UNIT - 1

❖ Software Engineering

Software is a program or set of programs containing instructions that provide the desired functionality. Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

Software Engineering is the process of designing, developing, testing, and maintaining software. It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software.

1. Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.
2. It is a rapidly evolving field, and new tools and technologies are constantly being developed to improve the software development process.
3. By following the principles of software engineering and using the appropriate tools and methodologies, software developers can create high-quality, reliable, and maintainable software that meets the needs of its users.
4. Software Engineering is mainly used for large projects based on software systems rather than single programs or applications.
5. The main goal of Software Engineering is to develop software applications for improving quality, budget, and time efficiency.
6. Software Engineering ensures that the software that has to be built should be consistent, correct, also on budget, on time, and within the required requirements.

Objectives of Software Engineering

1. **Maintainability:** It should be feasible for the software to evolve to meet changing requirements.
2. **Efficiency:** The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
3. **Correctness:** A software product is correct if the different requirements specified in the SRS Document have been correctly implemented.
4. **Reusability:** A software product has good reusability if the different modules of the product can easily be reused to develop new products.

5. **Testability:** Here software facilitates both the establishment of test criteria and the evaluation of the software concerning those criteria.
6. **Reliability:** It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
7. **Portability:** In this case, the software can be transferred from one computer system or environment to another.
8. **Adaptability:** In this case, the software allows differing system constraints and the user needs to be satisfied by making changes to the software.
9. **Interoperability:** Capability of 2 or more functional units to process data cooperatively.

Advantages of Software Engineering

There are several advantages to using a systematic and disciplined approach to software development, such as:

1. **Improved Quality:** By following established software engineering principles and techniques, the software can be developed with fewer bugs and higher reliability.
2. **Increased Productivity:** Using modern tools and methodologies can streamline the development process, allowing developers to be more productive and complete projects faster.
3. **Better Maintainability:** Software that is designed and developed using sound software engineering practices is easier to maintain and update over time.
4. **Reduced Costs:** By identifying and addressing potential problems early in the development process, software engineering can help to reduce the cost of fixing bugs and adding new features later on.
5. I**ncreased Customer Satisfaction:** By involving customers in the development process and developing software that meets their needs, software engineering can help to increase customer satisfaction.
6. **Better Team Collaboration:** By using Agile methodologies and continuous integration, software engineering allows for better collaboration among development teams.
7. **Better Scalability**: By designing software with scalability in mind, software engineering can help to ensure that software can handle an increasing number of users and transactions.

8. **Better Security:** By following the Software Development Life Cycle (SDLC) and performing security testing, software engineering can help to prevent security breaches and protect sensitive data.

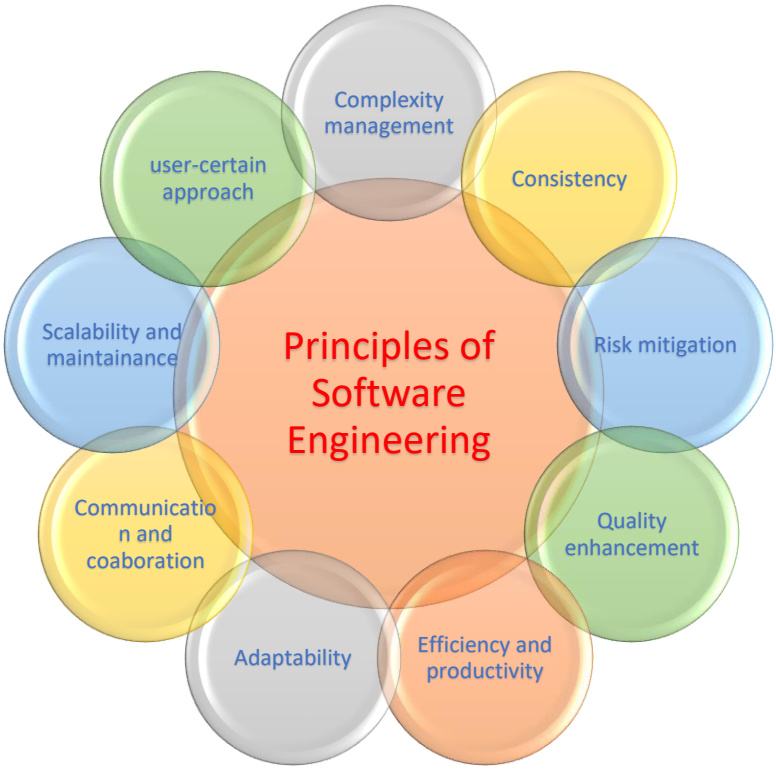
Disadvantages of Software Engineering

While Software Engineering offers many advantages, there are also some potential disadvantages to consider:

1. **High upfront costs:** Implementing a systematic and disciplined approach to software development can be resource-intensive and require a significant investment in tools and training.
2. **Limited flexibility:** Following established software engineering principles and methodologies can be rigid and may limit the ability to quickly adapt to changing requirements.
3. **Bureaucratic**: Software Engineering can create an environment that is bureaucratic, with a lot of processes and paperwork, which may slow down the development process.
4. **Complexity**: With the increase in the number of tools and methodologies, software engineering can be complex and difficult to navigate.
5. **Limited creativity:** The focus on structure and process can stifle creativity and innovation among developers.
6. **High learning curve:** The development process can be complex, and it requires a lot of learning and training, which can be challenging for new developers.
7. **High dependence on tools:** Software engineering heavily depends on the tools, and if the tools are not properly configured or are not compatible with the software, it can cause issues.
8. **High maintenance**: The software engineering process requires regular maintenance to ensure that the software is running efficiently, which can be costly and time-consuming.

❖ Principles in software development

Principles in software development serve as guiding rules and fundamental concepts that help streamline the process, enhance the quality of the software, and improve the overall efficiency of development projects. These principles are not just theoretical concepts; they provide practical strategies to tackle the complexities and challenges that arise during the software development lifecycle. Here's why there is a requirement for principles in software development:



1. **Complexity management:** Software development involves intricate designs, interactions, and functionalities. Principles offer a structured approach to managing this complexity, breaking down the process into manageable components and stages.
2. **Consistency:** Principles provide a consistent framework for software development. They help ensure that all team members adhere to a common set of guidelines, leading to uniformity in code quality, design patterns, and project execution.

3. **Risk mitigation:** Developing software is fraught with uncertainties and risks. Principles such as iterative development and change management help identify and mitigate risks early in the process, reducing the chances of costly errors later on.

4. **Quality enhancement:** Principles like objective quality control and modular design contribute to the improvement of software quality. By implementing these principles, developers can identify and rectify defects, leading to a more reliable and stable end product.

5. **Efficiency and productivity:** Principles promote efficiency by offering proven methodologies and best practices. For instance, the component-based approach

encourages code reuse, saving time and effort in development. This ultimately boosts productivity across the development team.

6. Adaptability: The software industry is dynamic, with evolving user requirements and technological advancements. Principles such as evolving levels of details and model-based evolution allow for flexible adaptation to changes, ensuring that the software remains relevant over time.

7. Communication and collaboration: Principles promote effective communication within development teams and with stakeholders. Clear guidelines and shared understanding enable smoother collaboration, leading to better decision-making and problem-solving.

8. Scalability and maintainability: Principles like architecture-first approach and modularity lay the foundation for scalable and maintainable software. Designing a solid architecture and breaking down software into modules make it easier to extend, modify, and enhance the system as needed.

9. Cost efficiency: Applying principles can reduce development costs in the long run. By catching errors early, avoiding rework, and promoting efficient practices, software development becomes more cost-effective.

10. User-centric approach: Principles help developers align their efforts with user needs and expectations. By following principles like demonstration-based approaches, developers can ensure that the software addresses real-world problems and provides meaningful solutions.

Principles in software development provide a roadmap for creating high-quality software that meets user needs, adapts to changes, and stands the test of time

❖ Careers Are There in Software Engineering

A degree in software engineering and relevant experience can be utilized to explore several computing job choices. Software engineers have the opportunity to seek well-paying careers and professional progress, although their exact possibilities may vary depending on their particular school, industry, and region.

Following are the job choices in software engineering:

- SWE (Software Engineer)
- SDE (Software Development Engineer)
- Web Developer
- Quality Assurance Engineer

- Web Designer
- Software Test Engineer
- Cloud Engineer ·
- Front-End Developer
- Back-End Developer
- DevOps Engineer.
- Security Engineer.

Skills do Software Engineers Need

To achieve success, software engineers require a unique set of technical and soft skills. These skills include computer programming knowledge for designing, developing, testing, and debugging software, as well as soft skills for interacting with stakeholders, team members, and company leadership.

Following are some must have technical skills to become Software Engineers:

- Coding and computer programming
- Software testing
- Object-oriented design (OOD)
- Software development

Following are some must have soft skills to become Software Engineers:

- Communication skills
- Team player
- Problem solving
- Attention to detail

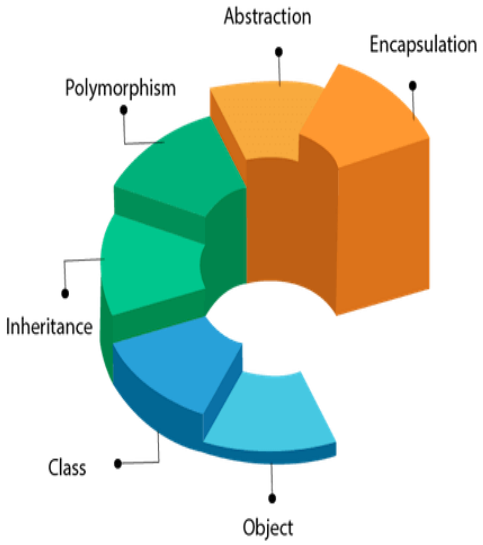
❖ OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction

- Encapsulation
- Association
- Aggregation
- Composition

OOPs (Object-Oriented Programming System)



Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

- One to One
- One to Many
- Many to One, and
- Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be undirectional or bidirectional.

Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java. Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

❖ Unified Modeling Language (UML) diagrams

A picture is worth a thousand words. That's why Unified Modeling Language (UML) diagramming was created: to forge a common visual language in the complex world of software development that would also be understandable for business users and anyone who wants to understand a system. Learn the essentials of UML diagrams along with their origins, uses, concepts, types and guidelines on how to draw them using our UML diagram tool

UML

The Unified Modeling Language (UML) was created to forge a common, semantically and syntactically rich visual modeling language for the architecture, design, and implementation of complex software systems both structurally and behaviorally. UML has applications beyond software development, such as process flow in manufacturing.

It is analogous to the blueprints used in other fields, and consists of different types of diagrams. In the aggregate, UML diagrams describe the boundary, structure, and the behavior of the system and the objects within it.

UML is not a programming language but there are tools that can be used to generate code in various languages using UML diagrams. UML has a direct relation with object-oriented analysis and design.

UML and its role in object-oriented modeling and design

There are many problem-solving paradigms or models in Computer Science, which is the study of algorithms and data. There are four problem-solving model categories: imperative, functional, declarative and object-oriented languages (OOP). In object-oriented languages, algorithms are expressed by defining 'objects' and having the

objects interact with each other. Those objects are things to be manipulated and they exist in the real world. They can be buildings, widgets on a desktop, or human beings.

Object-oriented languages dominate the programming world because they model real-world objects. UML is a combination of several object-oriented notations: Object-Oriented Design, Object Modeling Technique, and Object-Oriented Software Engineering.

different types of UML diagrams, try one or all of these tutorials to guide you through the process of drawing both structural and behavioral diagrams.

- Structural Diagram Examples

CLASS DIAGRAMS

Class diagrams represent the static structures of a system, including its classes, attributes, operations, and objects. A class diagram can display computational data or organizational data in the form of implementation classes and logical classes, respectively.

COMPONENT DIAGRAMS

Component diagrams show how components are combined to form larger components or software systems. These diagrams are meant to model the dependencies of each component in the system. A component is something required to execute a stereotype function. A component stereotype may consist of executables, documents, database tables, files, or library files.

DEPLOYMENT DIAGRAMS

A deployment diagram models the physical deployment and structure of hardware components. Deployment diagrams demonstrate where and how the components of a system will operate in conjunction with each other.

- Behavioral Diagram Examples

ACTIVITY DIAGRAM

Activity diagrams show the procedural flow of control between class objects, along with organizational processes like business workflows. These diagram are made of specialized shapes, then connected with arrows. The notation set for activity diagrams is similar to those for state diagrams.

USE CASE DIAGRAM

A use case is a list of steps that define interaction between an actor (a human who interacts with the system or an external system) and the system itself. Use case

diagrams depict the specifications of a use case and model the functional units of a system. These diagrams help development teams understand the requirements of their system, including the role of human interaction therein and the differences between various use cases.

SEQUENCE DIAGRAM

Sequence diagrams, also known as event diagrams or event scenarios, illustrate how processes interact with each other by showing calls between different objects in a sequence. These diagrams have two dimensions: vertical and horizontal. The vertical lines show the sequence of messages and calls in chronological order, and the horizontal elements show object instances where the messages are relayed.

- ### Advantages of Unified Modelling Language (UML)

1) Visualisation

UML diagrams showcase system elements, connections, and operations. This visual picture helps in understanding and creating intricate systems.

2) Analysis and Design

UML aids in both the analysis and design phases of software development. It helps represent system needs and turn them into a design that can be implemented.

3) Communication

UML diagrams effectively communicate complex concepts to various stakeholders, such as Developers, Designers, Testers, and Business Users.

4) Standardisation

UML offers a standardised way of depicting system models. This guarantees that developers and stakeholders can effectively communicate with each other using a shared visual representation.

5) Clarity and Adaptability

UML offers standard diagrams that are clear and easy to understand. These diagrams can be modified to meet different project requirements, improving communication and comprehension.

6) Simplified Debugging Process

UML diagrams simplify the process of identifying and resolving issues. They visually illustrate system parts and relationships, making troubleshooting and issue resolution easier.

- **Disadvantages of Unified Modelling Language (UML)**

1) Time Consuming

Developing and managing large UML diagrams can be time-consuming, particularly on extensive projects. This may delay the development schedule and raise the project budget.

2) Designing

Creating diagrams with UML can be complex, necessitating scrutiny to guarantee accurate and efficient designs.

3) Ambiguity

Understanding UML diagrams can vary from person to person, possibly leading to confusion within the team.

4) Complexity

UML's wide range of capabilities may seem daunting, especially for newbies, causing difficulty in becoming proficient.

5) Overhead

Developing and keeping up with UML diagrams can require extra effort, especially for smaller projects that don't require extensive documentation.

6) Learning Curve

Learning UML can be challenging because of its many features and types of diagrams, requiring practical training and practice.

7) Over-modelling or under-modelling

Over-modelling or under-modelling refers to the problem of creating machine learning (ML) models that are too complex or simplistic, leading to poor performance on unseen data.

❖ Software Development Life Cycle (SDLC)

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the

necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

Need of SDLC

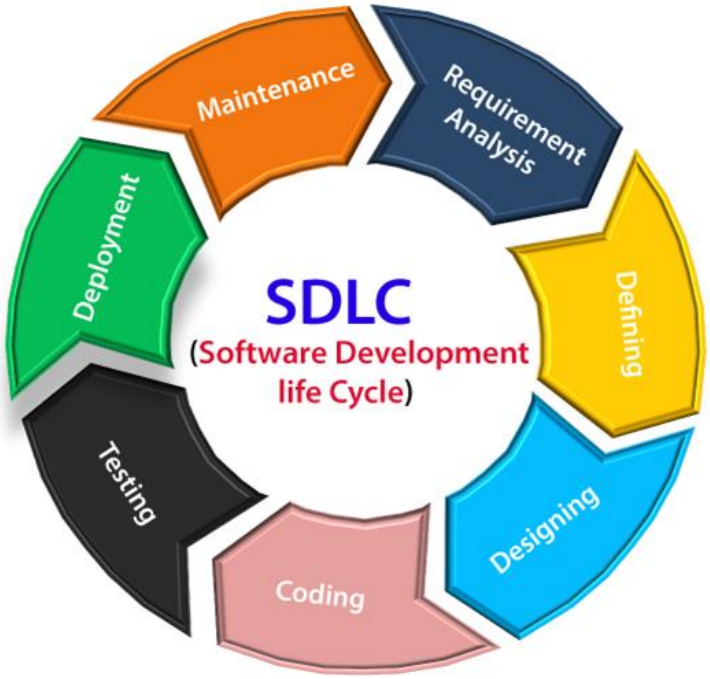
The development team must determine a suitable life cycle model for a particular plan and then observe to it.

Without using an exact life cycle model, the development of a software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team representative about when and what to do. Otherwise, it would point to chaos and project failure. This problem can be defined by using an example. Suppose a software development issue is divided into various parts and the parts are assigned to the team members. From then on, suppose the team representative is allowed the freedom to develop the roles assigned to them in whatever way they like. It is possible that one representative might start writing the code for his part, another might choose to prepare the test documents first, and some other engineer might begin with the design phase of the roles assigned to him. This would be one of the perfect methods for project failure.

A software life cycle model describes entry and exit criteria for each phase. A phase can begin only if its stage-entry criteria have been fulfilled. So without a software life cycle model, the entry and exit criteria for a stage cannot be recognized. Without software life cycle models, it becomes tough for software project managers to monitor the progress of the project.

SDLC Cycle

SDLC Cycle represents the process of developing software. SDLC framework includes the following steps:



The stages of SDLC are as follows:

Stage1: Planning and requirement analysis

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

Stage2: Defining Requirements

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

Stage3: Designing the Software

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

Stage4: Developing the project

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

Stage5: Testing

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.

During this stage, unit testing, integration testing, system testing, acceptance testing are done.

Stage6: Deployment

Once the software is certified, and no bugs or errors are stated, then it is deployed.

Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

Stage7: Maintenance

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

This procedure where the care is taken for the developed product is known as maintenance.