Unit 1: Introduction to Software Engineering

What is Software?

"***Software is a set of programs (sequence of instructions) that allows the users to perform a well-defined function or some specified task.***"

Software is responsible for directing all computer-related devices and instructing them regarding what and how the task is to be performed. However, the software is made up of binary language (composed of ones and zeros), and for a programmer writing the binary code would be a slow and tedious task. Therefore, software programmers write the software program in various human-readable languages such as Java, Python, C#, etc. and later use the source code.

Types of Software

Software's are broadly classified into three types, i.e., **System Software and Application Software.**

1) System Software

The system software is the main software that runs the computer. When you turn on the computer, it activates the hardware and controls and coordinates their functioning. The application programs are also controlled by system software. An operating system is an example of system software.

2) Application Software:

Application software is a set of programs designed to perform a specific task. It does not control the working of a computer as it is designed for end-users. A computer can run without application software. Application software can be easily installed or uninstalled as required. It can be a single program or a collection of small programs. Microsoft Office Suite, Adobe Photoshop, and any other software like payroll software or income tax software are application software. As we know, they are designed to perform specific tasks.

Characteristics of Software:

1. **Functionality**: The software's ability to provide the intended features and functions to meet user requirements.

2. **Reliability**: The software's ability to perform consistently and reliably under varying conditions without failure.

3. **Usability**: How user-friendly and easy the software is to understand, learn, and use.

4. **Efficiency**: The software's ability to execute tasks with minimal resource consumption, such as CPU, memory, and time.

5. **Maintainability**: How easily the software can be modified, updated, and extended while retaining its integrity.

6. **Portability**: The ease with which software can be transferred from one environment to another, such as from one operating system to another.

7. **Security**: Measures taken to protect the software and its data from unauthorized access, attacks, and vulnerabilities.

8. **Scalability**: The software's ability to handle increased workloads or users without sacrificing performance.

9. I**nteroperability**: The software's ability to interact and work seamlessly with other systems or software.

10. **Flexibility**: The software's capacity to adapt to changing requirements and new functionalities.

11. **Robustness**: How well the software can handle unexpected inputs, errors, and adverse conditions without crashing or malfunctioning.

12. **Testability**: The ease with which the software can be tested to ensure its correctness and identify defects.

13. **Reusability**: The extent to which software components or modules can be reused in different contexts or projects.

14. **Modularity**: The software's design in separate, interchangeable components that can be developed and maintained independently.

15. Adaptability: The software's ability to be modified to suit new requirements or environments without a complete overhaul.

16. **Consistency**: The uniformity in design, behaviour, and documentation throughout the software.

17. **Cost Effectiveness**: The balance between the value the software provides and the resources required to develop, maintain, and operate it.

Programs v/s Software:

On the basis of	Program	Software
Definition	A computer program is a set of instructions that is used as a process of creating a software program by using programming language.	Software is a set of programs that enables the hardware to perform a specific task.
Types	Programs do not have further categorization.	The software can be of three types: system software, application software, and programming software.

User Interface	A program does not have a user interface.	Every software has a user interface that may be in graphical format or in the form of a command prompt.
Size	Programs are smaller in size, and their size exists between Kilobyte (Kb) to a megabyte (Mb).	Software's are larger in size, and their size exists between megabytes (Mb) to gigabytes (Gb).
Time taken	A program takes less time to be developed.	Whereas software requires more time to be developed.
Features and functionality	A program includes fewer features and limited functionalities.	It has more features and functionalities.
Development approach	The development approach of a program is unorganised, unplanned, and unprocedural.	The development approach of software is well planned, organised, and systematic.
Documentation	There is a lack of documentation in the program.	Softwares are properly documented.

Examples	Examples of the program are - video games, malware, and many more.	Examples of software are - Adobe Photoshop, Adobe Reader, Google Chrome, and many more.

Definition of Software Engineering:

Software Engineering is a systematic and disciplined approach to designing, developing, testing, deploying and maintaining software systems. It involves the application of engineering principles, methodology and best practices to create high quality software that meets users needs, is reliable, scalable and maintainable.

Importance of Software Engineering:

1. **Quality Assurance**: Software engineering provides systematic approaches to designing, developing, and testing software. This ensures that the resulting software products are of high quality, reliable, and meet user expectations.

2. **Efficiency and Productivity**: Proper software engineering practices improve the efficiency and productivity of development teams. Well-defined processes, methodologies, and tools streamline the development lifecycle and help deliver projects on time and within budget.

3. **User Satisfaction**: Software engineering methodologies involve gathering and analyzing user requirements, leading to software products that align with user needs and preferences. This enhances user satisfaction and adoption.

4. **Innovation and Technological Advancement**: Software engineering drives technological innovation by enabling the creation of new software applications and solutions that address emerging challenges and opportunities.

5. **Economic Growth**: The software industry contributes significantly to global economies. Well-engineered software products create job opportunities, drive business growth, and stimulate economic development.

6. **Global Accessibility**: Software engineering allows for the development of digital solutions that can be accessed globally, enabling businesses to reach wider markets and individuals to access resources from anywhere.

7. **Safety and Security**: Properly engineered software incorporates security measures to protect sensitive data and systems from cyber threats and attacks, ensuring user privacy and organizational integrity.

8. **Scalability and Flexibility**: Software engineering practices promote the development of scalable and flexible software architectures that can accommodate growing user bases and changing requirements.

9. **Interdisciplinary Collaboration**: Software engineering involves collaboration among professionals from various domains, such as design, development, testing, and project management. This interdisciplinary approach leads to well-rounded solutions.

10. **Regulatory Compliance**: In regulated industries such as healthcare and finance, software engineering ensures that software meets industry-specific regulations and standards.

11. **Risk Management**: Software engineering emphasizes risk assessment and management throughout the development process, reducing the likelihood of project failures and costly rework.

12. **Long-Term Sustainability**: Well-engineered software is easier to maintain and update over time, ensuring the longevity of software products and minimizing technical debt.

13. **Competitive Advantage**: Organizations that invest in software engineering practices gain a competitive edge by delivering higher-quality software products that meet user needs and offer advanced features.

14. **Adaptation to Change**: Software engineering methodologies, such as Agile, enable teams to adapt to changing requirements and market conditions, facilitating quick responses to evolving demands.

15. **Education and Skill Development**: The field of software engineering contributes to education and skill development by offering opportunities for individuals to learn and improve their technical expertise.

Principles of Software Engineering:

1. **Modularity**: Divide the software into smaller, manageable modules or components. This makes development, testing, and maintenance easier and allows for better organisation and reusability of code.

2. **Abstraction**: Hide complex implementation details behind well-defined interfaces. This allows developers to focus on the high-level functionality of modules without getting bogged down in the technical details.

3. **Encapsulation**: Enclose data and the methods that operate on that data within a single unit (class or module). This helps prevent unintended interference and ensures data integrity.

4. **Separation of Concerns**: Divide the software into distinct sections, each addressing a specific concern or responsibility. This improves code readability, maintainability, and allows different parts of the system to evolve independently.

5.**Modifiability**: Design software in a way that makes it easy to modify or extend. Changes in requirements or technology should not lead to extensive rework.

6. **Scalability**: Design software to handle increased workloads or growing user bases without a significant drop in performance. This includes considerations for distributed systems and load balancing.

7. **Simplicity**: Strive for simplicity in design and implementation. Avoid unnecessary complexity that can lead to confusion, errors, and difficulty in maintenance.

8. **Consistency**: Follow consistent coding standards, naming conventions, and design patterns throughout the project. This promotes readability and predictability across the codebase.

9. **Testing and Validation**: Incorporate testing early and throughout the development process. Testing helps identify defects and ensures that the software meets its requirements.

10. **Documentation**: Document code, design decisions, and processes thoroughly. Clear documentation aids in understanding, maintenance, and collaboration among team members.

11. **Iterative Development**: Use an iterative approach where software is developed in small increments or cycles. Each iteration adds new functionality or improvements, allowing for continuous feedback and adaptation.

12. **Version Control**: Use version control systems to track changes, collaborate with others, and manage different versions of the software codebase.

13. **Code Reusability**: Design code in a way that promotes reusability. This reduces redundant development efforts and improves efficiency.

14. **Software Development Lifecycle:** Follow established software development lifecycle models (e.g., Waterfall, Agile, DevOps) to manage the progression of a project from conception to deployment and maintenance.

15. **Quality Assurance**: Implement quality assurance processes to ensure that the software meets its requirements, adheres to standards, and is free from critical defects.

16. **User-Centred Design**: Design software with the end user in mind. Involve users in requirements gathering, usability testing, and feedback collection to create products that meet their needs.

17. **Continuous Improvement**: Embrace a culture of continuous improvement. Regularly assess development processes and outcomes, seeking ways to enhance efficiency, quality, and team collaboration.

Difference between Software Engineering and Software:

Aspect	Software Engineering	Software Programming
Focus	Entire software development lifecycle	Writing code
Scope	Broad, encompasses planning, design, testing, etc.	Narrow, mainly focused on coding
Goal	Develop high-quality, reliable software systems	Create functional code
Emphasis	Process, methodologies, and best practices	Implementation and coding

Activities	Requirements analysis, design, testing, etc.	Writing code, debugging, and testing
Design Considerations	Architecture, scalability, modularity, security	Functional logic, code structure, efficiency
Collaboration	Involves teamwork, project management, and more	May involve collaboration but can be solo
Creativity and Innovation	Involves designing solutions to complex problems	May require creative problem-solving
Maintenance	Considered throughout the software lifecycle	May be less emphasized or considered later
Time Involvement	Spans the entire development lifecycle	More focused on initial development
Documentation	Comprehensive documentation is emphasized	Documentation may be less detailed

Software Lifecycle	Adheres to software development methodologies	Typically part of the development process
Skill Set	Requires broader skills in design, analysis, etc.	Requires strong coding and logic skills

Members in Software Development:

Software development is a collaborative process that involves various roles and individuals with different expertise. Here are some of the key members typically involved in software development:

1. **Software Developers/Engineers**: These individuals write the code that makes up the software. They translate design specifications into actual working code and implement the software's functionality.

2. **Software Architects**: Architects are responsible for designing the overall structure and organization of the software. They make high-level decisions about the system's architecture, components, and technologies.

3. **System Analysts**: System analysts gather and analyze user requirements, turning them into detailed specifications that developers can work from. They bridge the gap between users and developers.

4. **Quality Assurance/Testers**: Testers are responsible for ensuring that the software meets quality standards and functions as intended. They design and execute test cases to identify and report bugs or issues.

5. **UI/UX Designers**: User Interface (UI) and User Experience (UX) designers create the visual elements and user interactions of the software. They focus on making the software aesthetically pleasing and user-friendly.

6. **Project Managers**: Project managers oversee the entire software development process. They plan, schedule, and coordinate tasks, allocate resources, and ensure that the project is completed on time and within budget.

7. **Product Owners/Product Managers**: Product owners or managers represent the stakeholders and are responsible for defining and prioritizing features, as well as making sure the software aligns with business goals.

8. **Database Administrators**: DBAs manage the databases used by the software. They design, maintain, and optimize the database structure, ensuring data integrity and security.

9. **DevOps Engineers**: DevOps engineers focus on automating and improving the development, deployment, and operations processes. They aim to achieve efficient and reliable software delivery.

10. **Security Experts**: Security experts address potential vulnerabilities and risks in the software. They implement measures to protect the software from threats and attacks.

11. **Documentation Specialists**: These individuals create user manuals, technical documentation, and other materials that help users understand and use the software effectively.

12. **Technical Writers**: Technical writers produce documentation that explains the software's features, functionality, and usage to a non-technical audience.

13. **Support and Maintenance Team**: This team provides ongoing support for the software, addressing user inquiries, fixing issues, and releasing updates.

14. **Stakeholders**: Stakeholders include anyone with an interest in the software's development, such as customers, end-users, investors, and regulatory bodies.