

# Lecture - 2: Introduction to Software Engineering

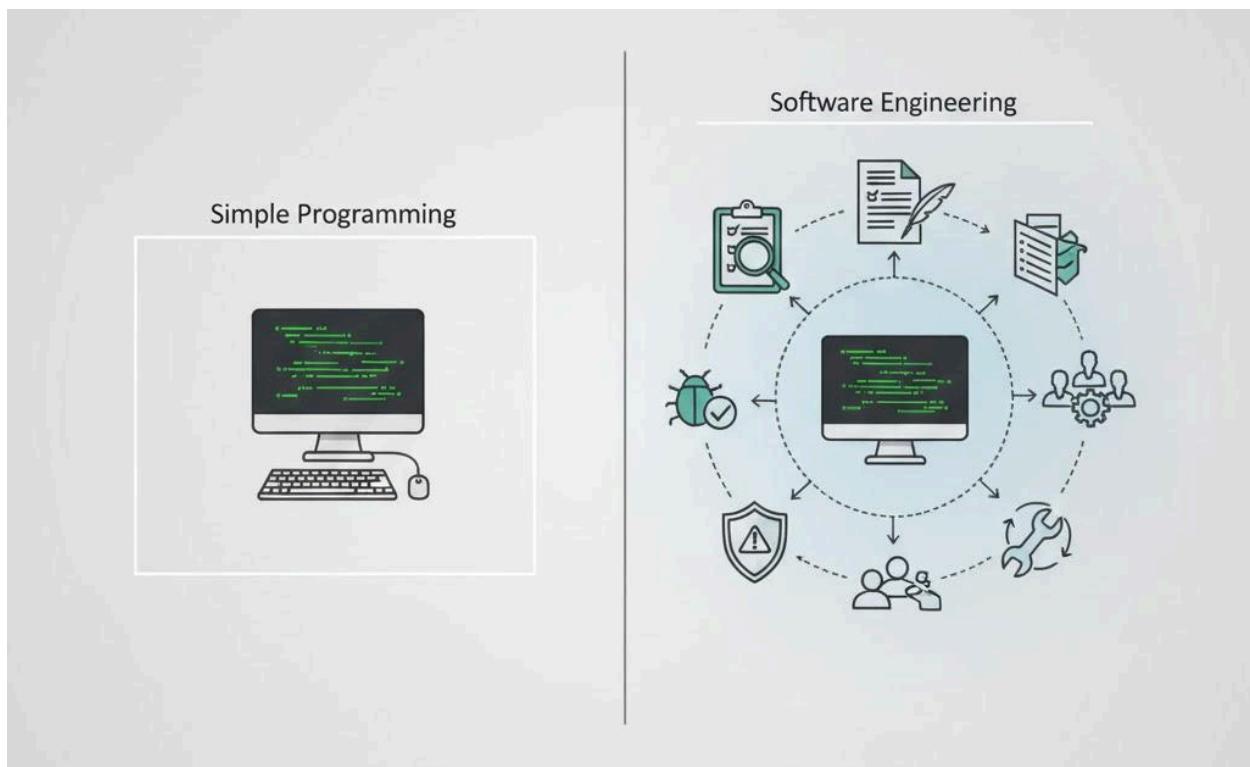
Software Engineering is a **systematic, disciplined, and organized approach** to the **design, development, testing, deployment, and maintenance of software systems**. It applies **engineering principles** to software creation in order to build systems that are **reliable, scalable, efficient, and maintainable**.

Unlike simple programming, Software Engineering focuses not only on **writing code**, but also on **planning, documentation, testing, teamwork, risk management, and long-term maintenance**.

In simple words:

*Programming is about making software work.*

*Software Engineering is about making software work correctly, efficiently, and for a long time.*



## 2. Formal Definition of Software Engineering

According to IEEE:

**Software Engineering is the application of a systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software.**

This definition highlights three important ideas:

1. **Systematic** – follows structured steps
2. **Disciplined** – follows rules, standards, and best practices
3. **Quantifiable** – quality, cost, time, and performance are measurable

	<b>Systematic</b> – Structured Steps
	<b>Systematic</b> – uses defined processes
	<b>Disciplined</b> – adheres to rules, standards, and best practices – ensures consistency
	<b>Quantifiable</b> – Metrics & Data – data-driven decisions

### 3. History of Software Engineering

#### 3.1 Early Days (Before 1960)

- Computers were rare and expensive
- Software was small and written by hardware engineers
- No formal development process
- Programs were simple and short-lived

##### Example:

Early scientific calculation programs written only for specific machines.

#### 3.1 Early Days (Before 1960)

The diagram illustrates the early days of software engineering through three main sections. The top section shows a smartphone-like device with a gear icon, connected by a dotted arrow to a person wearing a hard hat with a gear icon above their head, symbolizing the transition from hardware to software. The middle section features a document icon with a large red X over it, labeled 'No formal development process'. To its right is a clock icon with a document icon inside, labeled 'short-lived', representing the lack of a structured process and the temporary nature of early programs. The bottom section shows a computer monitor displaying a mathematical formula ( $\Sigma = \sqrt{\frac{x}{2}}$ ) connected by a line to a small robot-like device labeled 'specific machine', representing the creation of programs for specific scientific calculators or machines.

**Example:**

A computer monitor displays a mathematical formula ( $\Sigma = \sqrt{\frac{x}{2}}$ ). A line connects the monitor to a small robot-like device labeled 'specific machine', representing a program designed for a specific scientific calculator or machine.

## 3.2 The Software Crisis (1960s–1970s)

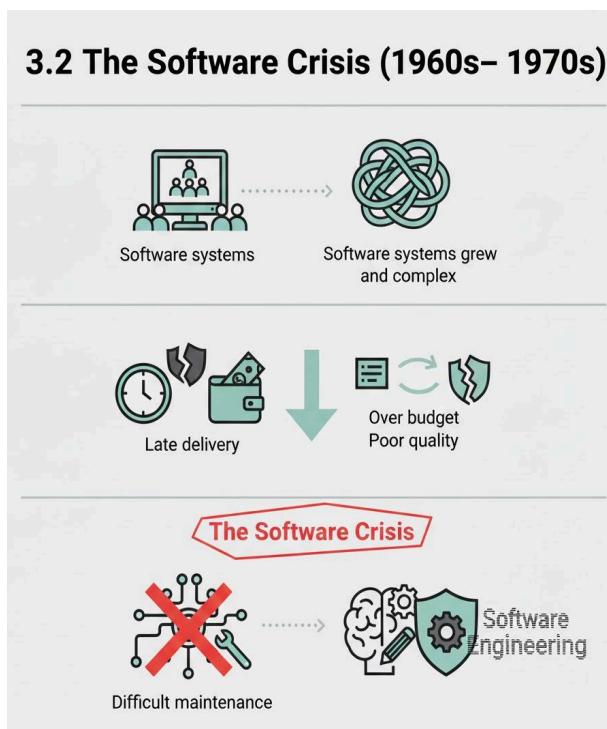
As computers became popular:

- Software systems grew **large and complex**
- Projects started **failing frequently**
- Common problems:
  - Late delivery
  - Over budget
  - Poor quality
  - Difficult maintenance

This period was called the **Software Crisis**.

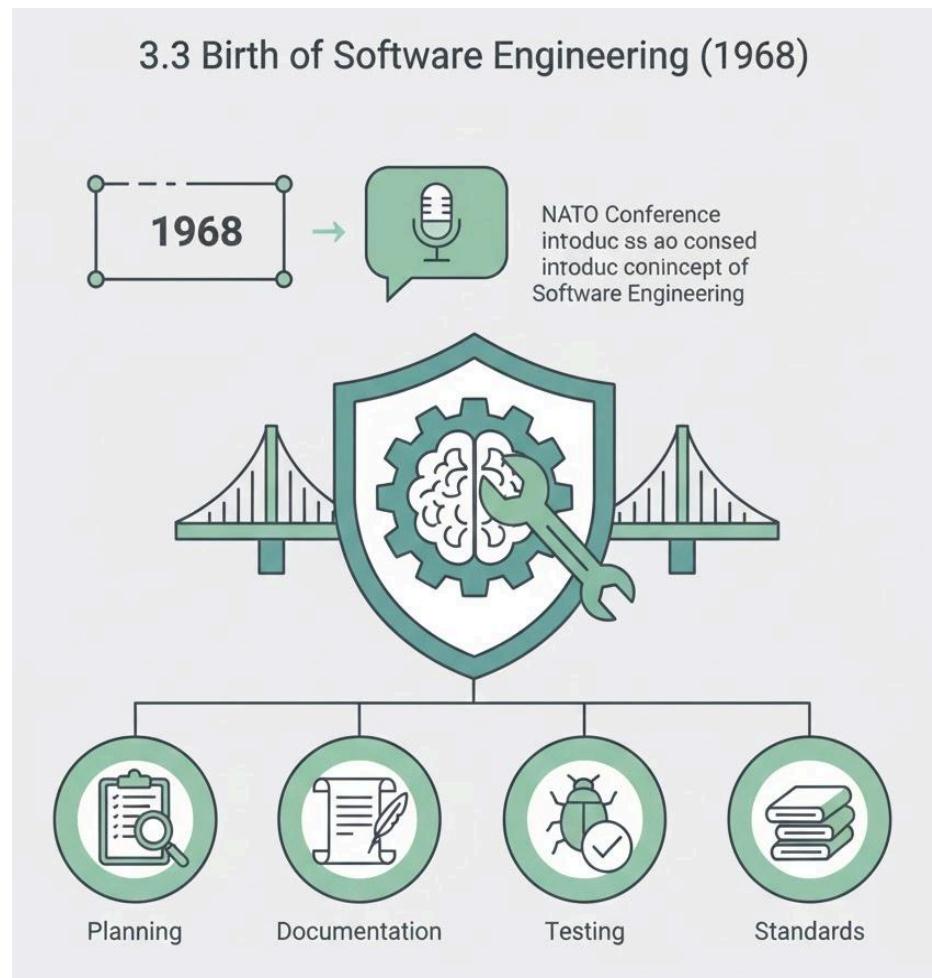
### Real-World Example:

Many government and defense projects failed because software became too complex to manage without proper planning. This crisis led to the birth of **Software Engineering** as a discipline:



### 3.3 Birth of Software Engineering (1968)

- The term “**Software Engineering**” was first introduced at a NATO conference in 1968
- Goal: Treat software development like **civil or mechanical engineering**
- Emphasis on:
  - Planning
  - Documentation
  - Testing
  - Standards

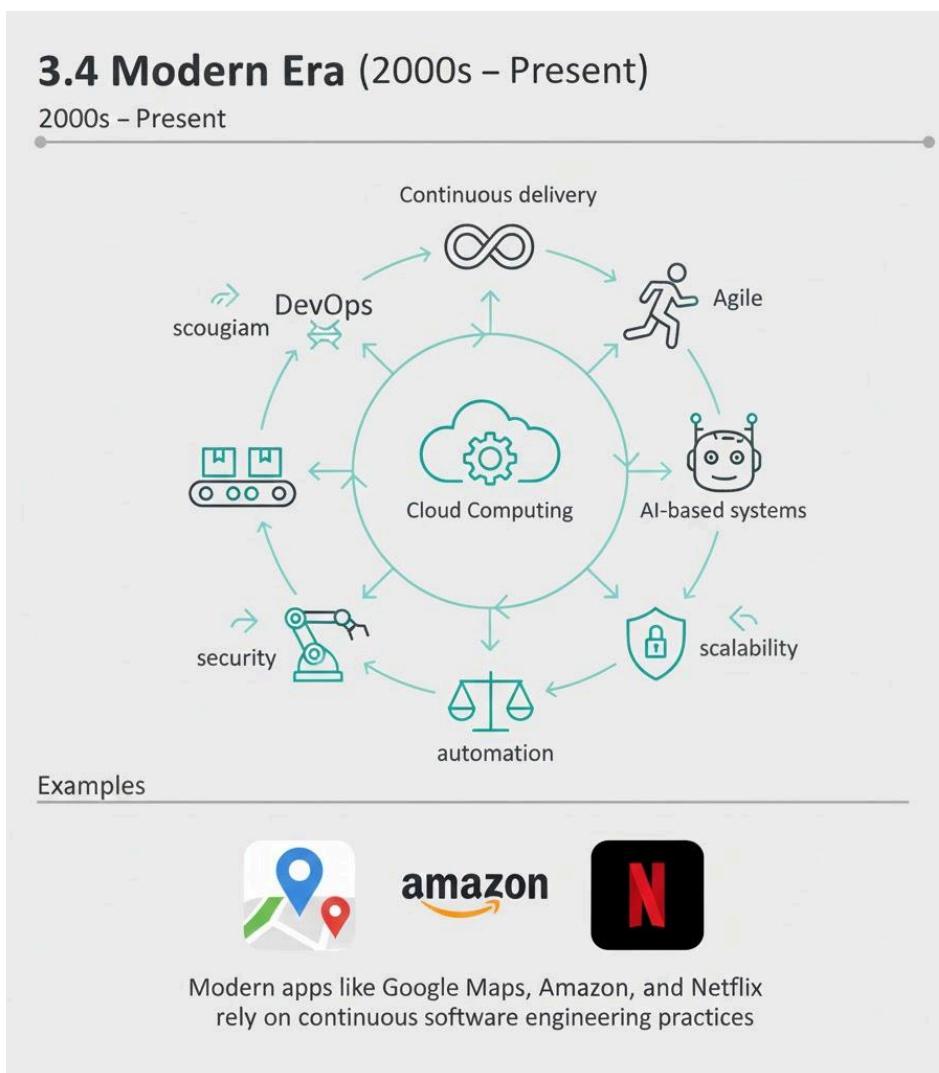


### 3.4 Modern Era (2000s – Present)

- Agile, DevOps, Cloud Computing
- AI-based systems
- Continuous delivery and automation
- High focus on security, ethics, and scalability

#### Example:

Modern apps like Google Maps, Amazon, and Netflix rely on **continuous software engineering practices**.

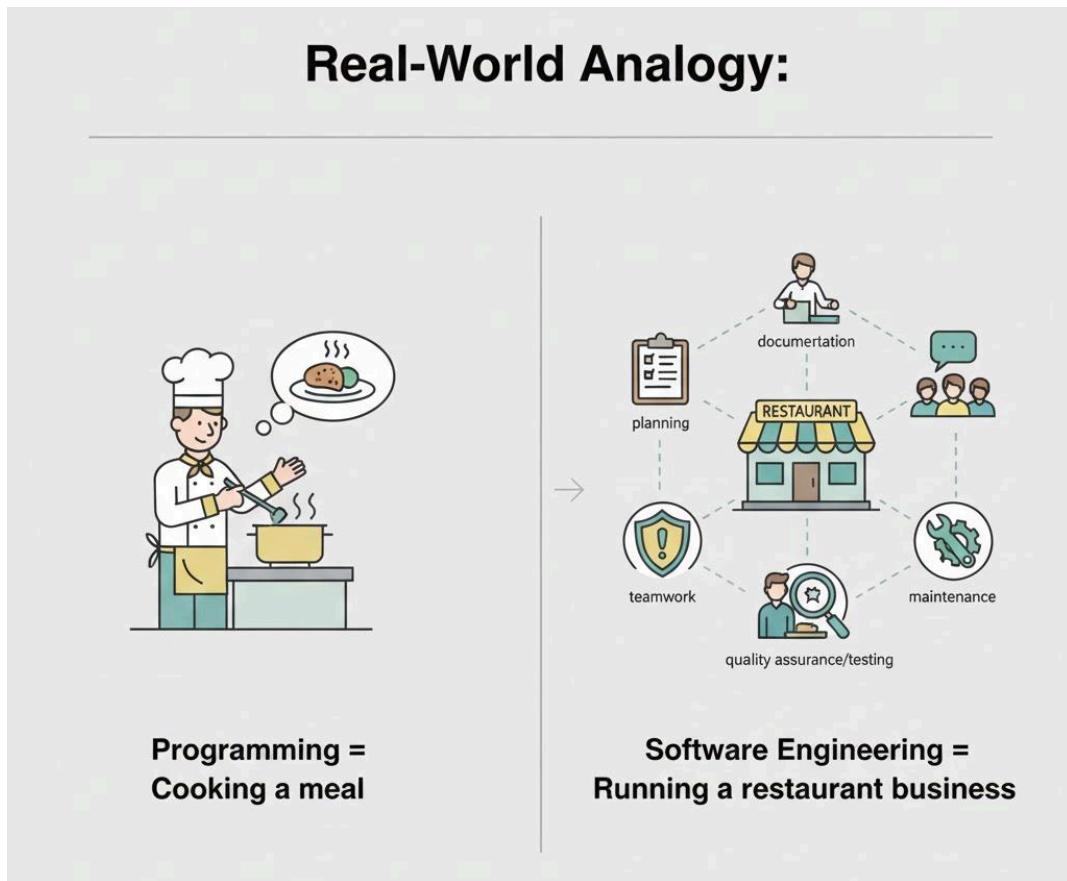


## 4. Software Engineering vs Programming

Programming	Software Engineering
Writing code	Complete development process
Individual focus	Team-based approach
Short-term solution	Long-term maintenance
Less documentation	Detailed documentation
Small applications	Large, complex systems

### Real-World Analogy:

- Programming = Cooking a meal
- Software Engineering = Running a restaurant business



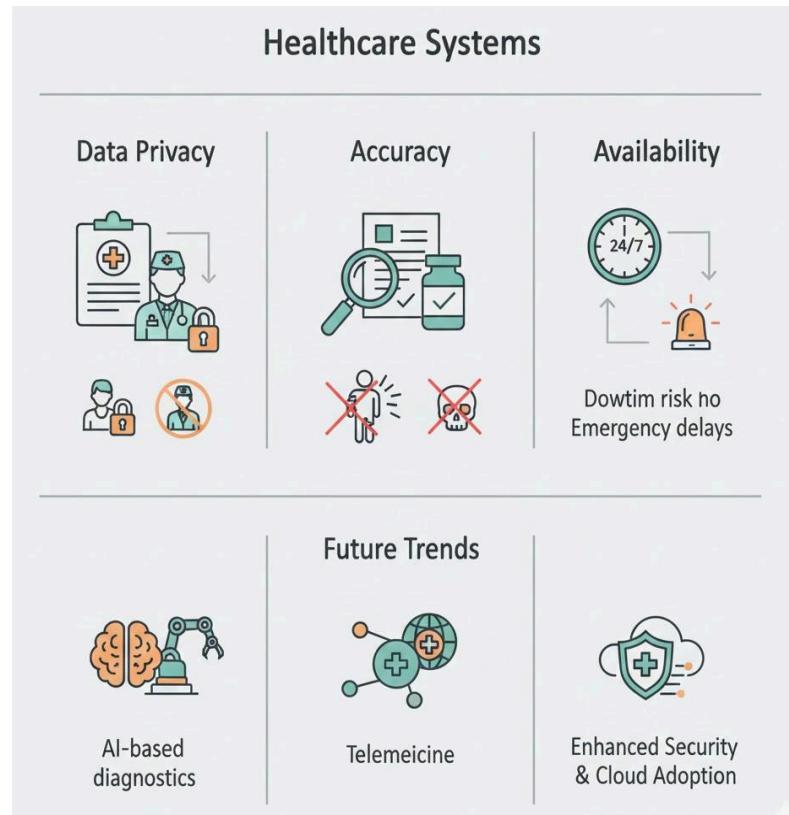
## 5. Software Engineering in Real Corporate Systems

## **Banking Systems:**

- **Transaction Consistency**
  - Ensures money is neither lost nor duplicated
  - Uses ACID properties (Atomicity, Consistency, Isolation, Durability)
  - Example: Fund transfer must debit and credit correctly
- **Security**
  - Encryption of sensitive data (PIN, passwords, account numbers)
  - Secure authentication and authorization
  - Protection against fraud and cyber-attacks
- **Audit Logs**
  - Every transaction is recorded
  - Used for tracking, investigations, and compliance
  - Mandatory for financial transparency
- **Regulatory Compliance**
  - Must follow RBI, PCI-DSS, GDPR guidelines
  - Software must enforce legal rules automatically

## **Healthcare Systems:**

- **Data Privacy**
  - Patient data must remain confidential
  - Access restricted to authorized medical staff only



- **Accuracy**

- Correct diagnosis reports and prescriptions
- Software errors can risk human life

- **Availability**

- Systems must be available 24/7
- Downtime can delay emergency treatment

## E-Commerce Platforms:

- **High Traffic Handling**

- Handles millions of users simultaneously
- Uses load balancing and scalable architecture

- **Payment Security**
  - Secure online transactions
  - Prevention of payment fraud
- **Recommendation Engines**
  - Personalized product suggestions
  - Improves customer experience and sales



## 6. Role of Software Engineers in Companies

### Corporate Expectations

- Understand **business requirements**, not just code
- Design **scalable and maintainable systems**
- Write **clean, readable, and testable code**

- Collaborate with:
    - Product managers
    - QA teams
    - Designers
    - DevOps engineers
  - Follow:
    - Coding standards
    - Documentation practices
  - Consider:
    - Performance optimization
    - Security risks
- 

## What Companies Do NOT Expect

- Just making the code work
  - Writing unstructured or hard-to-maintain code
  - Ignoring scalability and security
- 

## 7. Software Engineering as a Career Skill

### Why Companies Value Software Engineering Mindset

- Reduces **long-term project risk**
- Improves **software quality**

- Enables **faster feature development**
- Supports **large team collaboration**
- Makes systems easier to upgrade and maintain