Unit 1: Introduction (2Hrs.)

Computer works only in response to instructions provided externally. Computer interprets and executes these instructions and provides response to the user accordingly. A set of programs intended to provide users with a set of interrelated functionalities is known as software package. Software in broad sense consists of computer programs and associated documentation. Software enables the users to interact with a computer, its hardware, or perform tasks. For e.g. An accounting software package such as Tally provides users the functionality to perform accounting-related activities.

Software and its Types

Software can be broadly categorized into two main types based on how they are developed and who they are intended for:

Generic software

Generic software is developed for a broad market. It is intended to meet the needs of a wide range of users, not tailored to any one customer.

Characteristics:

- Designed, developed, and controlled by the development team.
- Requires extensive planning, designing, testing, and marketing.
- Must accommodate a wide variety of user needs and use cases.
- Example: Word processing software like Microsoft Word or spreadsheet software like Microsoft Excel.

Custom (or bespoke) software

Custom software is developed specifically for an individual client or organization, based on their particular requirements.

Characteristics:

- The client (customer) controls and defines the software requirements.
- There is no need for marketing as it is developed for a specific user or group.
- Usually built to automate or support specific business processes.
- Example: A control system for an industrial machine or a custom CRM system for a business.

Attributes of Good Software

Good software is not just about delivering the required functionality; it must also meet key quality attributes to ensure long-term usability, reliability, and efficiency. The main attributes of good software include:

Acceptability: Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

Dependability and security: Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Software has to be secure so that malicious users cannot access or damage the system.

Efficiency: Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, resource utilization, etc.

Maintainability: Software should be written in such a way that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.

Portability: Software should be able to run across different hardware and software environments with minimal modifications. Key aspects of software portability include cross-platform compatibility (Windows, macOS, Linux) and adaptability (e.g., different browsers, screen sizes).

Software Engineering and its Importance

Software engineering is an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance after it has been deployed to the target environment (server, cloud).

In the above definition, there are two key phrases:

1. Engineering discipline Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods.

Engineers also recognize that they must work within organizational and financial constraints, and they must look for solutions within these constraints.

2. All aspects of software production Software engineering are not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software development.

Importance of Software Engineering

Software engineering is crucial in modern technology-driven society for developing software that is trustworthy, cost efficient, scalable and adaptable. Some of the key importance of software engineering includes:

1. **Ensures Reliability and Correctness**
    - Builds **dependable** software that performs as expected under varying conditions.
    - Reduces risks of failures (e.g., crashes, data corruption) through systematic testing and validation.
2. **Improves Efficiency and Performance**
    - Optimizes resource usage (CPU, memory, network) for faster and scalable applications.
    - Helps avoid wasteful coding practices that lead to sluggish performance.
3. **Reduces Development and Maintenance Costs**
    - Structured methodologies prevent **technical debt**, lowering long-term expenses.
    - Proper documentation and modular design make future updates easier and cheaper.
4. **Enhances Security and Safety**
    - Protects against cyber threats (hacking, data breaches) through secure coding practices.
5. **Supports Scalability and Adaptability**
    - Enables software to grow with user demand (e.g., cloud applications, enterprise systems).
    - Facilitates **easy modifications** to meet evolving business or regulatory needs.
6. **Ensures User Satisfaction (Usability & Compatibility)**
    - Focuses on **user-centric design** for intuitive interfaces and smooth experiences.
    - Ensures compatibility across different platforms (Windows, macOS, mobile, web).

7. **Facilitates Team Collaboration**
    - Standardized processes (Agile, DevOps) improve coordination among developers, testers, and stakeholders.
    - Version control (e.g., Git) and documentation help teams work efficiently.
8. **Meets Legal and Industry Standards**
    - Complies with regulations to avoid legal penalties.
    - Follows best practices for auditability and accountability.
9. **Enables Faster Time-to-Market**
    - Agile and iterative approaches allow quicker releases of functional software.
    - Reduces delays caused by poor planning or unstructured coding.

Fundamental Software Engineering Activities

The systematic approach that is used in software engineering is sometimes called a software process. A software process is a sequence of activities that leads to the production of a software product. Four fundamental activities are common to all software processes. All software processes include these core stages:

1. **Software Specification (Requirements Engineering)**

    Goal: Define what the software should do and its constraints.

    Activities:

    - Gathering user needs.
    - Documenting functional & non-functional requirements.
    - Feasibility analysis (cost, time, technical constraints).
2. **Software Development (Design & Implementation)**

    Goal: Transform requirements into a working system.

    Activities:

    - **Design:** Architecture, data models, UI/UX.
    - **Implementation:** Writing code following best practices (modularity, readability).
3. **Software Validation (Testing & Quality Assurance)**

    Goal: Ensure the software meets requirements and is defect-free.

    Activities:

    - Unit testing, integration testing, user acceptance testing (UAT).

    - Performance & security testing.

4. **Software Evolution (Maintenance & Updates)**
   Goal: Adapt software to changing needs over time.
   Activities:
    - Bug fixes, performance optimization.
    - Adding new features based on user feedback.

Difference between Software Engineering and Computer Science

Software Engineering focuses on building software prioritizing process (Agile, testing). Computer Science focuses on understanding computation prioritizing innovation (new algorithms/models).

	Software Engineering (SE)	**Computer Science (CS)**
Primary Focus	Building reliable, scalable, and maintainable software systems.	Studying algorithms, computation, and theoretical foundations of computing.
Goal	Deliver high-quality software products efficiently.	Advance knowledge of computation and problem-solving.
Core Topics	- Software design & architecture - Development methodologies (Agile, DevOps) - Testing & QA - Project management	- Algorithms & data structures - Theory of computation - AI/ML - Cryptography - Computational math
Approach	Practical, application-oriented.	Theoretical, mathematical, and experimental.
Key Skills	- Coding best practices - System design	- Algorithm design - Mathematical modeling - Research & proof techniques

	Software Engineering (SE)	**Computer Science (CS)**
	- Team collaboration - Version control (Git)	
Output	Software products (apps, systems, tools).	Research papers, algorithms, new computational models.
Work Examples	- Developing a web/mobile app - Optimizing a cloud service - Debugging production code	- Designing a new sorting algorithm - Training an AI model

Difference between Software Engineering and System Engineering

Software engineering is the application of scientific principles to the design and creation of software. The field uses a systematic approach to collect and analyze business requirements in order to design, build, and test software applications to satisfy those business requirements. When computing began in the late 1950s, software engineering was a relatively undefined discipline, but over time it transformed into a modernized engineering field. The software engineering field became a discipline in the 1960s and evolved as new technologies were developed and the approach to software development became more scientific. Trends in software engineering transformed from ad hoc programming towards more formal and standardized methods.

System Engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software Engineering is part of this more general process. System Engineering focus on designing, integrating, and managing complex systems over their entire lifecycle. Unlike Software Engineering, which deals only with software, SE encompasses hardware, software, processes, and human factors to ensure all components work together effectively.

While Software Engineering focuses on coding and software development, System Engineering ensures that software, hardware, and processes function as a unified system.

Example:

- A Software Engineer develops the flight control software for a drone.
- A System Engineer ensures that the drone's software, sensors, motors, and communication systems work together seamlessly.

Challenges and Cost of Software Engineering

Software engineering is a complex and evolving field that faces numerous challenges, often leading to increased costs in development, maintenance, and scalability. Below are the key challenges and associated costs in software engineering.

1. **Heterogeneity**
   Modern software must operate across diverse environments, including:
   - Distributed systems (cloud, edge computing, IoT)
   - Multiple devices (computers, smartphones, wearables, embedded systems).
   - Legacy systems (old software written in outdated languages).

   The challenge is to ensure **dependability, interoperability, and flexibility** while managing this heterogeneity.

2. **Business & Social Change**
   Businesses and society evolve rapidly due to **new technologies (AI, blockchain)** and **global market shifts**. Traditional software engineering methods (e.g., Waterfall) are **too slow** for modern demands leading to delays in delivering value. The industry must adopt faster, adaptive approaches (e.g., Agile, DevOps, Low-Code) to accelerate development while maintaining quality.
3. **Security & Trust**
   With software deeply embedded in daily life (e.g., banking, healthcare, IoT), **security breaches and cyberattacks** pose major risks. Ensuring **trustworthy software** requires:
   - Secure coding practices (e.g., OWASP guidelines)
   - Encryption & authentication mechanisms
   - Regular security audits & penetration testing
   - Protection against malicious attacks (e.g., DDoS, phishing)
4. **Scale**
   Software must function efficiently across **vastly different scales**:
   - **Small-scale**: Embedded systems (wearables, IoT sensors) with limited resources.

- **Large-scale**: Cloud-based, globally distributed systems (e.g., social media, e-commerce) handling millions of users.

The challenge lies in designing **scalable architectures** (e.g., microservices, serverless) that maintain **performance, reliability, and cost-efficiency** at any scale.