Principle of Software Engineering

Introduction – Lecture 1

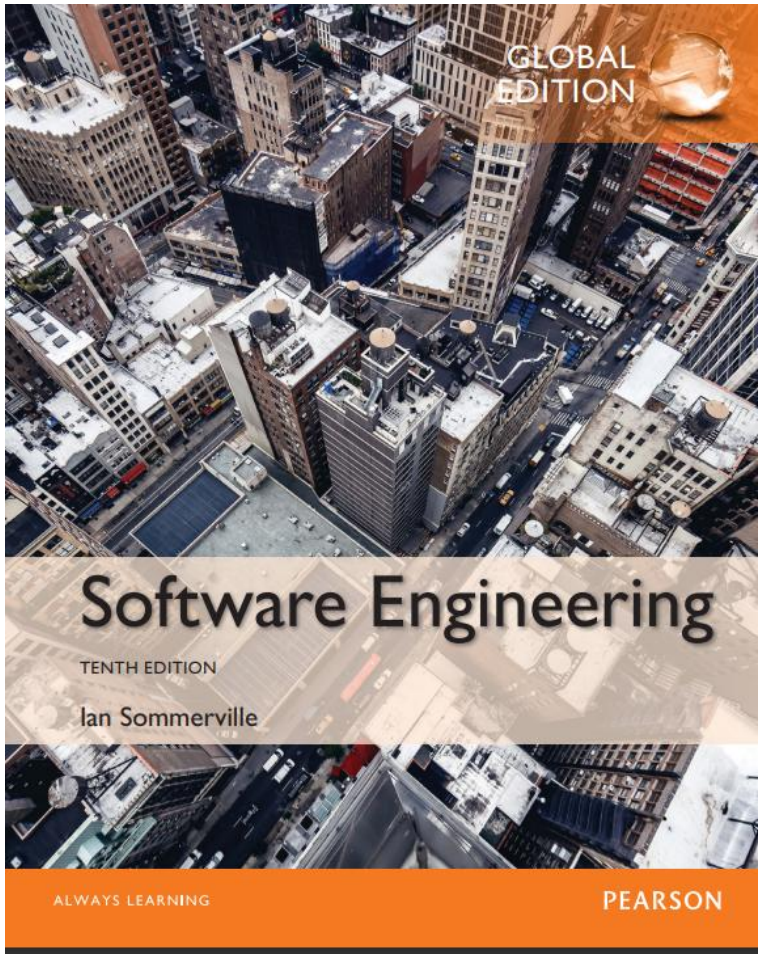
Week 01

Grading

- Midterm: 25%
- Final: 50%
- Programming Projects/Assignments: 10%
- Homework and Quizzes: 10%

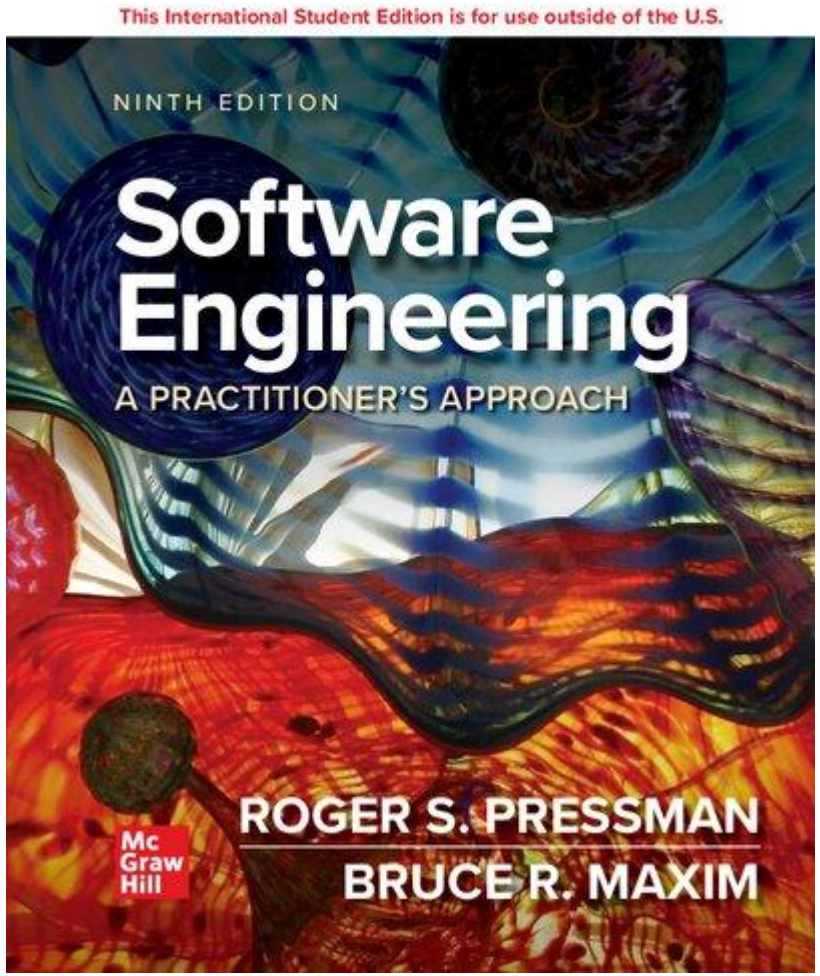
Full Attendance 5 Marks

Text Book



Title: Software Engineering Tenth Edition
Author(s): Ian Sommerville
Publisher: Pearson Publishers
Year: 2016
ISBN: 978-1-292-09613-1

Text Book



Title: Software Engineering (9th Ed)

A PRACTITIONER'S APPROACH

Author(s): Roger S. Pressman, Bruce R. Maxim

Publisher: McGraw-Hill Education

Year: 2019

ISBN: 9781260548006,1259872971

Focus of the Course

- This is a **Concept** based course of Software Engineering.
- Software Engineering Processes
- SDLC
- Phases
- Development Models

Topics to be discussed

- Recommended Book
- Introduction
- Software, categories of software
- Software Engineering
- Computer Science Vs Software Engineering
- Software Engineering [Layers]
- Software Engineer ethical Responsibilities
- Conclusion

What is Software?

- **Software** is a collection of **instructions, data, and programs** that tell a computer how to perform specific tasks. Unlike hardware, which is the physical part of a computer, **software is intangible** and enables the hardware to function.

What is Software?

- Software is a set of items:
  - Programs:
    - code
  - Data:
    - the data on which the program operates
  - Documents:
    - All documentation associated with the program and its use.

Types of Software

Software is broadly classified into two main categories:

1. **System Software**
2. **Application Software**

Additionally, there are other specialized types of software like **Programming Software, Middleware, and Firmware.**

1. System Software

- System software manages the computer's hardware and provides a foundation for running applications. It operates in the background and ensures system stability and performance.
- **Types of System Software:**

- **Operating System (OS)** – Manages computer resources (e.g., Windows, macOS, Linux).
- **Utility Programs** – Perform maintenance tasks (e.g., Antivirus, Disk Cleanup, Backup Software).
- **Device Drivers** – Allow the OS to communicate with hardware (e.g., Printer drivers, GPU drivers).

📌 **Example:** When you plug in a new printer, the device driver helps the OS recognize and use it.

2. Application Software

- Application software is designed for **end users** to perform **specific tasks**. It is installed on **top of system** software.
- **Types of Application Software:**
- **Productivity Software** – Microsoft Word, Excel, PowerPoint.
- **Web Browsers** – Google Chrome, Mozilla Firefox, Safari.
- **Multimedia Software** – Adobe Photoshop, VLC Media Player.
- **Business Software** – ERP, QuickBooks, CRM systems.
- **Entertainment Software** – Games, Streaming Apps (Netflix, Spotify).

3. Programming Software

Programming software helps developers create, debug, and maintain applications.

- **Examples:**
- **Compilers** – Convert code into machine language (e.g., C++).
- **Text Editors** – Used for writing code (e.g., Notepad++, Sublime Text).
- **Debuggers** – Identify and fix coding errors (e.g., GDB, Xcode).

4. Middleware

Middleware acts as a **bridge** between different software applications or systems, ensuring smooth communication.

- **Examples:**
- **Database Middleware** – Helps applications interact with databases (e.g., MySQL Connector).

5. Firmware

Firmware is **pre-installed software** embedded in hardware devices, providing low-level control. Unlike regular software, firmware is stored in **ROM (Read-Only Memory)** and is not easily changed.

- **Examples:**
- **BIOS** – Controls the startup process of a computer.
- **Smartphone Firmware** – Runs on Android and iOS devices.
- **Embedded Systems Firmware** – Found in smart TVs, routers, and medical devices.

Summary

Type of Software	Purpose	Examples
System Software	Manages hardware and system operations	Windows, Linux, macOS, Device Drivers
Application Software	Performs specific user tasks	Microsoft Office, Web Browsers, Media Players
Programming Software	Helps developers write and debug programs	Python, Java, C++ IDEs, Compilers
Middleware	Connects different applications and systems	Apache Kafka, MySQL Connector
Firmware	Embedded software for hardware control	BIOS, Router Firmware, Embedded Systems

Generic Software

Generic Software refers to software that is designed for a broad range of users and purposes rather than being customized for a specific organization or individual. It is mass-produced and available for public use, usually without modifications.

Examples:

1. Microsoft Office (Word, Excel, PowerPoint), Google Docs, LibreOffice.
2. Multimedia Software
3. Enterprise Resource Planning (ERP) Software

Bespoke Software Engineering

Bespoke software (also called **custom software**) refers to software that is specifically designed and developed for an individual, organization, or business based on their unique requirements. Unlike **generic software**, which serves a broad audience, bespoke software is tailor-made to fit specific needs.

Example:

1. **Banking Systems** 
2. **Healthcare Management Software** 
3. **E-Commerce Platforms** 
4. **Enterprise Resource Planning (ERP) Systems**

Software Engineering vs. Computer Science

Software Engineering (SE) and Computer Science (CS) are closely related fields in computing but have distinct focuses. While **Computer Science** is more theoretical and focuses on computing principles, **Software Engineering** is more practical and focuses on designing, developing, and maintaining software systems.

1. Definition

- **Software Engineering (SE):**
- The application of engineering principles to **design, develop, test, and maintain software** systems.
- Focuses on **software lifecycle**, team collaboration, and large-scale system development.
- **Computer Science (CS):**
- The study of **computational theory, algorithms, programming languages, and data structures**.
- Deals with **mathematics, logic, and problem-solving** to create new computing technologies.

Comparison Table

Feature	Software Engineering	Computer Science
Focus	Practical application of computing to build software	Theoretical concepts behind computing
Development Process	Uses SDLC, Agile, DevOps	Research and problem-solving
Project Type	Large-scale software applications	Computing algorithms, AI, and security
Mathematics Requirement	Moderate	High
Career Options	Software Development, DevOps, Testing	AI, Cybersecurity, Research
Primary Goal	Build and maintain reliable software systems	Explore and advance computing theory

Software Engineering vs. System Engineering

Software Engineering (SE) and System Engineering (SysE) are related fields, but they have different scopes and responsibilities. While **Software Engineering** focuses on developing software applications, **System Engineering** deals with the design, integration, and management of complex systems that may include **hardware, software, networks, and human interactions**.

Definition

1. **Software Engineering (SE):**

The application of engineering principles to **design, develop, test, and maintain software**.

Focuses on software development processes, programming, and quality assurance.

2. **System Engineering (SysE):**

The process of **designing, integrating, and managing** complex systems that involve **hardware, software, processes, and people**.

Focuses on **the entire system lifecycle**, from concept to operation.

Real-World Examples

Software Engineering Examples:

- Developing **mobile apps** (e.g., WhatsApp, Instagram)
- Building **operating systems** (e.g., Windows, Linux)
- Creating **enterprise software** (e.g., Microsoft Office, Salesforce)

System Engineering Examples:

- Designing **autonomous vehicle systems** (Tesla, Waymo)
- Managing **aerospace systems** (NASA, Boeing)
- Developing **telecommunication networks** (5G, IoT systems)

Software Engineering Layers

Software engineering is a fully layered technology, to develop software we need to go from one layer to another. All the layers are connected and each layer demands the fulfillment of the previous layer

1. **Quality Focus**
2. **Process Layer**
3. **Methods Layer**
4. **Tools Layer**

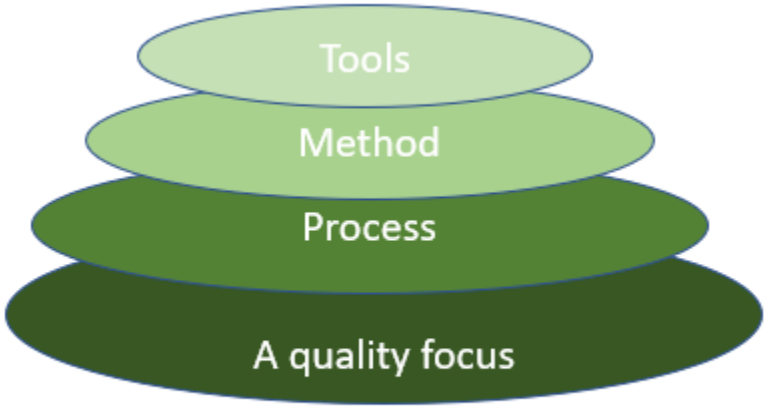


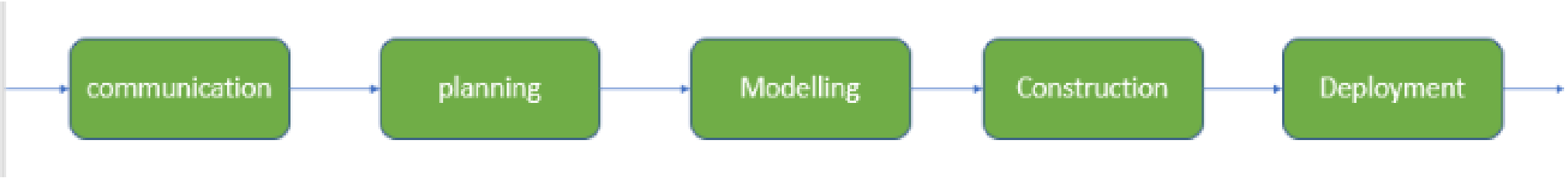
Fig: The diagram shows the layers of software development

1. Quality Focus (Foundation Layer)

A quality focus: It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.

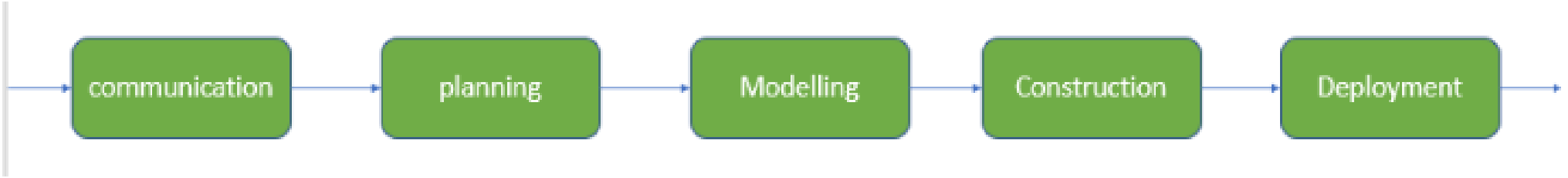
2. Process Layer 🔄

Process: It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time. Process defines a framework that must be established for the effective delivery of software engineering technology. The software process covers all the activities, actions, and tasks required to be carried out for software development.



2. Process Layer 🔄

Communication: It is the first and foremost thing for the development of software. Communication is necessary to know the actual demand of the client.
Planning: It basically means drawing a map for reduced the complication of development.
Modeling: In this process, a model is created according to the client for better understanding.
Construction: It includes the coding and testing of the problem.
Deployment:- It includes the delivery of software to the client for evaluation and feedback.



3. Methods Layer 📜

Method: During the process of software development the answers to all **“how-to-do”** questions are given by method. It has the information of all the tasks which includes communication, **requirement analysis, design modeling, program construction, testing, and support**.

4. Tools Layer

Tools: Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

Summary of Software Engineering Layers

Layer	Purpose	Examples
Quality Focus	Ensures high-quality, reliable software	Code reviews, testing, compliance
Process Layer	Defines how software is developed	SDLC, Agile, DevOps
Methods Layer	Provides techniques for development	OOP, system design, testing
Tools Layer	Supports development with automation	Git, Docker, Selenium

Professional and Ethical Responsibilities of Software Engineers

Software engineers have a **critical role** in the development of software that is reliable, safe, and secure. This responsibility extends beyond technical skills to ensuring **ethical behavior** and maintaining **professional integrity** in all aspects of their work.

1. Professional Responsibilities

1.1 Delivering High-Quality Software
Ensure that software products meet the required **quality, performance, and security** standards.
Follow established **software engineering best practices**.

Example: A software engineer should adhere to **coding standards** and conduct rigorous **testing** to ensure that the software performs efficiently.

1.2 Commitment to Continuous Learning
Keep skills up-to-date with the latest technologies and trends in software engineering.
Participate in professional development opportunities like **certifications, workshops**, and **conferences**.

Example: A software engineer could attend **Agile methodology** workshops to stay current with project management trends.

1. Professional Responsibilities

1.3 Contributing to Team and Organizational Goals
Work collaboratively with colleagues, clients, and stakeholders to meet project goals.
Communicate effectively and handle feedback constructively.

Example: Participating in **code reviews** to ensure the team's work adheres to quality standards and learning from peers.

1.4 Ensuring Reliability and Safety
Engineers must ensure that the software works as expected under all conditions and **minimize risk** to users.
Consider **failures, recovery strategies**, and the **impact of errors**.
Example: Engineers should ensure that **data loss** in a system has a **backup strategy** in place to prevent serious consequences.

2. Ethical Responsibilities

2.1 Public Safety and Welfare
Software engineers must prioritize **public safety**, **privacy**, and the **welfare of users**.
They must **avoid harm** to individuals or society through unethical software development.

Example: A software engineer working on healthcare software must ensure that the application **protects patient data** and **complies with privacy laws**.

2.2 Avoiding Conflicts of Interest
Avoid situations where personal or financial interests conflict with professional duties.

Example: If a software engineer has financial ties to a company selling competing products, they must **disclose the conflict** and may need to **recuse themselves** from related decisions.

2. Ethical Responsibilities

2.3 Honesty and Integrity

Provide **accurate information** about software products and progress, both internally and to clients or customers.

Acknowledge mistakes and take steps to **correct them**.

Example: If a software feature does not work as intended, the engineer must **report the issue** honestly rather than **hide it**.

2.4 Respecting Intellectual Property

Respect **copyrights, patents, and trademarks** associated with software.

Do not **copy, reverse engineer, or distribute software** illegally.

Example: Software engineers should ensure they **license third-party libraries** properly and avoid using proprietary code without permission.

2. Ethical Responsibilities

2.5 Confidentiality and Data Protection
Ensure that sensitive information is **kept confidential** and not misused.
Example: A software engineer working for a financial institution must ensure that **user data** is kept **secure** and **protected** from unauthorized access.

2.6 Social Responsibility
Engineers should consider the **broader societal impact** of their software, such as its effect on **jobs, the economy**, and **the environment**.

They should avoid **developing software** that can contribute to **social harm**.

Example: A software engineer developing a social media platform should consider the **psychological impact** of the platform on users and take steps to **prevent harm**.

Summary of Professional & Ethical Responsibilities

Responsibility	Details
Delivering High-Quality Software	Ensure software reliability, performance, and security.
Commitment to Continuous Learning	Stay updated with current technologies and practices.
Team Collaboration	Work together to achieve organizational goals.
Public Safety & Welfare	Prioritize user safety, security, and privacy.
Avoid Conflicts of Interest	Ensure transparency and integrity in professional duties.
Honesty & Integrity	Be transparent about software progress and issues.
Respecting Intellectual Property	Follow legal and ethical guidelines related to software ownership.
Confidentiality & Data Protection	Protect sensitive information and user privacy.
Social Responsibility	Consider the societal impact of software products.

Attributes of Quality and good Software

Quality software is defined by its ability to meet both functional and non-functional requirements while being efficient, reliable, and maintainable. The key **attributes** of quality software focus on both **performance** and **user satisfaction**. Below are the primary attributes that define quality software:

1. Functionality

The **functional correctness** of a software system refers to its ability to perform the tasks it was designed to do. It includes:

- **Correctness**: The software meets the specified requirements and behaves as expected.
- **Suitability**: The software fulfills the needs of the users or business.
- **Accuracy**: The software performs computations and operations with precision and correctness.
- **Example**: A **banking application** must accurately calculate balances and process transactions without error.

2. Reliability

Reliability measures the software's ability to perform its required functions under specified conditions over time. It focuses on:

- **Stability**: The software should run without crashing or causing unexpected behavior.
- **Fault tolerance**: The software should handle errors gracefully without causing system failures.
- **Recoverability**: In case of failure, the software should be able to recover without losing critical data.

Example: A **web server** should maintain uptime, even during high traffic, and recover quickly in case of failure.

3. Usability

Usability reflects how easy and user-friendly the software is. It focuses on:
Ease of use: The software should be simple and intuitive for users to operate without requiring extensive training.
Learnability: New users should quickly learn to use the software with minimal effort.
User interface design: A clean, intuitive, and responsive design enhances user satisfaction.
Example: A **mobile app** should have an intuitive interface that allows users to navigate through features without confusion.

4. Efficiency

Efficiency refers to how well the software performs tasks with minimal resources. Key factors include:

Performance: The software should perform its tasks quickly and efficiently.

Resource utilization: The software should make efficient use of system resources, such as CPU, memory, and network bandwidth.

Example: A **video streaming service** should deliver high-quality videos without lag and use minimal bandwidth.

5. Maintainability

Maintainability is the ease with which software can be modified to fix defects, improve performance, or adapt to changes. This involves:
Modularity: The software should be broken into smaller, independent modules, making it easier to maintain and update.
Code clarity: Clear, readable code is easier to maintain and modify.
Testability: The software should be easy to test and debug.
Example: A **content management system (CMS)** should allow developers to easily add new features and fix bugs without major rework.

6. Portability

Portability refers to the software's ability to function on different platforms with minimal modification. It includes:
Adaptability: The software should be able to work on various devices or operating systems.
Installability: The software should be easy to install and configure across different environments.
Example: A **cross-platform mobile application** should work seamlessly on both **iOS** and **Android**

7. Security

Security is critical to protecting the software from threats and ensuring the safety of data. It involves:

- **Confidentiality**: Ensuring that sensitive data is protected from unauthorized access.
- **Integrity**: Ensuring that data remains accurate and unaltered by unauthorized users.
- **Authentication & Authorization**: Verifying user identities and granting them the appropriate access level.
- **Encryption**: Protecting data during transmission.

Example: A **payment gateway** must ensure user payment information is encrypted and cannot be tampered with.

8. Compatibility

Compatibility ensures that the software interacts well with other software and hardware. Key factors include:

- **Interoperability**: The ability of the software to work with other systems or software.
- **Coexistence**: The software should run alongside other software without causing conflicts or issues.

Example: A **document editor** should be able to open and save files in **multiple formats** (e.g., Word, PDF).

9. Scalability

Scalability measures how well the software can handle an increasing load or demand. It includes:

- **Handling increased users**: The software should perform well as the number of users or data increases.
- **Growth**: The software should be able to expand its capabilities without a complete redesign.

Example: A **cloud-based service** should be able to handle growing numbers of users by adding additional server resources.

10. Flexibility

Flexibility refers to the software's ability to adapt to new requirements or environments. It includes:

- **Configurability**: The software should allow users to configure settings according to their preferences.
- **Extensibility**: The software should allow new features to be added easily without disturbing existing functionality.

Example: An **enterprise resource planning (ERP)** system should allow businesses to customize modules to suit their specific needs.

Summary of Key Quality Attributes

Attribute	Description
Functionality	Correctness, suitability, and accuracy in fulfilling requirements.
Reliability	Stability, fault tolerance, and recoverability from failures.
Usability	Easy-to-use, intuitive, and user-friendly interfaces.
Efficiency	High performance with minimal resource consumption.
Maintainability	Easy to modify, update, and fix issues in the software.
Portability	Ability to operate on different platforms with minimal changes.
Security	Protection from unauthorized access, data integrity, and confidentiality.
Compatibility	Ability to work with other systems and devices without issues.
Scalability	Ability to handle growing user or data demands efficiently.
Flexibility	Adaptability to new requirements or environments.

Software Crises

The term **"Software Crisis"** refers to the challenges and problems faced by the software development industry due to the growing complexity, demand, and scale of software systems. It describes the mismatch between the **growing demand** for software and the ability of the industry to produce it efficiently, securely, and reliably.

Origins of the Software Crisis:
The software crisis began in the **1960s** when software became an essential component of computers and applications.
As **technology rapidly advanced**, the demand for more sophisticated and complex software increased, leading to a shortage of skilled engineers and an inability to meet deadlines and performance requirements.

The **software crisis** highlights the growing challenges faced by the industry due to increased software complexity, limited resources, and the need for better quality. However, adopting **modern methodologies, tools, and practices** can help address these issues and create a more efficient software development process.

Thank You

Any Questions?