Software Engineering

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Definitions

IEEE defines software engineering as:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.
(2) The study of approaches as in the above statement.

Fritz Bauer, a German computer scientist, defines software engineering as:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as **software evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.

Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Software Evolution Laws

Lehman has given laws for software evolution. He divided the software into three different categories:

S-type (static-type) - This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.

P-type (practical-type) - This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.

E-type (embedded-type) - This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

Software Paradigms

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another.

Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes –

- Coding
- Testing
- Integration

Components of Software :

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

Program –
A computer program is a list of instructions that tell a computer what to do.

Documentation –
Source information about the product contained in design documents, detailed code comments, etc.

Operating Procedures –
Set of step-by-step instructions compiled by an organization to help workers carry out complex routine operations.

There are four basic key process activities:

Software Specifications –
In this process, detailed description of a software system to be developed with its functional and non-functional requirements.

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

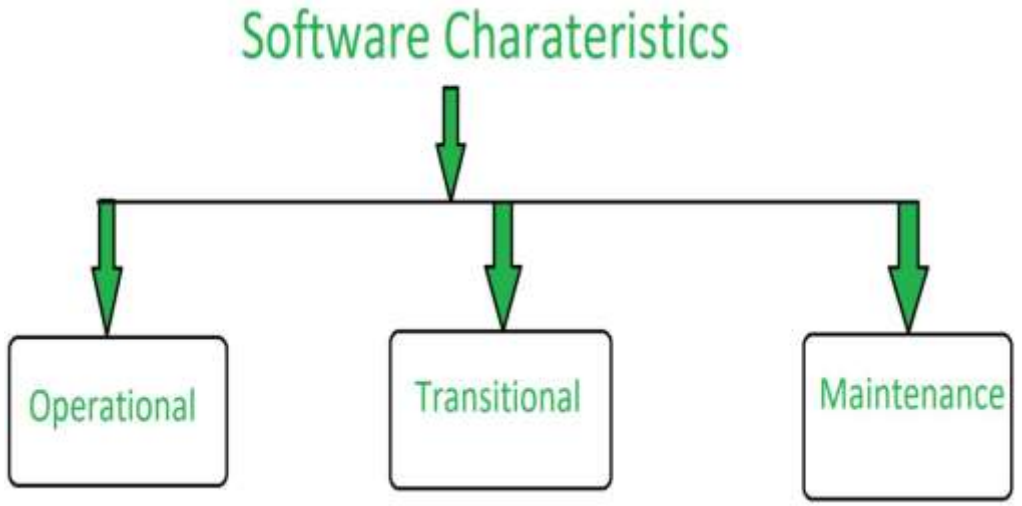
Software Validation –
In this process, evaluation software product is done to ensure that the software meets the business requirements as well as the end users needs.

Software Evolution –
It is a process of developing software initially, then timely updating it for various reasons.

Software is treated as a good software by the means of different factors. A software product is concluded as a good software by what it offers and how well it can be used. The factors that decide the software properties are divided into three categories:

Operational, Transitional, and Maintenance. These are explained as following below.

1. **Operational:**
   In operational categories, the factors that decide the software performance in operations. It can be measured on:



- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

2. Transitional:
When the software is moved from one platform to another, the factors deciding the software quality:

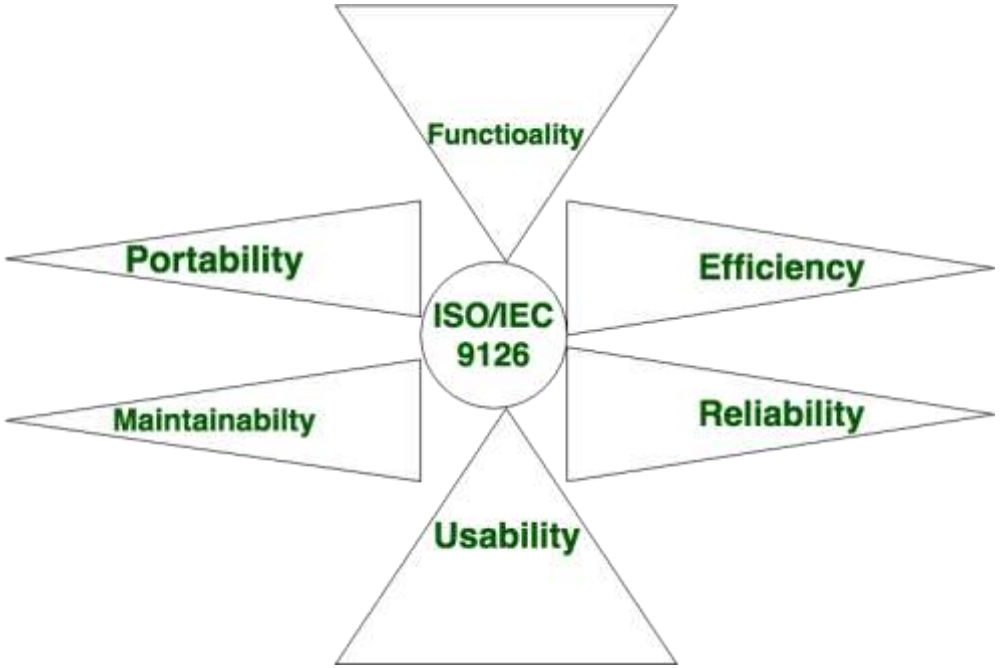
- Portability
- Interoperability
- Reusability
- Adaptability

3. Maintenance:
In this categories all factors are included that describes about how well a software has the capabilities to maintain itself in the ever changing environment:

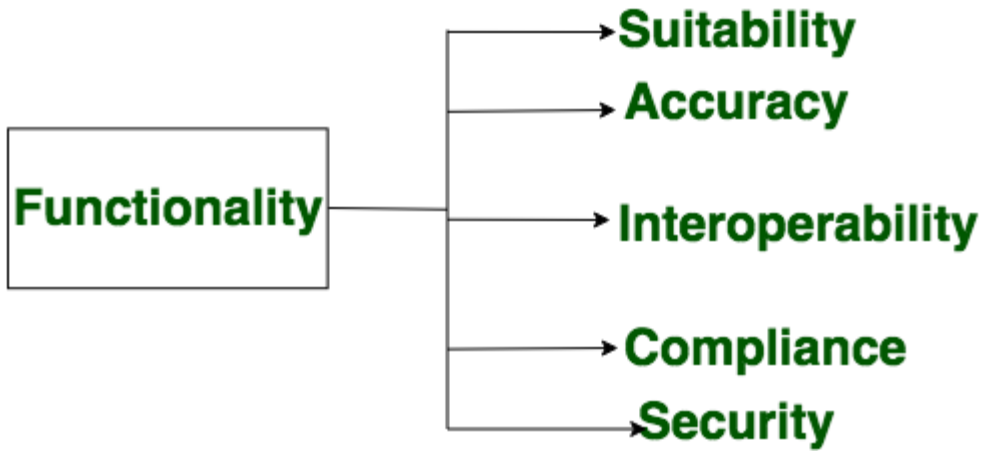
- Modularity
- Maintainability
- Flexibility
- Scalability

Software Characteristics

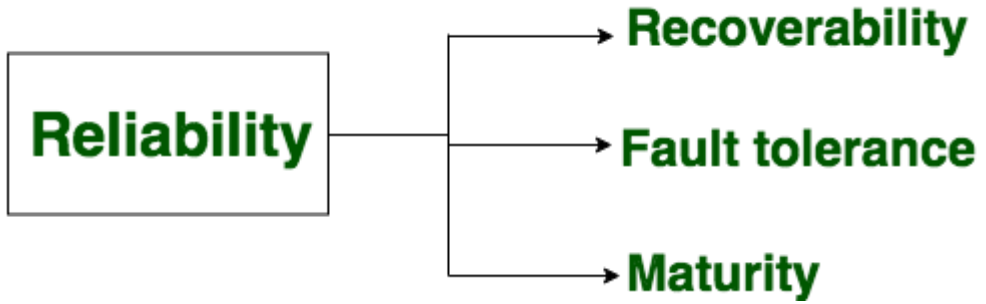
Software is defined as a collection of computer programs, procedures, rules, and data. Software Characteristics are classified into six major components:



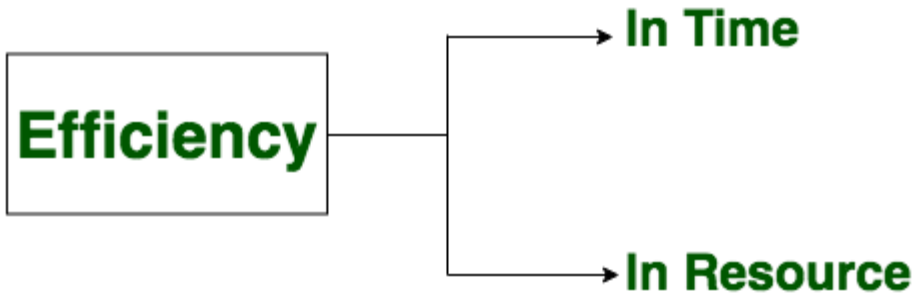
- **Functionality:**
  It refers to the degree of performance of the software against its intended purpose. Required functions are:



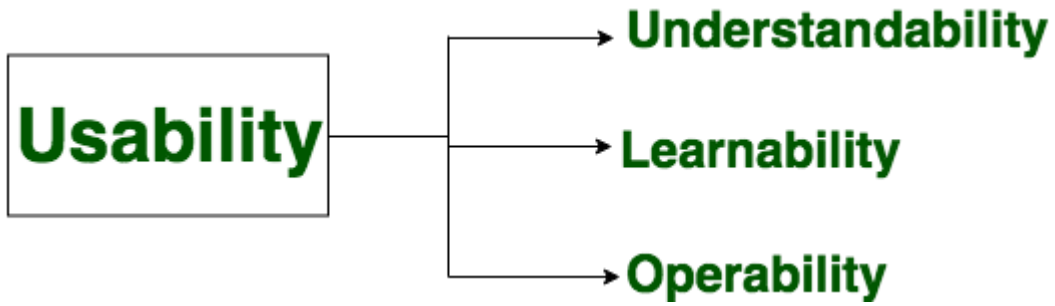
- **Reliability:**
  A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time. Required functions are:



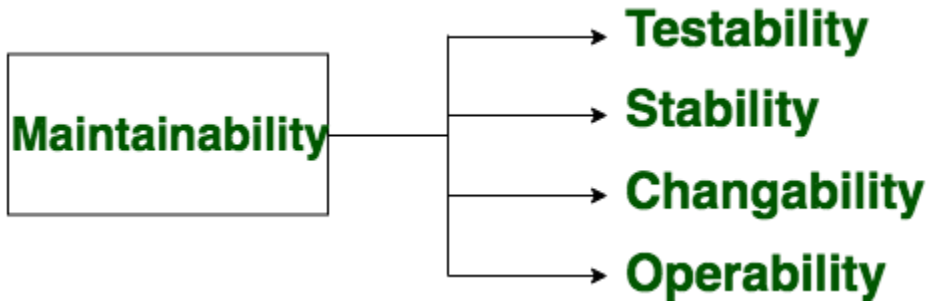
- **Efficiency:**
  It refers to the ability of the software to use system resources in the most effective and efficient manner. The software should make effective use of storage space and executive command as per desired timing requirements. Required functions are:



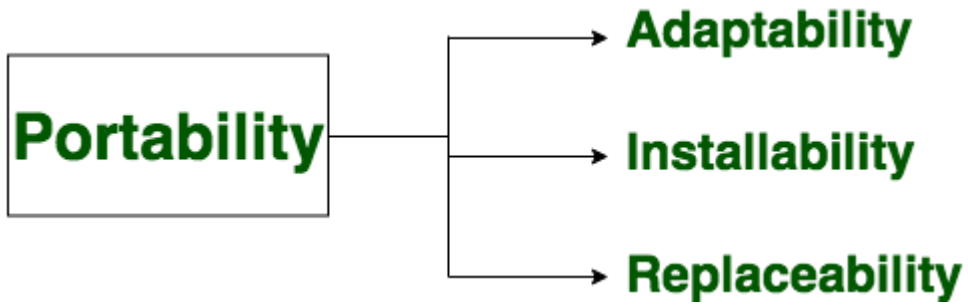
- **Usability:**
  It refers to the extent to which the software can be used with ease. the amount of effort or time required to learn how to use the software. Required functions are:



- **Maintainability:**
  It refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors. Required functions are:



- **Portability:**
  A set of attributes that bears on the ability of software to be transferred from one environment to another, without or minimum changes. Required functions are:

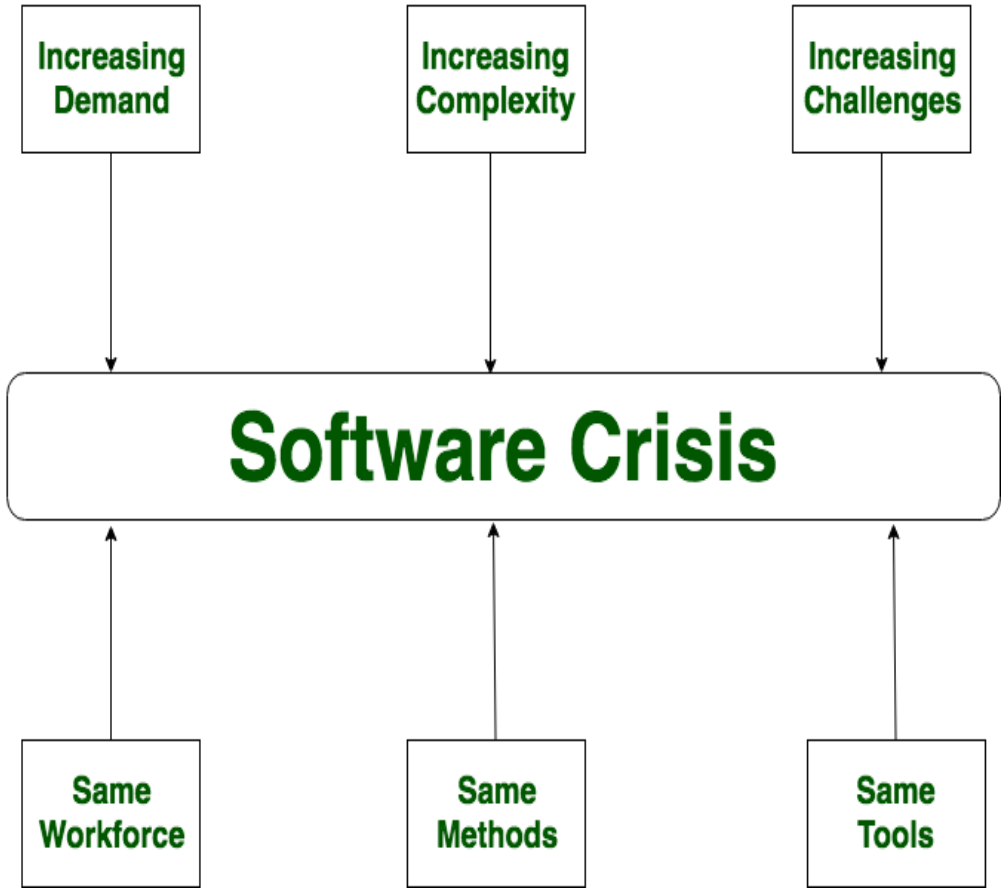


Apart from above mention qualities of software, there are various characteristics of software in software engineering:

1. **Software is developed or engineered; it is not manufactured in the classical sense:**
   - Although some similarities exist between software development and hardware manufacturing, few activities are fundamentally different.
   - In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems than software.
2. **The software doesn't "wear out.":**
   - Hardware components suffer from the growing effects of many other environmental factors. Stated simply, the hardware begins to wear out.
   - Software is not susceptible to the environmental maladies that cause hardware to wear out.
   - When a hardware component wears out, it is replaced by a spare part.
   - There are no software spare parts.
   - Every software failure indicates an error in design or in the process through which design was translated into machine-executable code. Therefore, the software maintenance tasks that accommodate requests for change involve considerably more complexity than hardware maintenance. However, the implication is clear—the software doesn't wear out. But it does deteriorate.
3. **The software continues to be custom-built:**
   - A software part should be planned and carried out with the goal that it tends to be reused in various projects.
   - Current reusable segments encapsulate the two information and the preparation that is applied to the information, empowering the programmer to make new applications from reusable parts.
   - In the hardware world, component reuse is a natural part of the engineering process.

Software Crisis

Software Crisis is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to using the same workforce, same methods, same tools even though rapidly increasing in software demand, the complexity of software, and software challenges. With the increase in the complexity of software, many software problems arise because existing methods were insufficient. If we will use the same workforce, same methods, and same tools after the fast increase in software demand, software complexity, and software challenges, then there arise some problems like software budget problems, software efficiency problems, software quality problems, software managing and delivering problem, etc. This condition is called a **software crisis.**



Causes of Software Crisis:

The cost of owning and maintaining software was as expensive as developing the software

- At that time Projects were running over-time
- At that time Software was very inefficient
- The quality of the software was low quality
- Software often did not meet user requirements
- The average software project overshoots its schedule by half
- At that time Software was never delivered
- Non-optimal resource utilization.
- Difficult to alter, debug, and enhance.
- The software complexity is harder to change.

Solution of Software Crisis:

There is no single solution to the crisis. One possible solution to a software crisis is *Software Engineering* because software engineering is a systematic, disciplined, and quantifiable approach. For preventing software crises, there are some guidelines:

- Reduction in software over budget.
- The quality of software must be high.
- Less time is needed for a software project.
- Experienced and skilled people working over the software project.
- Software must be delivered.
- Software must meet user requirements.

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.

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:

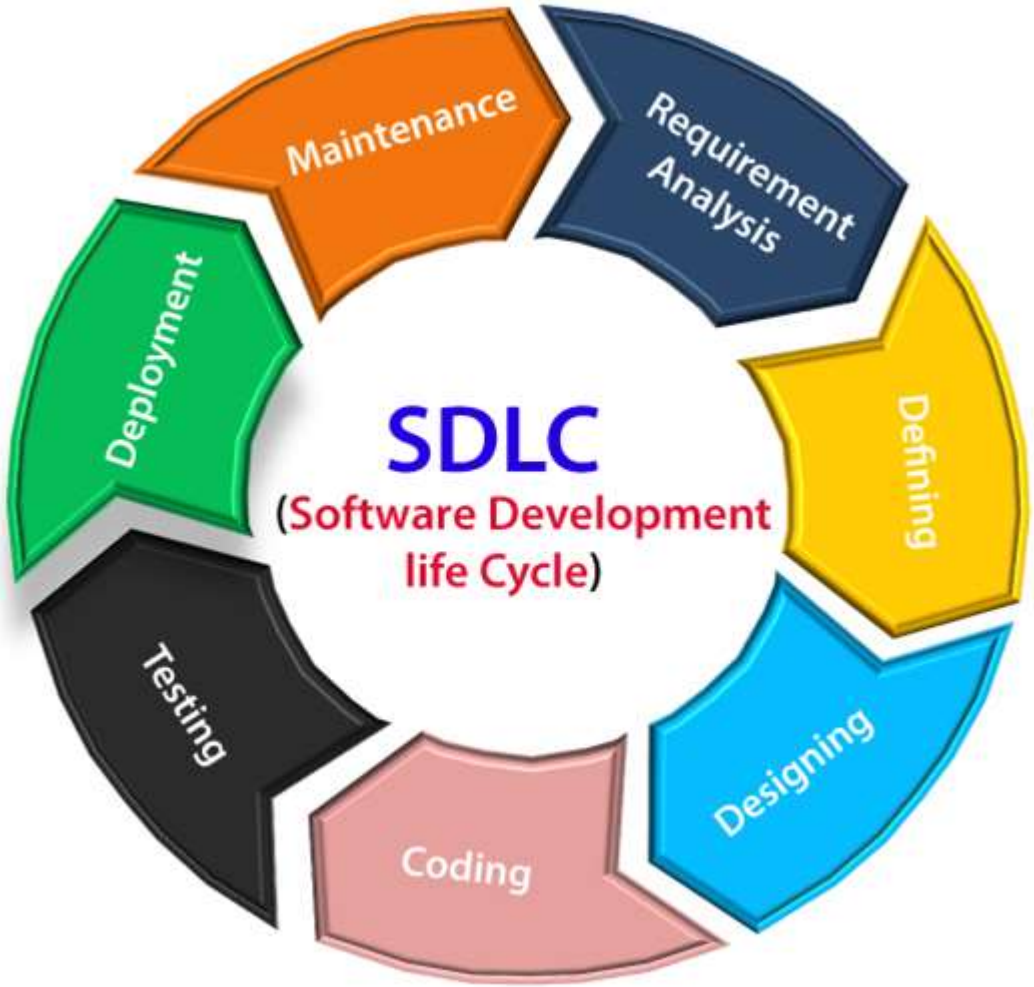
The stages of SDLC are as follows:
Stage1: Planning and requirement analysis

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.

Difference between Software Engineering Process and Conventional Engineering Process :

S.No.	Software Engineering Process	Conventional Engineering Process
1.	Software Engineering Process is a process which majorly involves computer science, information technology and discrete mathematics.	Conventional Engineering Process is a process which majorly involves science, mathematics and empirical knowledge.
2.	It is mainly related with computers, programming and writing codes for building applications.	It is about building cars, machines, hardware, buildings etc.
3.	In Software Engineering Process construction and development cost is low.	In Conventional Engineering Process construction and development cost is high.
4.	It can involve the application of new and untested elements in software projects.	It usually applies only known and tested principles to meet product requirements.
5.	In Software Engineering Process, most development effort goes into building new designs and features.	In Conventional Engineering Process, most development efforts are required to change old designs.
6.	It majorly emphasize on quality.	It majorly emphasize on mass production.

THE RELATIONSHIP OF SOFTWARE ENGINEERING TO OTHER AREAS OF COMPUTER SCIENCE

Software engineering has emerged as an important field within computer science. Indeed, there is a relationship between it and many other areas in computer science: these areas both influence and get influenced by software engineering. In the following subsections, we explore the relationship between software engineering and some of the other important fields of computer science.

Programming Languages

The influence of software engineering on programming languages is rather evident. Programming languages are the central tools used in software development. As a result, they gave profound influence on how well we can achieve our software engineering foals. In turn, these foals influence the development of programming languages. The most notable example of this influence in recent programming languages is the support of modularity features, such as separate and independent compilation and the separation of specification from implementation in order to support team development of large software. The Ada programming language, for example, supports the development of packages allowing the separation of the packages interface from its implementation and librarian of packages that can be used as components in he development of independent software systems. This is a step towards making it possible to build software by chord from a catalogue of available components and combining them similar to the way hardware is built. Design of object oriented programming languages like C++ in another major development for one purpose of rapid software development.

Operating Systems

The influence of operating systems on software engineering is quite strong primarily because operating systems were the first really large software systems built, and therefore they were the first instances of software that needed to be engineered . Many of the first software design ideas originated from early attempts at building operating systems.
Examples of the influence of software engineering techniques on the structures of operating systems can be seen in portable operating systems and operating systems that are structured to contain a small protected kernel that provided a minimum of functionality for interfacing with the hardware and a non-protected part that provided the majority of the functionality previously associated with operating systems. For example, the non-protected part may allow the user to control the paging scheme, which has traditionally been viewed as an integral part of the operating system.
Similarly, in early operating systems, the command language interpreter was an integral part of the operation system. Today, it is viewed as just another utility program. This allows, for example each user to have a personalized version of the interpreter. On many UNIX systems, there are at least three different such interpreters.

Database

Database represent another class of large software systems whose development has influenced software engineering through the discovery of new design techniques. Perhaps the most important influence of the database field on software engineering is through the notion of data-indigence, which is yet another instance of the separate of specification afro implementation. The database allows applications to be written that user data without worrying about the underlying representation of the data.

Another interring impact of database technology on software engineering is that it allows database system to be used ac components of large software systems. Since databases have solved the many problems associated with the management of concurrent access to large amounts of information by multiple users, there is no need to reinvent these solutions when we are building a software systems – we can simply use an existing data – base system as a component.

One interesting influence of software engineering on data-base technology has its roots in early attempts to use databases to support software development environments. This experience showed that traditional database technology was incapable of dealing with the problems posed by software engineering processes. For example, the following requirements are not handled well a by traditional data bases : strong large structured objects such as sources programs or use manuals; storing large unstructured objects such as object code and executable code; maintaining different versions of the same object; and strong objects, such as a product, with many large structured and unstructured fields, such as source code, object code, and a user manual.

There is presently considerable work going on in the database area to address such problems, ranging from introducing new models for databases to adapting current data-base models

Artificial Intelligence

Artificial intelligence is another field that has exerted influence on software engineering. The technique supported by artificial intelligence include the user of logic in both software specifications and programming languages.

The logic orientation seems to be filling the gap between specification and implementation by raising the level of implementation language higher than before. The logic approach to specification and programming is also called declarative. The idea is that we declare the specifications or requirements rather than specifying them procedurally; the declarative description is then executable. Logic programming languages such as PROLOG help us follow this methodology.

Software engineering techniques have been used in newer artificial intelligence systems – for example systems. These systems are modularised, with a clear separation between the facts known by the expert systems and the rules used by the systems for processing the facts – for example, a rule to decide on a course of action. The separation has enabled the building and commercial availability of expert system shells that include the rules only. A user can apply the shell to an application of interest in supplying application-specific facts. The idea is that the expertise about the application is provided by the user and the general principles of how to apply expertise to any problem are provided by the shell.

10 key differences between computer science and software engineering

1. Core educational studies

Computer science and software engineering may share some overlapping core studies, however, when studying computer science students may typically complete courses that focus on the computing, analysis, storage and application of data and data systems of computer programs and software. Software engineering focuses on taking these principles and applying them to the product design, interplay, performance and other functional aspects of computer programs.

For instance, the key difference here is that computer science education program focuses on the science behind making computers work, while software engineering applies those scientific and mathematical principles to the building, designing and implementation of hardware and software programs.

2. Career paths

Another key difference between a computer science degree and a software engineering degree is the variety of options in career paths. Generally, computer science degrees may offer candidates a broad range of job options in the informational technology industry, from computer programming for website design and working in IT support roles to working as a game developer. Conversely, a degree in software engineering can narrow a candidate's career path to specialized roles in corporations, companies and even mid-sized businesses developing and building applications and software programs.

3. Common tasks in the job

Typically, a computer scientist may complete daily tasks that analyze and monitor the processes of new and developing computer applications, either working for a software firm or independently contracting with different companies. Computer scientists may be responsible for maintaining their company's security networks, data systems or other computing systems that the business relies on to operate. A software engineer might be an employee of a similar company, but they may work to develop and design the specific software programs their organization might need for its operations.

4. Hardware and software interaction

Computer science may deal with the interaction between software programs with computer hardware. For instance, a computer scientist might determine ways to create software programs that are compatible with computer hardware. A software engineer, however, deals only with software programs, specifically creating, maintaining, testing and producing software products.

5. Software design

Software design is another aspect where computer science and software engineering can differ. When approaching software design, a computer scientist may typically work with theories and algorithms for how a program works, how it may be best designed and how to apply programming languages to the application. When software engineers work with software design, they may use a computer scientist's information and analyses to design the framework to build a specific program.

Furthermore, a software engineer can be given specific program requirements to use when building a software design, whereas a computer scientist works with computer languages and mathematical calculations to make decisions about how a program should be designed.

6. Programming and development

Another difference between computer science and software engineering is the application of software programming and development. Computer science will focus on computing and calculating the best ways to program software as well as finding calculations that allow engineers and developers to build software programs that meet product requirements. Software engineers essentially use the analysis and outlines from computer scientists to aid in the full development and construction of new frameworks and software programs.

7. Engineering concepts

While computer science majors may study aspects of mathematical engineering as it applies to computers and computing systems, engineering principles and concepts are generally paired with software engineering studies. Computer science may require knowledge of engineering principles, such as building entirely new frameworks for applications to run off of, however, software engineering is where real-world engineering concepts are applied.

8. Scientific theories

Computer science is also different from software engineering because it focuses heavily on scientific theories behind computer operations, computing and data systems as well as how software is designed. Software engineering, however, can use these theories to aid in the design and processes of building frameworks, hardware and software programs and applications. So while computer science studies and develops theories behind computer operations, software engineering applies these theories to build real-world computer applications.

9. Product management

Computer science may typically focus on the complexity and algorithms of software programs as well as other analytics like the computational science of programming, structures of visuals and graphics and user interaction. The approach an engineer takes in product management includes the design process, application, evaluation, automation testing and quality assurance checks of a software product.

10. Computer coding

Computer science and software engineering may both focus on computer coding and languages, however, software engineering may focus more heavily on learning coding to use it when developing and building software. A computer scientist may focus on coding as it relates to computer languages, and they may also use various computer coding to calculate compatibility between hardware and software applications.

THE EVOLVING ROLE OF SOFTWARE

Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation.

As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments). Software delivers the most important product of our time—information.

Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.

The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems. The lone programmer of an earlier era has been replaced by a team of software specialists, each focusing on one part of the technology required to deliver a complex application.

Changing Nature of Software:

The nature of software has changed a lot over the years.

1.System software: Infrastructure software come under this category like compilers, operating systems, editors, drivers, etc. Basically system software is a collection of programs to provide service to other programs.

2. Real time software: These software are used to monitor, control and analyze real world events as they occur. An example may be software required for weather forecasting. Such software will gather and process the status of temperature, humidity and other environmental parameters to forcast the weather.

3. Embedded software: This type of software is placed in “Read-Only- Memory (ROM)”of the product and control the various functions of the product. The product could be an aircraft, automobile, security system, signalling system, control unit of power plants, etc. he embedded software handles hardware components and is also termed as intelligent software .

4. Business software : This is the largest application area. The software designed to process business applications is called business software. Business software could be payroll, file monitoring system, employee management, account management. It may also be a data warehousing tool which helps us to take decisions based on available data. Management information system, enterprise resource planning (ERP) and such other software are popular examples of business software.

5. Personal computer software: The software used in personal computers are covered in this category. Examples are word processors, computer graphics, multimedia and animating tools, database management, computer games etc. This is a very upcoming area and many big organisations are concentrating their effort here due to large customer base.

6. Artificial intelligence software: Artificial Intelligence software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis. Examples are expert systems, artificial neural network, signal processing software etc.

7. Web based software: The software related to web applications come under this category. Examples are CGI, HTML, Java, Perl, DHTML etc.