE 802.11 for Wi-Fi)
- Focus: Hardware, electrical, and computer engineering
- Has its own IEEE Code of Ethics
- Last Updated 2020 (minor revision)

Access link:
https://www.ieee.org/about/corporate/governance/p7-8

ACM Code of Ethics and Professional Conduct

ACM (Association for Computing Machinery)

- World's largest society for computing professionals
- Focus: Computer science, education, software research
- Offers journals, conferences, and certifications
- Has its own ACM Code of Ethics
- Last updated in 2018.

Access link: https://www.acm.org/code-of-ethics

IEEE/ACM Code of Ethics

What is the IEEE/ACM Code of Ethics?

- A joint code created in 1999 specifically for Software Engineers
- Known as the "**Software Engineering Code of Ethics and Professional Practice**"
- Includes **8 core principles** (public, client, product, judgment, management, profession, colleagues, self)

Access link: https://www.acm.org/code-of-ethics/software-engineering-code

Other Popular Codes of Ethics for Software Engineers

While the **IEEE/ACM Software Engineering Code of Ethics** is widely recognized, several other respected professional bodies offer ethical guidelines relevant to software engineering:

- **BCS Code of Conduct (UK)**
  - From the British Computer Society
  - Focus on public interest, professional competence, and accountability
- **Australian Computer Society (ACS) Code of Ethics**
  - Tailored for ICT professionals in Australia
  - Includes privacy, respect, social impact
- **IFIP Code of Ethics**
  - From the International Federation for Information Processing
  - Promotes ethical computing globally

Activity: Ethical Dilemma Discussion

- Scenario: You are a software engineer. The management told you to release a product with known bugs to meet a deadline.
- Group discussion:
  - Which code/s you follow?
  - As a professional software engineer, what do you have to do?

Lecture 1 - Summary

What You Learned Today:

- **What is Software Engineering?**
  Applying engineering principles to build reliable, maintainable software.
- **Why SE Matters:**
  Prevents failures, saves time and cost, ensures safety and satisfaction.
- **SE vs Programming:**
  SE is process-driven; programming is just one part of it.
- **Software Crisis:**
  Many early projects failed due to lack of structure → need for SDLC.

Lecture 1 – Summary cont...

Overview of SDLC Phases:

1. **Feasibility Study** – Should we build it? (Check technical, financial, legal, etc.)
2. **Requirements Gathering** – What does the client/user want?
3. **System Design** – Blueprint of the system (UI, database, architecture)
4. **Implementation** – Writing code (module by module)
5. **Testing** – Checking correctness, integration, and client acceptance
6. **Deployment** – Delivering and launching the software in real life
7. **Maintenance** – Ongoing bug fixes, updates, and improvements

Lecture 1 – Summary cont...

Requirement Gathering:

- Techniques: Interviews, observations, questionnaires, workshops, prototyping
- Types of Requirements:
  - *Functional* (what system should do)
  - *Non-functional* (how it should behave)
  - *Limitations* (technical or legal boundaries)
- Stakeholders: End users, internal staff, external partners, sponsors

Lecture 1 – Summary cont...

Software Engineering Ethics:

- Importance of acting responsibly, honestly, and safely
- Key Principles: Privacy, fairness, transparency, responsibility
- Codes: **IEEE**, **ACM**, **IEEE/ACM Joint Code**, **BCS**, **ACS**

References

- Sommerville, I. (2016). *Software.Engineering* (10th ed.). Pearson Education.

Chapter 1: Introduction to Software Engineering

  - Chapter 2: Software Processes
  - Chapter 3: Agile Software Development
  - Chapter 23: Professional Responsibility (Ethics)

- Pressman, R. S., & Maxim, B. R. (2014). *Software.Engineering¿A.Practitioner's. Approach* (8th ed.). McGraw-Hill.
  - Chapter 1: The Nature of Software
  - Chapter 2: Process Models
  - Chapter 3: Agile Development
  - Chapter 4: Prescriptive Process Models
  - Chapter 19: Software Quality Assurance (includes ethics aspects)

References

- IEEE-CS/ACM Joint Task Force. (1999). Software Engineering Code of Ethics and Professional Practice, Version 5.2.
- ACM. (2018). ACM Code of Ethics and Professional Conduct.
- IEEE. (2020). IEEE Code of Ethics.
- Australian Computer Society (ACS). Code of Ethics.
- British Computer Society (BCS). Code of Conduct.
- Leveson, N. (1995). Medical Devices: The Therac-25. In Safeware: System Safety and Computers. Addison-Wesley.
- SWEBOK. (2014). Guide to the Software Engineering Body of Knowledge (Version 3.0).
    - Chapter 1: Software Requirements
    - Chapter 4: Software Construction
    - Chapter 10: Software Engineering Professional Practice

Thank You!!!