Introduction: Introduction to Software & Software Engineering, Goals and principles, Myths, Importance of Software Engineering, Software Development Life Cycle, Software Process and its standard, Work product.

Software :

Software is a collection of programs and data that tell a computer how to perform specific tasks. Software often includes associated software documentation. This is in contrast to hardware, from which the system is built and which actually performs the work

Software engineering :

The term **software engineering** is the product of two words, **software**, and **engineering**.

The **software** is a collection of integrated programs.

Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.

Computer programs and related documentation such as requirements, design models and user manuals.

Engineering is the application of **scientific** and **practical** knowledge to **invent, design, build, maintain**, and **improve frameworks, processes, etc**.

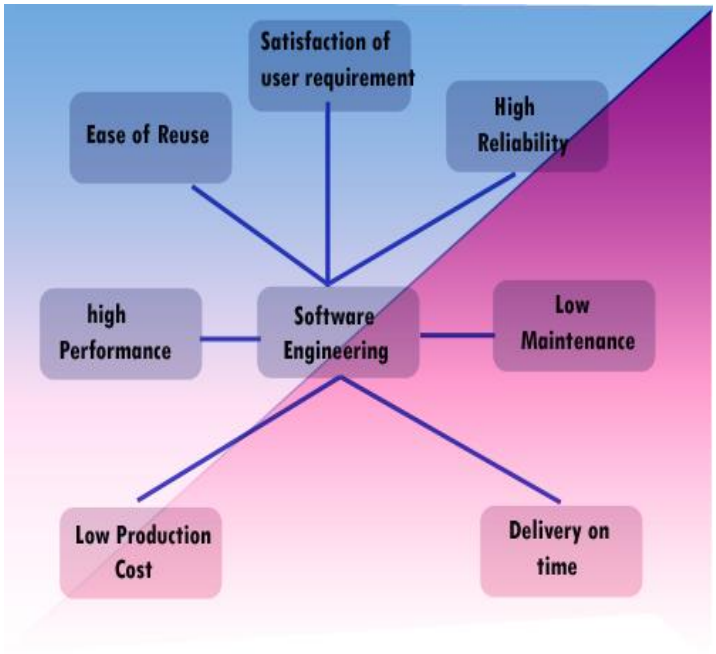
Goals of software:

User Satisfaction:

This is the first out of all goals of software engineering and is also the most important goal as all the stuff is for a customer or user so we should be very focused on user satisfaction while developing any software.

A few programmers do this as they start developing the software immediately without understanding the actual requirement of the end-user and this results in the improper flow of software which the user did not want in actuality.

So by doing this programmer loses his energy and user faith or user satisfaction and if the programmer rebuilds it again then it is an overhead to him to build it again.



High reliability:

this is the second out of all the goals of software engineering. this one tells us that we do not have any scope to have any mistakes or bugs in our final product which is going to release at the user end.

If it has mistakes and bugs then it can affect our relationship with our customers and thus this can highly impact the selling of our software in the market and can create a high loss percentage.

As Microsoft has also some bugs in the earlier release of Windows and users were facing lots of problems. So the software is to be released only if high reliability has been achieved and there is no chance of any bugs.

Low Maintenance Cost:

This is the third one out of all the goals of software engineering. Maintenance is a process in which a small problem or bugs that have been detected while using the software at the user end are addressed and fixed easily. But it does not mean restructuring software from scratch or starting.

I mean to say if there is any problem with the software then you have to design it once again. This happens if the software has very poor quality and is made without any testing and parameters.

Delivery on time:

This is the fourth goal of software engineering. The delivery time matters while you develop software for your client or customer.

This is not possible to tell the exact time to complete the software but if the developing work is to be done in a systematic order by breaking the whole project into parts and estimating the time for each module.

By doing this analysis a proposed deadline can be given for completing a project for a client.

Low Production Cost:

As per the low production goals of software engineering software that is cost-effective gets always the attention of users.

If the software succeeds in matching the user requirement there is a big chance of sales or profit in either way.

High Performance:

Software performance is generally measured by its speed and memory consumption so we should develop it in such a way that it can be run in minimal memory space with high speed.

This optimization of the software will make it useful for users and will have a high demand in the market.

Ease of reuse:

If you are building a small unit of big software then it is very necessary to then we should try to make it in such a way that it can be reused if it is needed in building the same software or in other software too.

Principles Of software

1. **Modularity**: Breaking down the software into smaller, independent, and reusable components or modules. This makes the software easier to understand, test, and maintain.
2. **Abstraction**: Hiding the implementation details of a module or component and exposing only the necessary information. This makes the software more flexible and easier to change.
3. **Encapsulation**: Wrapping the data and functions of a module or component into a single unit, and providing controlled access to that unit. This helps to protect the data and functions from unauthorized access and modification.
4. **DRY principle (Don’t Repeat Yourself):** Avoiding duplication of code and data in the software. This makes the software more maintainable and less error-prone.
5. **KISS principle (Keep It Simple, Stupid)**: Keeping the software design and implementation as simple as possible. This makes the software more understandable, testable, and maintainable.

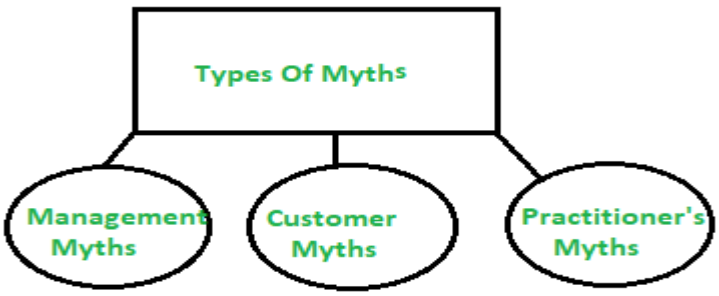
6. **YAGNI (You Ain't Gonna Need It):** Avoiding adding unnecessary features or functionality to the software. This helps to keep the software focused on the essential requirements and makes it more maintainable.

7. **SOLID principles:** A set of principles that guide the design of software to make it more maintainable, reusable, and extensible. This includes the Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, and Dependency Inversion Principle.

8. **Test-driven development:** Writing automated tests before writing the code, and ensuring that the code passes all tests before it is considered complete. This helps to ensure that the software meets the requirements and specifications.

Software Myths:

Most, experienced experts have seen myths or superstitions (false beliefs or interpretations) or misleading attitudes (naked users) which creates major problems for management and technical people. The types of software-related myths are listed below.



`Types of Software Myths

(i) Management Myths:

Myth 1:

We have all the standards and procedures available for software development.

Fact:

- Software experts do not know all the requirements for the software development.
- And all existing processes are incomplete as new software development is based on new and different problem.

Myth 2:

The addition of the latest hardware programs will improve the software development.

Fact:

- The role of the latest hardware is not very high on standard software development; instead (CASE) Engineering tools help the computer, they are more important than hardware to produce quality and productivity.
- Hence, the hardware resources are misused.

Myth 3:

- With the addition of more people and program planners to Software development can help meet project deadlines (If lagging behind).

Fact:

- If software is late, adding more people will merely make the problem worse. This is because the people already working on the project now need to spend time educating the newcomers, and are thus taken away from their work. The newcomers are also far less productive than the existing software engineers, and so the work put into training them to work on the software does not immediately meet with an appropriate reduction in work.

(ii)Customer Myths:

The customer can be the direct users of the software, the technical team, marketing / sales department, or other company. Customer has myths leading to false expectations (customer) & that's why you create dissatisfaction with the developer.

Myth 1:
A general statement of intent is enough to start writing plans (software development) and details of objectives can be done over time.

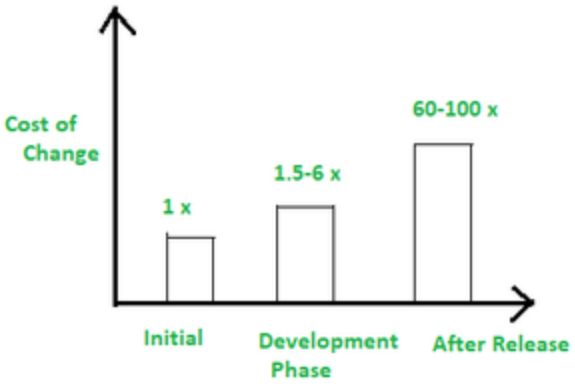
Fact:

- Official and detailed description of the database function, ethical performance, communication, structural issues and the verification process are important.
- Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous
  communication between customer and developer.

Myth 2:
Software requirements continually change, but change can be easily accommodated because software is flexible

Fact:

- It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small. However, as time passes, the cost impact grows rapidly—resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modification.



Cost of Change

Different Stages of Myths

(iii)Practitioner's Myths:
Myths 1:
They believe that their work has been completed with the writing of the plan.

Fact:

- It is true that every 60-80% effort goes into the maintenance phase (as of the latter software release). Efforts are required, where the product is available first delivered to customers.

Myths 2:
There is no other way to achieve system quality, until it is "running".

Fact:

- Systematic review of project technology is the quality of effective software verification method. These updates are quality filters and more accessible than test.

Myth 3:
An operating system is the only product that can be successfully exported project.

Fact:

- A working system is not enough, the right document brochures and booklets are also required to provide guidance & software support.

Myth 4:
Engineering software will enable us to build powerful and unnecessary document & always delay us.

Fact:

- Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times

Importance of Software Engineering

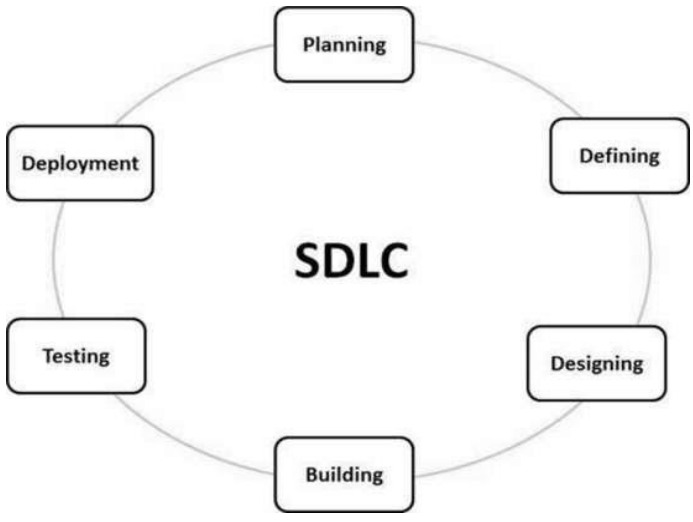
One of the main goals of software engineering is to produce top-notch software in a more efficient way. It involves the application of specific principles, processes, and practices to manage and revamp the development of a software system. In other words, organizations can leverage software engineering to:

1. Design and develop software products.
2. Develop and maintain databases.
3. Assess and improve security measures.
4. Develop and maintain networks.
5. Research, evaluate, and implement new technologies.
6. Develop user documentation and training materials.
7. Analyze and resolve software-related issues.
8. Conduct software testing and quality assurance.
9. Identify, document, and track software defects.
10. Maximize system performance and reliability.

Software Development Life Cycle

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.



Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

Software Process and its standard

Software processes in software engineering refer to the methods and techniques used to develop and maintain software. Some examples of software processes include:

- **Waterfall**: a linear, sequential approach to software development, with distinct phases such as requirements gathering, design, implementation, testing, and maintenance.
- **Agile**: a flexible, iterative approach to software development, with an emphasis on rapid prototyping and continuous delivery.
- **Scrum**: a popular Agile methodology that emphasizes teamwork, iterative development, and a flexible, adaptive approach to planning and management.
- **DevOps**: a set of practices that aims to improve collaboration and communication between development and operations teams, with an emphasis on automating the software delivery process.

Each process has its own set of advantages and disadvantages, and the choice of which one to use depends on the specific project and organization.

Software is the set of instructions in the form of programs to govern the computer system and to process the hardware components. To produce a software product the set of activities is used. This set is called a software process.

Software Development : In this process, designing, programming, documenting, testing, and bug fixing is done.

Components of Software :

There are three components of the software: These are : Program, Documentation, and Operating Procedures.

1. **Program –**
   A computer program is a list of instructions that tell a computer what to do.
2. **Documentation –**
   Source information about the product contained in design documents, detailed code comments, etc.
3. **Operating Procedures –**
   Set of step-by-step instructions compiled by an organization to help workers carry out complex routine operations.
4. **Code**: the instructions that a computer executes in order to perform a specific task or set of tasks.
5. **Data**: the information that the software uses or manipulates.
6. **User** interface: the means by which the user interacts with the software, such as buttons, menus, and text fields.
7. **Libraries**: pre-written code that can be reused by the software to perform common tasks.
8. **Documentation**: information that explains how to use and maintain the software, such as user manuals and technical guides.
9. **Test cases**: a set of inputs, execution conditions, and expected outputs that are used to test the software for correctness and reliability.

10. **Configuration files**: files that contain settings and parameters that are used to configure the software to run in a specific environment.
11. **Build and deployment** scripts: scripts or tools that are used to build, package, and deploy the software to different environments.
12. **Metadata**: information about the software, such as version numbers, authors, and copyright information.

All these components are important for software development, testing and deployment.

There are four basic key process activities:

1. **Software Specifications –**
   In this process, detailed description of a software system to be developed with its functional and non-functional requirements.
2. **Software Development –**
   In this process, designing, programming, documenting, testing, and bug fixing is done.
3. **Software Validation –**
   In this process, evaluation software product is done to ensure that the software meets the business requirements as well as the end users needs.
4. **Software Evolution –**
   It is a process of developing software initially, then timely updating it for various reasons.

Software Crisis :

1. **Size and Cost –**
   Day to day growing complexity and expectation out of software. Software are more expensive and more complex.
2. **Quality –**
   Software products must have good quality.
3. **Delayed Delivery –**
   Software takes longer than the estimated time to develop, which in turn leads to cost shooting up.
4. The term “software crisis” refers to a set of problems that were faced by the software industry in the 1960s and 1970s, such as:
5. High costs and long development times: software projects were taking much longer and costing much more than expected.
6. **Low quality**: software was often delivered late, with bugs and other defects that made it difficult to use.
7. **Lack of standardization**: there were no established best practices or standards for software development, making it difficult to compare and improve different approaches.
8. **Lack of tools and methodologies**: there were few tools and methodologies available to help with software development, making it a difficult and time-consuming process.
9. These problems led to a growing realization that the traditional approaches to software development were not effective and needed to be improved. This led to the development of new software development methodologies, such as the Waterfall and Agile methodologies, as well as the creation of new tools and technologies to support software development.

However, even today, software crisis could be seen in some form or the other, like for example software projects going over budget, schedule and not meeting the requirement.

Software Process Model:
A software process model is an abstraction of the actual process, which is being described. It can also be defined as a simplified representation of a software process. Each model represents a process from a specific perspective. Basic software process models on which different type of software process models can be implemented:

1. **A workflow Model –**
   It is the sequential series of tasks and decisions that make up a business process.
2. **The Waterfall Model –**
   It is a sequential design process in which progress is seen as flowing steadily downwards. Phases in waterfall model:

   (i) Requirements Specification

   (ii) Software Design

   (iii) Implementation

   (iv) Testing
3. **Dataflow Model –**
   It is diagrammatic representation of the flow and exchange of information within a system.
4. **Evolutionary Development Model –**
   Following activities are considered in this method:

   (i) Specification

   (ii) Development

   (iii) Validation
5. **Role / Action Model –**
   Roles of the people involved in the software process and the activities.

Need for Process Model:

The software development team must decide the process model that is to be used for software product development and then the entire team must adhere to it. This is necessary because the software product development can then be done systematically. Each team member will understand what is the next activity and how to do it. Thus process model will bring the definiteness and discipline in overall development process. Every process model consists of definite entry and exit criteria for each phase. Hence the transition of the product through various phases is definite.

If the process model is not followed for software development then any team member can perform any software development activity, this will ultimately cause a chaos and software project will definitely fail without using process model, it is difficult to monitor the progress of software product. Thus process model plays an important rule in software engineering.

Advantages or Disadvantages:

There are several advantages and disadvantages to different software development methodologies, such as:

Waterfall:
Advantages:

1. Clear and defined phases of development make it easy to plan and manage the project.
2. It is well-suited for projects with well-defined and unchanging requirements.

Disadvantages:

- Changes made to the requirements during the development phase can be costly and time-consuming.
- It can be difficult to know how long each phase will take, making it difficult to estimate the overall time and cost of the project.
- It does not have much room for iteration and feedback throughout the development process.

Agile:
Advantages:

1. Flexible and adaptable to changing requirements.
2. Emphasizes rapid prototyping and continuous delivery, which can help to identify and fix problems early on.
3. Encourages collaboration and communication between development teams and stakeholders.

Disadvantages:

1. It may be difficult to plan and manage a project using Agile methodologies, as requirements and deliverables are not always well-defined in advance.
2. It can be difficult to estimate the overall time and cost of a project, as the process is iterative and changes are made throughout the development.

Scrum:
Advantages:

1. Encourages teamwork and collaboration.
2. Provides a flexible and adaptive framework for planning and managing software development projects.
3. Helps to identify and fix problems early on by using frequent testing and inspection.

Disadvantages:

1. A lack of understanding of Scrum methodologies can lead to confusion and inefficiency.
2. It can be difficult to estimate the overall time and cost of a project, as the process is iterative and changes are made throughout the development.

DevOps:
Advantages:

1. Improves collaboration and communication between development and operations teams.
2. Automates software delivery process, making it faster and more efficient.
3. Enables faster recovery and response time in case of issues.

Disadvantages:

1. Requires a significant investment in tools and technologies.
2. Can be difficult to implement in organizations with existing silos and lack of culture of collaboration.
3. Need to have a skilled workforce to effectively implement the devops practices.
4. Ultimately, the choice of which methodology to use depends on the specific project and organization, as well as the goals and requirements of the project

Work product

Work Product means the tangible and intangible results of the Work, whether finished or unfinished, including drafts. Work Product includes, but is not limited to, documents, text, software (including source code), research, reports, proposals, specifications, plans, notes, studies, data, images, photographs, negatives, pictures, drawings, designs, models, surveys, maps, materials, ideas, concepts, know-how, and any other results of the Work. “Work Product” does not include any material that was developed prior to the Effective Date that is used, without modification, in the performance of the Work. Any other term used in this Contract that is defined in an Exhibit shall be construed and interpreted as defined in that Exhibit