Chapter

1

INTRODUCTION OF SOFTWARE ENGINEERING

1.1 ROLE OF SOFTWARE

Software is defined as a collection of programs, documentation and operating procedures. The **Institute of Electrical and Electronic Engineers** (IEEE) defines software as a 'collection of computer programs, procedures, rules and associated documentation and data.' It possesses no mass, no volume, and no colour, which makes it a non-degradable entity over a long period. Software does not wear out or get tired.

Software controls, integrates, and manages the hardware components of a computer system. It also instructs the computer what needs to be done to perform a specific task and how it is to be done. For example, software instructs the hardware how to print a document, take input from the user, and display the output.

Computer works only in response to instructions provided externally. Usually, the instructions to perform some intended tasks are organized into a program using a programming language like C, C++, Java, etc., and submitted to computer. Computer interprets and executes these instructions and provides response to the user accordingly. A set of programs intended to provide users with a set of interrelated functionalities is known as a **software package**. For example, an accounting software package such as Tally provides users the functionality to perform accounting-related activities.

1.1.1 Classification of Software

Software can be applied in countless fields such as business, education, social sector, and other fields. It is designed to suit some specific goals such as data processing, information sharing, communication, and so on. It is classified according to the range of potential of applications. These classifications are listed below.

(*a*) **System software:** This class of software manages and controls the internal operations of a computer system. It is a group of programs, which is responsible for using computer resources efficiently and effectively. For example, an operating system is a system software, which controls the hardware, manages memory and multitasking functions, and acts as an interface between application programs and the computer.

(*b*) **Real-time software:** This class of software observes, analyzes, and controls real world events as they occur. Generally, a real-time system guarantees a response to an external event within a specified period of time. An example of real-time software is the software used for weather forecasting that collects and processes parameters like temperature and humidity from the external environment to forecast the weather. Most of the defence organizations all over the world use real-time software to control their military hardware

(*c*) **Business software:** This class of software is widely used in areas where management and control of financial activities is of utmost importance. The fundamental component of a business system comprises payroll, inventory, and accounting software that permit the user to access relevant data from the database. These activities are usually performed with the help of specialized business software that facilitates efficient framework in business operations and in management decisions.

(*d*) **Engineering and scientific software:** This class of software has emerged as a powerful tool in the research and development of next generation technology. Applications such as the study of celestial bodies, under-surface activities, and programming of an orbital path for space shuttles are heavily dependent on engineering and scientific software. This software is designed to perform precise calculations on complex numerical data that are obtained during real-time environment.

(*e*) **Artificial intelligence (AI) software:** This class of software is used where the problem-solving technique is non-algorithmic in nature. The solutions of such problems are generally non-agreeable to computation or straightforward analysis. Instead, these problems require specific problem solving strategies that include expert system, pattern recognition, and game playing techniques. In addition, they involve different kinds of search techniques which include the use of heuristics. The role of artificial intelligence software is to add certain degrees of intelligence to the mechanical hardware in order to get the desired work done in an agile manner.

(*f*) **Web-based software:** This class of software acts as an interface between the user and the Internet. Data on the Internet is in the form of text, audio, or video format, linked with hyperlinks. Web browser is a software that retrieves web pages from the Internet. The software incorporates executable instructions written in special scripting languages such as CGI or ASP. Apart from providing navigation on the Web, this software also supports additional features that are useful while surfing the Internet.

(*g*) **Personal computer (PC) software:** This class of software is used for both official and personal use. The personal computer software market has grown over in the last two decades from normal text editor to word processor and from simple paintbrush to advanced image-editing software. This software is used predominantly in almost every field, whether it is database management system, financial accounting package, or multimedia-based software. It has emerged as a versatile tool for routine applications.

1.2 WHAT IS SOFTWARE ENGINEERING ?

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. Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.

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.**

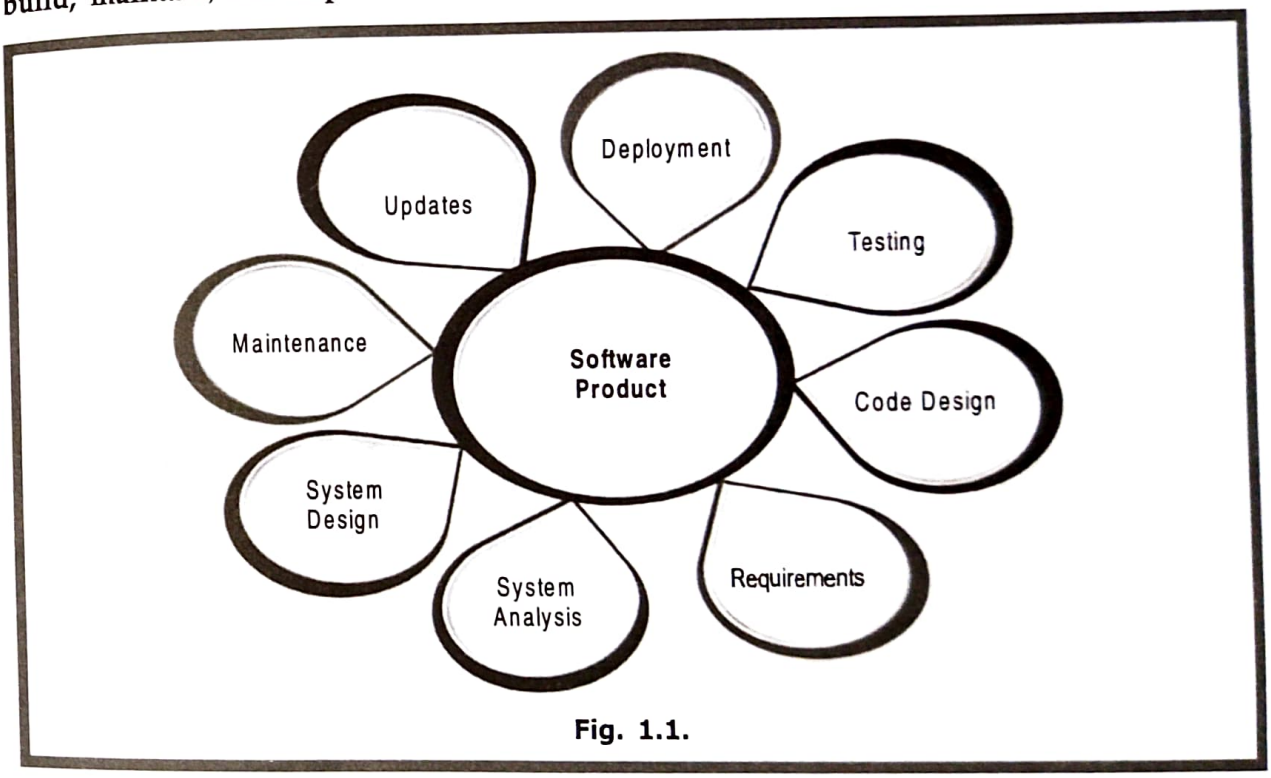

Fig. 1.1.

Software Engineering is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

1.3 DEFINITIONS OF SOFTWARE ENGINEERING

Software engineering is defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements.

Let's look at the various definitions of software engineering:

- IEEE, in its standard 610.12-1990, defines software engineering as the application of a systematic, disciplined, which is a computable approach for the development, operation, and maintenance of software.
- Fritz Bauer defined it as 'the establishment and used standard engineering principles. It helps you to obtain, economically, software which is reliable and works efficiently on the real machines'.
- Boehm defines software engineering, which involves, 'the practical application of scientific knowledge to the creative design and building of computer programs. It also includes associated documentation needed for developing, operating, and maintaining them.'
- It is a branch of engineering that deals with the development of software products. It operates within a set of principles, best practices, and methods that have been carefully honed throughout the years, changing as software and technology change.

Software engineering leads to a product that is reliable, efficient, and effective at what it does. While software engineering can lead to products that do not do this, the product will almost always go back into the production stage.

Software engineering is the branch of computer science that deals with the design, development, testing, and maintenance of software applications. Software engineers apply engineering principles and knowledge of programming languages to build software solutions for end users.

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

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."

1.4 KEY PRINCIPLES OF SOFTWARE ENGINEERING

- **Modularity:** Breaking the software into smaller, reusable components that can be developed and tested independently.
- **Abstraction:** Hiding the implementation details of a component and exposing only the necessary functionality to other parts of the software.
- **Encapsulation:** Wrapping up the data and functions of an object into a single unit, and protecting the internal state of an object from external modifications.
- **Reusability:** Creating components that can be used in multiple projects, which can save time and resources.
- **Maintenance:** Regularly updating and improving the software to fix bugs, add new features, and address security vulnerabilities.
- **Testing:** Verifying that the software meets its requirements and is free of bugs.
- **Design Patterns:** Solving recurring problems in software design by providing templates for solving them.

- **Agile methodologies:** Using iterative and incremental development processes that focus on customer satisfaction, rapid delivery, and flexibility.
- **Continuous Integration & Deployment:** Continuously integrating the code changes and deploying them into the production environment.

Software engineering is a rapidly evolving field, and new tools and technologies are constantly being developed to improve the software development process. 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.

Software Engineering is mainly used for large projects based on software systems rather than single programs or applications. The main goal of Software Engineering is to develop software applications for improving quality, budget, and time efficiency. 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.

1.5 MAIN ATTRIBUTES OF SOFTWARE ENGINEERING

Software Engineering is a systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance of a software system. There are Four main Attributes of Software Engineering.

- Efficiency
- Robustness
- Reliability
- Maintainability

Dual Role of Software

There is a dual role of software in the industry. The first one is as a product and the other one is as a vehicle for delivering the product. We will discuss both of them.

1. **As a product:**
   - It delivers computing potential across networks of Hardware.
   - It enables the Hardware to deliver the expected functionality.
   - It acts as an information transformer because it produces, manages, acquires, modifies, displays, or transmits information.
2. **As a vehicle for delivering a product:**
   - It provides system functionality (e.g., payroll system)
   - It controls other software (e.g., an operating system)
   - It helps build other software (e.g., software tools)

1.6 OBJECTIVES OF SOFTWARE ENGINEERING

- **Maintainability:** It should be feasible for the software to evolve to meet changing requirements.
- **Efficiency:** The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
- **Correctness:** A software product is correct if the different requirements specified in the SRS Document have been correctly implemented.

- **Reusability:** A software product has good reusability if the different modules of th product can easily be reused to develop new products.
- **Testability:** Here software facilitates both the establishment of test criteria and th evaluation of the software with respect to those criteria.
- **Reliability:** It is an attribute of software quality. The extent to which a program ca be expected to perform its desired function, over an arbitrary time period.
- **Portability:** In this case, the software can be transferred from one computer syste or environment to another.
- **Adaptability:** In this case, the software allows differing system constraints and th user needs to be satisfied by making changes to the software.
- **Interoperability:** Capability of 2 or more functional units to process data cooperatively

1.7 ADVANTAGES OF SOFTWARE ENGINEERING

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

- **Improved Quality:** By following established software engineering principles and techniques, the software can be developed with fewer bugs and higher reliability.
- **Increased Productivity:** Using modern tools and methodologies can streamline the development process, allowing developers to be more productive and complete projects faster.
- **Better Maintainability:** Software that is designed and developed using sound software engineering practices is easier to maintain and update over time.
- **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.
- **Increased Customer Satisfaction:** By involving customers in the development process and developing software that meets their needs, software engineering can help to increase customer satisfaction.
- **Better Team Collaboration:** By using Agile methodologies and continuous integration, software engineering allows for better collaboration among development teams.
- **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.
- **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.

In summary, software engineering offers a structured and efficient approach to software development, which can lead to higher-quality software that is easier to maintain and adapt to changing requirements. This can help to improve customer satisfaction and reduce costs, while also promoting better collaboration among development teams.

1.8 DISADVANTAGES OF SOFTWARE ENGINEERING

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

- **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.
- **Limited flexibility:** Following established software engineering principles and methodologies can be rigid and may limit the ability to quickly adapt to changing requirements.
- **Bureaucratic**: Software Engineering can create an environment that is bureaucratic, with a lot of processes and paperwork, which may slow down the development process.
- **Complexity**: With the increase in the number of tools and methodologies, software engineering can be complex and difficult to navigate.
- **Limited creativity:** The focus on structure and process can stifle creativity and innovation among developers.
- **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.
- **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.
- **High maintenance**: The software engineering process requires regular maintenance to ensure that the software is running efficiently, which can be costly and time-consuming.

In summary, software engineering can be expensive and time-consuming, and it may limit flexibility and creativity. However, the benefits of improved quality, increased productivity, and better maintainability can outweigh the costs and complexity. It's important to weigh the pros and cons of using software engineering and determine if it is the right approach for a particular software project.

1.9 IMPORTANCE OF SOFTWARE ENGINEERING

The importance of Software engineering is as follows:

1. **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.
2. **To minimize software cost:** Software needs a lot of hardwork and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.

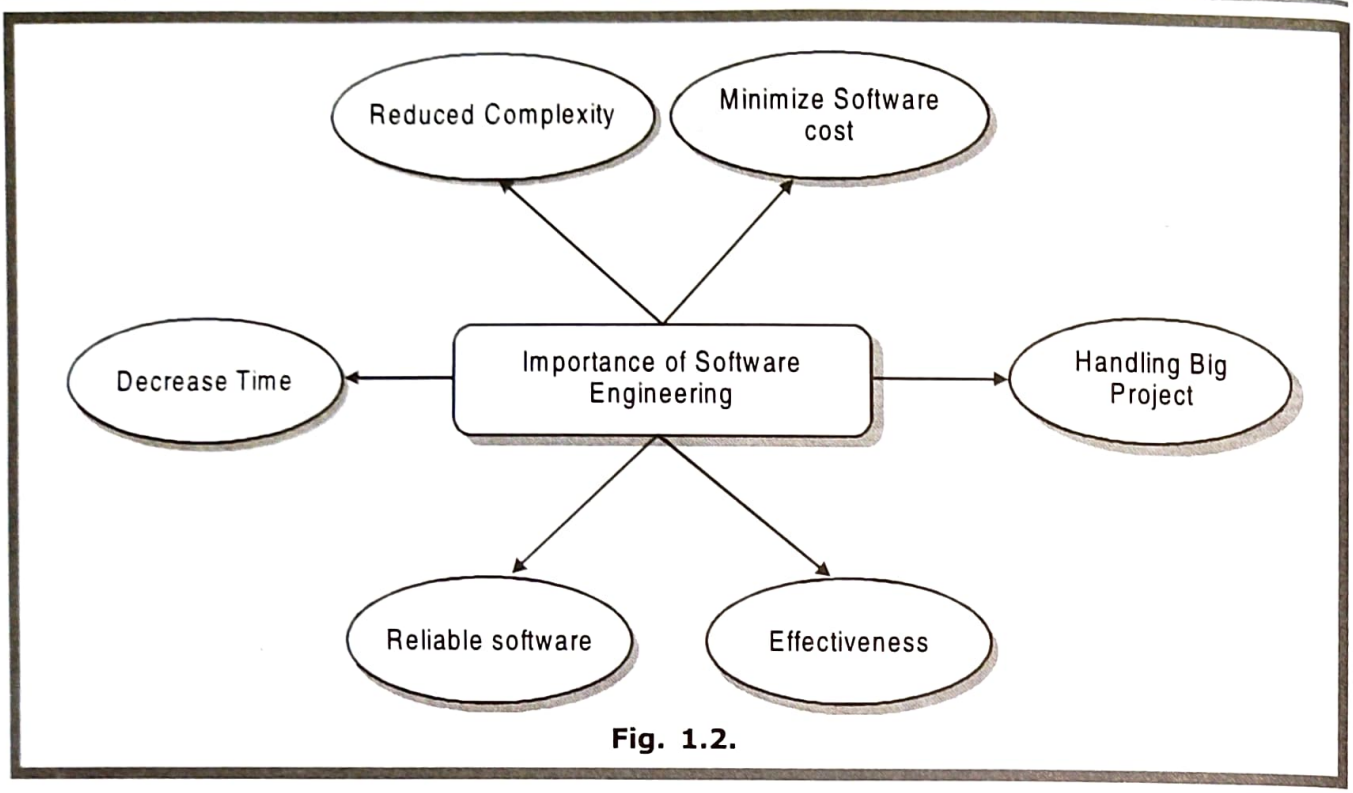

Fig. 1.2.

3. **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So if you are making your software according to the software engineering method, then it will decrease a lot of time.
4. **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So to handle a big project without any problem, the company has to go for a software engineering method.
5. **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
6. **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So Software becomes more effective in the act with the help of software engineering.

1.10 NEED OF SOFTWARE ENGINEERING

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working. Following are some of the needs stated:

- **Large software :** It is easier to build a wall than a house or building, likewise, as the size of the software becomes large, engineering has to step to give it a scientific process.
- **Scalability :** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost :** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But, cost of the software remains high if proper process is not adapted.
- **Dynamic Nature :** Always growing and adapting nature of the software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where the software engineering plays a good role.
- **Quality Management :** Better process of software development provides better and quality software product.
- **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.

1.11 SOFTWARE ENGINEERING CHARACTERISTICS

Software is defined as a collection of computer programs, procedures, rules, and data. Software Characteristics are classified into six major components. Software engineering is the process of designing, developing, testing, and maintaining software. In this article, we will look into the characteristics of Software in detail. We have also discussed each component of Software characteristics in detail. The characteristics of the software include:

- It is intangible, meaning it cannot be seen or touched.
- It is non-perishable, meaning it does not degrade over time.
- It is easy to replicate, meaning it can be copied and distributed easily.

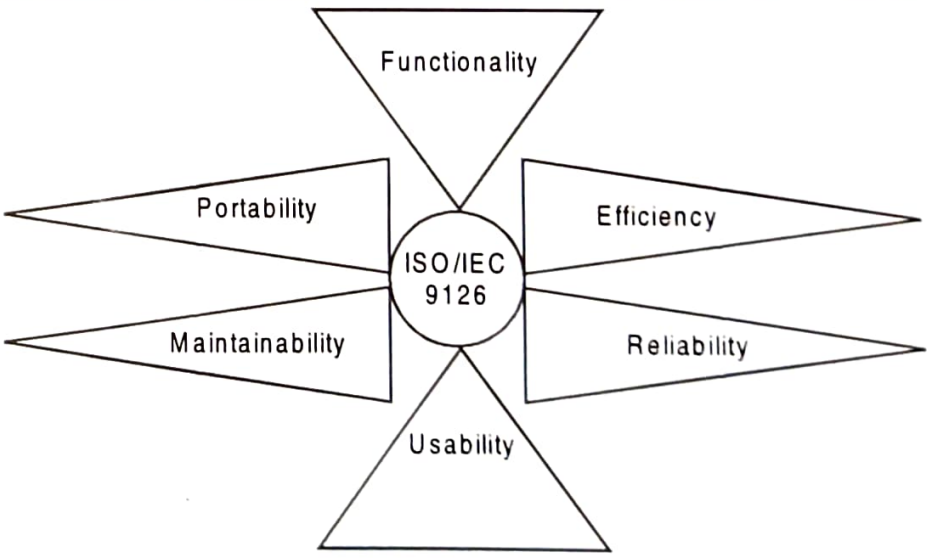

Fig. 1.3. Software Characteristics

- It can be complex, meaning it can have many interrelated parts and features.
- It can be difficult to understand and modify, especially for large and complex systems.
- It can be affected by changing requirements, meaning it may need to be updated or modified as the needs of users change.
- It can be affected by bugs and other issues, meaning it may need to be tested and debugged to ensure it works as intended.

1.12 COMPONENTS OF SOFTWARE ENGINEERING

The quality of a software product is determined by what it offers and how easily it can be used. Software is judged by different people on different grounds. Customers, for instance, want software that meets their specific needs. Similarly, developers engaged in designing, coding, and maintaining the software determine the quality of the software by assessing its internal characteristics. Let's check them out.

There are basically 6 components of Software Characteristics that are discussed here. We will discuss each one of them in detail.

1.12.1 Functionality

It refers to the degree of performance of the software against its intended purpose.

Functionality refers to the set of features and capabilities that a software program or system provides to its users. It is one of the most important characteristics of software, as it determines the usefulness of the software for the intended purpose. Examples of functionality in software include:

- Data storage and retrieval
- Data processing and manipulation
- User interface and navigation
- Communication and networking
- Security and access control
- Reporting and visualization
- Automation and scripting

The more functionality a software has, the more powerful and versatile it is, but also the more complex it can be. It is important to balance the need for functionality with the need for ease of use, maintainability, and scalability.

Required functions are:

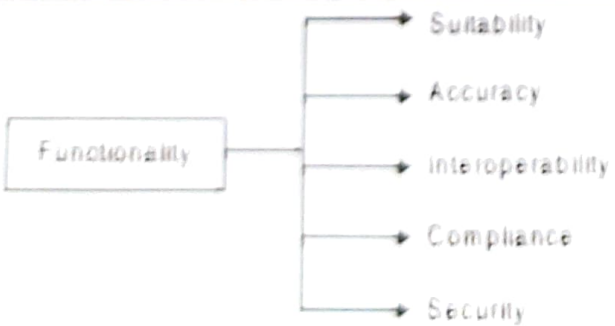

Fig. 1.4.

1.12.2 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

Reliability is a characteristic of software that refers to its ability to perform its intended functions correctly and consistently over time. Reliability is an important aspect of software quality, as it helps ensure that the software will work correctly and not fail unexpectedly.

Examples of factors that can affect the reliability of software include:

1. Bugs and errors in the code
2. Lack of testing and validation
3. Poorly designed algorithms and data structures
4. Inadequate error handling and recovery
5. Incompatibilities with other software or hardware

To improve the reliability of software, various techniques and methodologies can be used, such as testing and validation, formal verification, and fault tolerance.

Software is considered reliable when the probability of it failing is low and it is able to recover from the failure quickly, if any.

Required functions are:

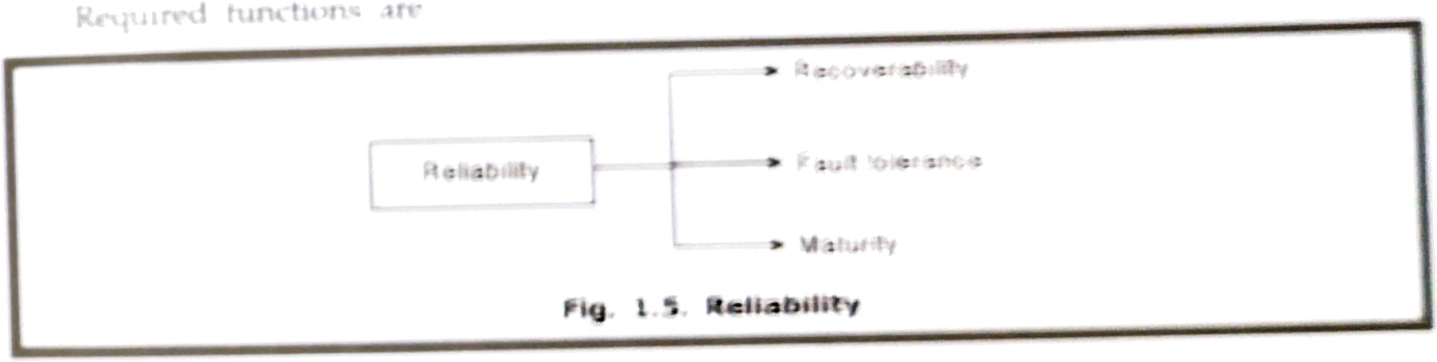

Fig. 1.5. Reliability

1.12.3 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.

Efficiency is a characteristic of software that refers to its ability to use resources such as memory, processing power, and network bandwidth in an optimal way. High efficiency means that a software program can perform its intended functions quickly and with minimal use of resources, while low efficiency means that a software program may be slow or consume excessive resources.

Examples of factors that can affect the efficiency of the software include:

1. Poorly designed algorithms and data structures
2. Inefficient use of memory and processing power
3. High network latency or bandwidth usage
4. Unnecessary processing or computation
5. Unoptimized code

1.12

To improve the efficiency of software, various techniques, and methodologies can be used, such as performance analysis, optimization, and profiling.

Efficiency is important in software systems that are resource-constrained, high-performance, and real-time systems. It is also important in systems that need to handle a large number of users or transactions simultaneously.

Required functions are:

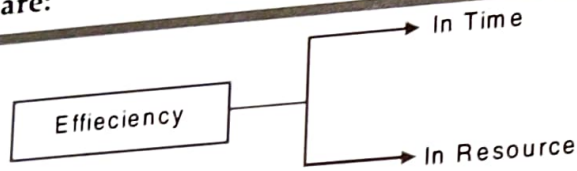

Fig. 1.6. Efficiency

1.12.4 Usability (User-friendly)

The user-friendliness of the software is characterized by its ease of use. In other words, learning how to use the software should require less effort or time. Navigating the software is extremely important since it helps determine the journey the user takes within the software. This is imperative to ensure visitors remain on your website and have a positive experience, which leads to an increase in sales and brand loyaltyAn important indicator of a good piece of software is its user interface, i.e., the smooth flow of its design. A product with a great UI (User Interface) design is more likely to get noticed than one without. If a software program isn't user-friendly, users may have trouble navigating the software and using some of its features. Software should require less time or effort to learn. Ideally, software should be easy to use even by people with no IT experience.. 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:

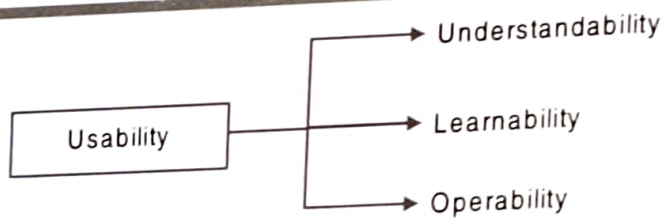

Fig. 1.7. Usability

1.12.5 Maintainability

Maintainability refers to how easily you can repair, improve and comprehend software code. In some ways, maintaining is similar to being flexible. Maintainability deals with the modification of errors and minor alterations to software code, while flexibility focuses on major functional extensions. It also involves maintaining the services and functionality of the software.

Most of the time, developers are not the ones who maintain the software. Therefore, good documentation is crucial, which includes code documentation, interface definitions, etc. The maintainability of software products is affected by the quality of the documentation. Typically, more than half of development budgets are spent on software maintenance.

Maintenance should therefore be integrated into the development lifecycle for effective software maintenance.It refers to the ease with which modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.

Required functions are:

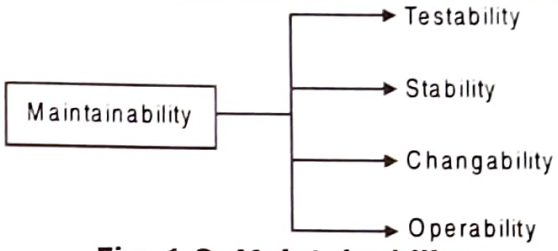

Fig. 1.8. Maintainability

1.12.6 Portability

Software portability is a critical factor that cannot be ignored. Portability refers to the ability to use software in different environments. This is the ease with which software can be ported from one platform to another without (or with minimal) changes, while obtaining similar results. As simple as it may sound, it refers to the ability of software to work on different hardware platforms without any (or little) modifications needed.

Furthermore, you should be aware that porting software to a new environment is comparatively cheaper than developing an equivalent application from scratch. There can be no doubt that portability is a crucial aspect of reducing development costs.A set of attributes that bears on the ability of software to be transferred from one environment to another, without minimum changes.

Required functions are:

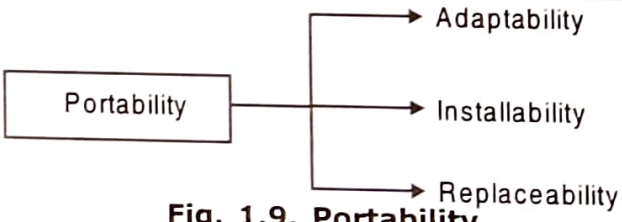

Fig. 1.9. Portability

1.12.7 Integrity

There are multiple interpretations of software integrity. Some people tend to associate integrity with security, believing it is resistant to hacks and privacy violations. To others, high integrity means that the software cannot be modified without authorization. Integrity is key for demonstrating the safety, security, and maintainability of your software. In addition, software that needs to be compliant with industry regulations and coding standards requires high code integrity. Achieving software integrity can be difficult. Yet, with the right practices to improve safety, security, and maintainability, the challenge can be easily overcome. In these days of increased security threats, all software must include this factor.

1.12.8 Flexibility

Software Flexibility refers to the ability of the software solution to adapt to potential or future changes in its requirements. When evaluating the flexibility of software, look at how simple it is to add, modify, or remove features without interfering with the current operation.

It is essential to keep up with rapidly changing markets, technologies, and customer needs. In software development, change is inevitable; it can arise during the development process itself or as the result of future requirements. Flexibility is therefore highly valued. Consequently, any software product must be scalable, flexible, and easily adaptable to future technology. When designing or building a software product, be sure to plan for these changes that are inevitably going to occur. Loose coupling of components is the key to creating highly flexible systems.

1.13 VARIOUS CHARACTERISTICS OF SOFTWARE IN SOFTWARE ENGINEERING

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.

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 the 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.

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 pieces of 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.

14 RELATIONSHIP OF SOFTWARE ENGINEERING WITH OTHER DISCIPLINES

Here, how software engineering related to other disciplines:

- **Computer Science:** Gives the scientific foundation for the software as electrical engineering mainly depends on physics.

- **Management Science:** Software engineering is labor-intensive work which demands both technical and managerial control. Therefore, it is widely used in management science.
- **Economics:** In this sector, software engineering helps you in resource estimation and cost control. Computing system must be developed, and data should be maintained regularly within a given budget.
- **System Engineering:** Most software is a component of a much larger system. For example, the software in an Industry monitoring system or the flight software on an airplane. Software engineering methods should be applied to the study of this type of systems.

1.15 SOFTWARE MYTHS

The development of software requires dedication and understanding on the developers' part. Many software problems arise due to myths that are formed during the initial stages of software development. Unlike ancient folklore that often provides valuable lessons, software myths propagate false beliefs and confusion in the minds of management, users and developers

Management Myths. Managers, who own software development responsibility, are often under strain and pressure to maintain a software budget, time constraints, improved quality, and many other considerations. Common management myths are listed in Table 1.1

Table 1.1

Myths	Realities
❒ The members of an organization can acquire all the information they require from a manual, which contains standards, procedures, arid principles.	❒ Standards arc often incomplete, inadaptable, and outdated ❒ Developers are often unaware of all the established standards. ❒ Developers rarely follow all the known standards because not all the standards tend to decrease The delivery time of software while maintaining its quality.
❒ If the project is behind schedule, increasing the number of programmers can reduce the time gap.	❒ Adding more manpower to the project, which is already behind schedule, further delays the project. ❒ New workers take longer to learn about the project as compared to those already working on the project.
❒ If the project is outsourced to a third party, the management can relax and let the other firm develop software for them.	❒ Outsourcing software to a third party does not help the organization, which is incompetent in managing and conrolling the software project internally. The organization invariably suffers when it outsources the software project

User Myths. In most cases, users tend to believe myths about the software because software managers and developers do not try to correct the false beliefs. These myths lead to false expectations and ultimately develop dissatisfaction among the users. Common user myths are listed in Table 1.2

Table 1.2

Myths	Realities
❒ Brief requirement stated in the initial process is enough to start development; detailed requirements can be added at the Later stages.	❒ Starting development with incomplete and ambiguous requirements often lead to software failure. In steady a complete and formal description of requirements is essential before starting development. ❒ Adding requirements at a later stage often inquires repeating the entire development process.
❒ Software is flexible; hence software requirement changes can be added during any phase of the development process.	❒ Incorporating change requests earlier in the development process costs lesser than those that occurs at later stages. This is because incorporating changes later may require redesigning arid extra resources.

Developer Myths. In the early days of software development, programming was viewed as an art, but now software development has gradually become an engineering discipline. However, developers still believe in some myths. Some of the common developer myths are listed in Table 1.3.

Table 1.3

Myths	Realities
❒ Software development is considered complete when the code is delivered.	❒ 50% to 70% of all the effort are expended after the software is delivered to the user.
❒ The success of a software project depends on the quality of the product produced.	❒ The quality of programs is not the only factor that makes the project successful instead the documentation and software configuration also play a crucial role.
❒ Software engineering requires un necessary documentation, which slows down the project.	❒ Software engineering is about creating quality at every level of the software project. Proper documentation enhances quality which results in reducing the amount of rework.
❒ The only product that is delivered after the completion of a project is the working program(s).	❒ The deliverables of a successful project includes not only the working program but also the documentation to guide the users for using the software.
❒ Software quality can be assessed only after the program is executed.	❒ The quality of software can be measured during any phase of development process by applying some quality assurance mechanism. One such mechanism is formal technical review that can be effectively used during each phase of development to uncover certain errors.

1.16 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, and same tools even though rapidly increasing software demand, the complexity of software, and software challenges. With the increase in software complexity, 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 management, and delivery problem, etc. This condition is called a **Software Crisis.**

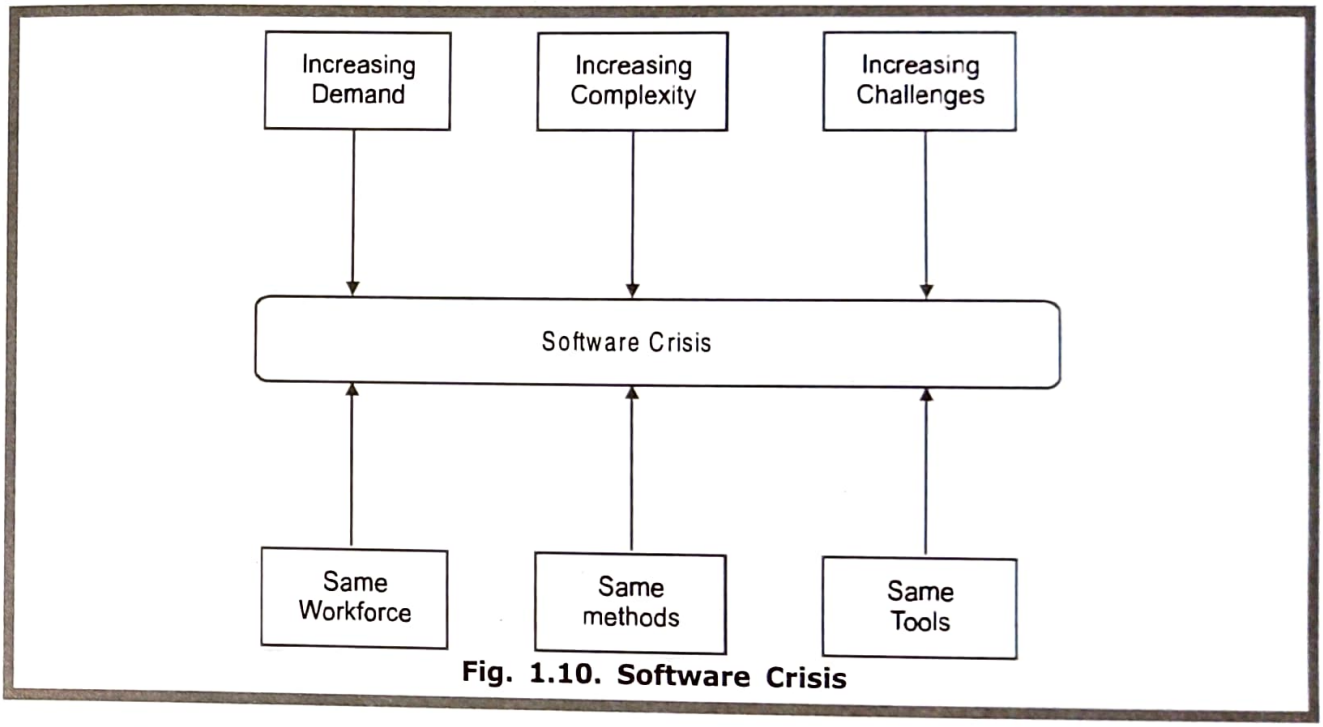

Fig. 1.10. Software Crisis

1.16.1 Causes of Software Crisis

- The cost of owning and maintaining software was as expensive as developing the software.
- At that time Projects were running overtime.
- 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.
- Challenging to alter, debug, and enhance.
- The software complexity is harder to change.

Factors Contributing to Software Crisis:

- Poor project management.
- Lack of adequate training in software engineering.
- Less skilled project members.
- Low productivity improvements.

1.16.2 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 the software must be high.
- Less time is needed for a software project.
- Experienced and skilled people working on the software project.
- Software must be delivered.
- Software must meet user requirements.

1.17 SOFTWARE EVOLUTION

Software Evolution is a term which refers to the process of developing software initially, then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities etc. The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers.

The cost and impact of these changes are accessed to see how much system is affected by the change and how much it might cost to implement the change. If the proposed changes are accepted, a new release of the software system is planned. During release planning, all the proposed changes (fault repair, adaptation, and new functionality) are considered.

A design is then made on which changes to implement in the next version of the system. The process of change implementation is an iteration of the development process where the revisions to the system are designed, implemented and tested.

The necessity of Software evolution: Software evaluation is necessary just because of the following reasons:

(a) **Change in requirement with time:** With the passes of time, the organization's needs and modus Operandi of working could substantially be changed so in this frequently changing time the tools(software) that they are using need to change for maximizing the performance.

(b) **Environment change:** As the working environment changes the things(tools) that enable us to work in that environment also changes proportionally same happens in the software world as the working environment changes then, the organizations need reintroduction of old software with updated features and functionality to adapt the new environment.

(c) **Errors and bugs:** As the age of the deployed software within an organization increases their preciseness or impeccability decrease and the efficiency to bear the increasing

complexity workload also continually degrades. So, in that case, it becomes necessary to avoid use of obsolete and aged software. All such obsolete Software's need to undergo the evolution process in order to become robust as per the workload complexity of the current environment.

(d) **Security risks:** Using outdated software within an organization may lead you to at the verge of various software-based cyber attacks and could expose your confidential data illegally associated with the software that is in use. So, it becomes necessary to avoid such security breaches through regular assessment of the security patches/ modules are used within the software. If the software isn't robust enough to bear the current occurring Cyber attacks so it must be changed (updated).

(e) **For having new functionality and features:** In order to increase the performance and fast data processing and other functionalities, an organization need to continuously evolutes the software throughout its life cycle so that stakeholders & clients of the product could work efficiently.

Laws used for Software Evolution:

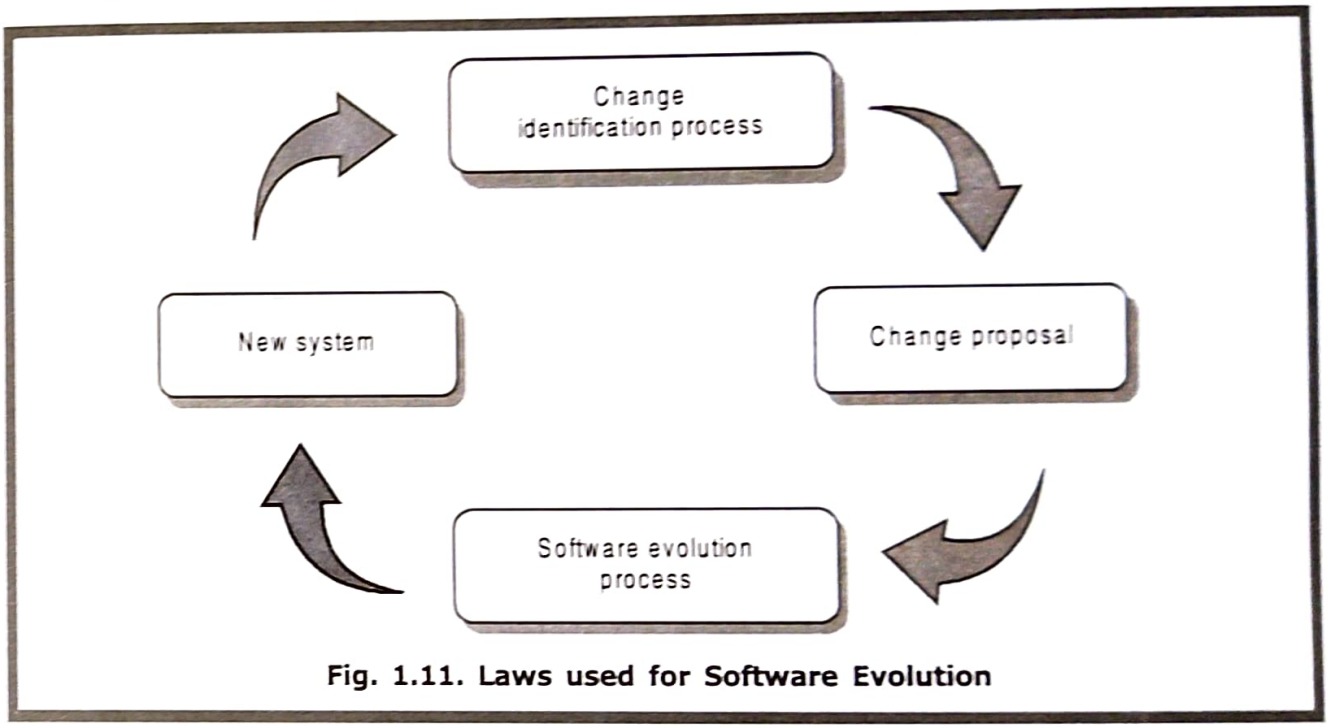

Fig. 1.11. Laws used for Software Evolution

1. **Law of continuing change:** This law states that any software system that represents some real-world reality undergoes continuous change or become progressively less useful in that environment.
2. **Law of increasing complexity:** As an evolving program changes, its structure becomes more complex unless effective efforts are made to avoid this phenomenon.
3. **Law of conservation of organization stability:** Over the lifetime of a program, the rate of development of that program is approximately constant and independent of the resource devoted to system development.
4. **Law of conservation of familiarity:** This law states that during the active lifetime of the program, changes made in the successive release are almost constant.

1.18 SOFTWARE EVOLUTION LAWS

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

1. **Static-type (S-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.
2. **Practical-type (P-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 obviously instant. For example, gaming software.
3. **Embedded-type (E-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.

1.19 E-TYPE SOFTWARE EVOLUTION

Lehman has given eight laws for E-Type software evolution :

1. **Continuing change:** An E-type software system must continue to adapt to the real world changes, else it becomes progressively less useful.
2. **Increasing complexity:** As an E-type software system evolves, its complexity tends to increase unless work is done to maintain or reduce it.
3. **Conservation of familiarity:** The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc., must be retained at any cost, to implement the changes in the system.
4. **Continuing growth:** In order for an E-type system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business.
5. **Reducing quality:** An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment.
6. **Feedback systems:** The E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.
7. **Self-regulation:** E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
8. **Organizational stability:** The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.
9. **Organizational stability:** The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.

1.20 EVOLUTION OF SOFTWARE ENGINEERING : FROM AN ART TO ENGINEERING DISCIPLINE

- ❒ Read
- ❒ Courses
- ❒ Discuss

Software Engineering is a systematic and cost-effective technique for Software Development. It is an engineering approach to develop Software.

For example: If someone wants to travel from Punjab to Delhi. There are two approaches one can follow to achieve the same result:

1. The normal approach is just to go out and catch the bus/train that is available.
2. A systematic approach is constructed as Firstly check on Google Maps about distance and after analyzing the timing of trains and buses online and after that, match user's preference suppose user have some work till 4:00 PM and trains slot are: 1:00 PM, 6:00 PM then the user will choose 6:00 PM time slot and reach Delhi.

From the above situation, one can easily analyze that Creating a systematic Approach is more optimal and time and cost-effective as compared to a normal approach. This will same occur while designing Software. So in Software Engineering working on an Engineering or a Systematic approach is more beneficial.

1.20.1 Introduction

Software engineering has evolved over time from being considered an art to becoming a recognized engineering discipline. In the early days of computing, software development was primarily done by individuals or small teams who wrote code based on their own experiences and knowledge. This approach was often referred to as "hacking" or "programming by intuition." As the field of computing grew, it became apparent that this approach was not sustainable and that a more structured and systematic approach was needed.

In the 1960s and 1970s, the field of software engineering began to take shape. Researchers and practitioners began to develop formal methods for software design and development, such as structured programming and the use of flowcharts to represent algorithms. In 1968, a conference on software engineering was held, and the term "software engineering" was officially coined.

In the following decades, software engineering continued to evolve and mature. The introduction of object-oriented programming in the 1980s led to a shift in how software was designed and developed. The 1990s saw the emergence of the Agile software development methodologies, which emphasized flexibility and responsiveness to change. Today, software engineering is a well-established field with its own set of best practices, methodologies, and tools. It is considered as a discipline of engineering and follows the same principles like any other engineering field.

In summary, software engineering has evolved from an art to a discipline with its own set of best practices, methodologies, and tools. The field has grown and matured over time, with new technologies and approaches being developed to improve the design and development of software.

1.20.2 Is Software Engineering A Science Or An Art?

Most people think that writing a good quality program is an Art. So let's discuss it as an Art or Science. Like other engineering, branches suppose Mechanical engineering. It is based on Science where there are specific rules and names for each component, technique, and working principle related to it and before they are standardized the experience is marked as their thumb rule and on the basis of it, the rules are standardized by various organizations. Similarly, in Software Engineering there is heavy use of knowledge which is gathered from

the experience of practitioners. Thus, various Organizations or Researchers's made systematically organized the experience of practitioners in the theoretical form. Before these are standardized, the experience act as Thumb Rule. Thus, like every other Engineering Discipline, Software Engineering is a Science that is transformed from an Art.

1.20.3 Impact of Software Engineering Evolution From An Art To Engineering Discipline

Software Engineering principles have evolved over the last sixty years with the contributions of various researchers and software professionals. From the beginning period Software Engineering acts as an Art after that with time, it transformed into a craft and finally to Engineering Discipline.

Initially, Programmers used an Ad Hoc programming style. Ad Hoc programming is an approachable solution in an unplanned or unorganized manner. In this type of Programming Style, no plan is created on how to create Structure and Steps to complete the programming task but without having any systematic approach the problem needs to be solved in the required time. This style is now referred to as exploratory, build and fix, and code and fix styles.

Like in today's world various researchers and scientists working on those things which are not even necessary but during the initial period, Programmers worked on those things which are really needed. But as time fleet the Ad Hoc Programming will cause various problems which results in less efficiency and another approach i.e. systematic approach is adopted.

Let us get started with the detailed step-by-step process of the evolution of software engineering. The following steps are discussed:

1. Software engineering as an art
2. Software engineering transition from art to craft
3. Software engineering transition from craft to an engineering discipline

1. Software Engineering As An Art

Software Engineering as an Art means, this can be only learned by specific people and other people's are not allowed to work on them.

- Software engineering as an art refers to the creative and intuitive aspect of software development. It involves the use of creativity, problem-solving skills, and artistic expression to design and develop software systems that are not only functional but also aesthetically pleasing and user-friendly.
- In the early days of software development, many programmers considered software development as an art, as it was more about creativity and intuition than about following a set of established processes and methodologies. Programmers would often write code in an ad-hoc manner, without much structure or planning, and the final product was often the result of their personal artistic expression.
- However, as software systems have become more complex and important, it has become clear that the traditional ad-hoc approach to software development is not sufficient. This has led to the emergence of the field of software engineering, which has focused on bringing more structure and discipline to the software development process.

- Despite this, the art of software engineering still plays an important role in the development process. For example, a good software engineer must also have a sense of aesthetics, be able to think creatively and out of the box and be able to develop software that is easy to use and understand.

In conclusion, while software engineering has evolved into a more formal and structured discipline, the art of software development still plays an important role in the design and development of software systems. It's the balance of both that leads to successful software development.

For Example: In ancient times only a few people know Gold Ornaments making. Those who know Gold Ornaments making kept it as a secret and will reveal to his genetic generation. So at this time, this is known as Art and during that time accuracy was very less.

The same in Software Development, only a few people know about Software designing and coding and at that time there is no set of rules or instruction for software designing. And those who are able to write code efficiently and essentially fix bugs are considered proficient and those who are not good at programming and didn't know about that were left wondering that how they could effortlessly write elegant and correct code each time.

Like Gold Ornaments are not efficiently designed in the same way Program Coding is not efficiently done due to this various issues cause which degrades the efficiency of Software but at time efficiency is not considered as an important aspect.

2. Software Engineering Transition from Art to Craft

Software Engineering transformed from Art to Craft when the area of people who know Software Designing and Coding will increase.

The transition of software engineering from an art to a craft can be seen as a gradual process that has occurred over time.

- In the early days of software development, software engineering was primarily considered an art form. Programmers would often write code in an ad-hoc manner, without much structure or planning, and the final product was often the result of their personal artistic expression.
- As software systems have become more complex and important, it has become clear that the traditional ad-hoc approach to software development is not sufficient. This led to the emergence of the field of software engineering, which focused on bringing more structure and discipline to the software development process.
- With the emergence of software engineering as a discipline, the focus shifted from the artistic expression of individual programmers to the use of established processes and methodologies to ensure the quality and reliability of software systems. This led to the development of new methodologies and techniques such as the Waterfall and Agile methodologies, as well as the development of new tools and technologies to support software development.
- As a result, software engineering has transitioned from an art form to a craft. Instead of relying on the creativity and intuition of individual programmers, software development has become more structured and disciplined, with a focus on using established processes and methodologies to ensure the quality and reliability of software systems.

In summary, software engineering transitioned from being considered an art form to being considered a craft as the field grew more structured, formalized and focused on established processes, methodologies and tools. The goal of software engineering is not only to deliver a functional product but also to deliver a product that is reliable, maintainable and efficient.

For Example: When the secret of Gold Ornaments making is revealed to the only generation after generation which will increase the number of people in that Art and will convert art into a craft where accuracy is increased.

The same in Software Engineering, the Specific Degree and P.H.D were introduced in the Universities and some specific people will go further to learn and there is a certain rule which must be fulfilled to take that degree. So the number of people which increases in that area, and they start researching about it and create Standards and Styles for Coding and Software Development which must be followed and these Standards will help to create a less error-free software.

Like gold, ornaments are taking decent shapes and become efficient. In the same way, Program Writing has also become efficient in terms of Code reusability and efficiency but still, it is at times, efficiency is not considered an important aspect.

3. Software Engineering Transition From Craft To Engineering Discipline

The transition of software engineering from a craft to an engineering discipline can be seen as a gradual process that has occurred over time as the field has matured.

- In the early days of software development, software engineering was considered a craft, with a focus on using established processes and methodologies to ensure the quality and reliability of software systems. However, as the field has grown and matured, software engineering has become more formalized and structured, with the development of new methodologies, techniques and tools to support software development.
- One of the key milestones in the transition of software engineering from a craft to an engineering discipline was the publication of the book "Software Engineering: A Practitioner's Approach" by Pressman in 1975. This book provided a comprehensive overview of the field and defined software engineering as an engineering discipline.
- With the growing recognition of software engineering as an engineering discipline, the focus has shifted from the use of established processes and methodologies to ensure the quality and reliability of software systems, to a more scientific and systematic approach to software development.
- This has led to the development of new techniques and methodologies such as formal methods, model-driven development, and software architecture, which are based on sound scientific principles and are designed to ensure the quality and reliability of software systems.
- In conclusion, software engineering has transitioned from being considered a craft to being considered an engineering discipline as the field has matured and grown more formalized and structured. The field has shifted its focus from relying on established processes and method

In today's world Software Engineering acts as an Engineering Discipline when everyone can learn Software designing and coding irrespective of that they are pursuing a degree or not.

For example: In today's world everyone can learn Gold Ornament making and accuracy has much more increased with the help of various machines. Also, it is converted into a Professional study where people will learn about how? Why? The science behind it etc.

The same in software engineering occur where everyone can learn about Software Development with or without pursuing any Professional Studies and accuracy of Software Development increased with the help of Standards and improved rules created by researchers.

Software engineering principles are now widely used in industry and new principles are still continuing to emerge at a very rapid rate.

4. Benefits

There are many benefits to treating software development as an engineering discipline. Some of the key benefits include:

1. **Improved quality:** By following established best practices and methodologies, software engineers are able to produce higher quality software that is more reliable and less prone to errors.
2. **Increased productivity:** Formal methods and tools can help software engineers work more efficiently and effectively, leading to increased productivity.
3. **Greater predictability:** By following a structured and systematic approach, software engineers can make more accurate predictions about the time and resources required to complete a project.
4. **Better communication:** Software engineering practices can help ensure that all stakeholders, including developers, managers, and clients, have a clear understanding of the project's goals and requirements.
5. **Greater maintainability:** Software engineering practices can help ensure that the software is designed in a way that makes it easy to maintain and update over time.
6. **Better cost management:** By following established best practices and methodologies, software engineers can reduce the cost of development, testing and maintenance of software.
7. **Better Scalability:** Software engineering provides the process and methodologies to design the software in a way that it is easy to scale up or down as per the requirement.

In summary, treating software development as an engineering discipline can lead to higher quality software that is more reliable, predictable, and maintainable, as well as increased productivity and better cost management.

5. Important Points

1. Software engineering has evolved from being considered an art form to being considered an engineering discipline.
2. The transition from art to craft involved bringing more structure and discipline to the software development process.
3. The transition from craft to engineering discipline involved a more formalized and structured approach, with a focus on using established
4. processes and methodologies to ensure the quality and reliability of software systems.
5. Key milestones in this transition include the publication of the book "Software Engineering: A Practitioner's Approach" and the development of new methodologies, techniques, and tools to support software development.

6. The goal of software engineering is to deliver a functional, reliable, maintainable, and efficient software.

1.21 PROGRAM Vs. SOFTWARE

Every computer system requires instructions in order to do something. Program and software are such instructions given to the computer to perform some function. A program is a small block of code that instructs the system to do its task where as a software is a set of programs which instructs computer just like program does. But the functionalities and features of a software are more compared to that of a program.

1.21.1 What is a Program?

A program is a set of instructions that are given to the computer to execute. It allows the computer to perform a specific task. Each and every program has a unique function to do. These programs are stored in the memory and are run when it is necessary.

Every program is written in programming languages such as C, C++, Java etc., these programs are run using a compiler. Compilers then convert the source code into an object code (which is in 0s and 1s) and generates the output. Programs are small and they don't have any user interface.

A program is created by a single user and even a single line of code can be a program. It is the basic unit of a software.

1.21.2 What is a Software?

A software is a set of programs or instructions given to a computer system. It is a collection of programs that performs a function. It controls the working of a computer and manages all the components. Software is mainly classified into two types. They are Application software and System software.

1.21.3 Application Software

An application software is a software program than can perform a specific task. This task performed can be business related or personal. This software is accessed by the user and its sole purpose is to serve the user requirements. This software is written in high level programming languages.

Application software are installed by the user. User can use different application software based on their need. One can install a number of application software in their system. We can run multiple application software at the same time.

Games like candy crush, web browsers, messengers like WhatsApp, social media apps ae all examples of application software. Users directly interact with this software to perform their task.

1.21.4 System Software

System software is the main software of a computer that has a direct access to the system's hardware. It takes care and monitors all the other functions of a computer. System software executes the application software. It acts as a layer between the application software and the system's hardware. It is written in low level programming languages.

Example : Operating system, Firmware, Boot loaders, Device drivers etc.,

Difference between Program and Software

The following table highlights the major differences between Program and Software.

Program	Software
A program is a set of instructions to perform a specific task	A software is a collection is programs or instruc-tions to execute a task
Program can be a single line of code	Software consists of a number of lines of code
Program can be written by a single user	Software is written by many users
A single program can't be a software	Software can be a program
A program is generally written in low level programming languages	Software generally use high level programming languages
It is not classified further	Software is further classified into application and system software
It is small in size and ranges from kilobytes to megabytes	It is huge and its size ranges from megabytes to gigabytes
It doesn't have any user interface	It provides command line and graphical user interface
It is compiled every time when output is needed	It is already compiled and debugged during its development
As a program is small and perform only a specific task, it has limited features	It has a wide range of features and function-alities as it is a collection of many programs
It is simple and easy to create a program.	Creating a software is a complex task
It takes less time for program creation	More time is needed to create a software
It takes low cost to develop a program	Development of a software is costly
It requires only basic knowledge to build a program	User must be experienced and well trained in order to create a software
It depends on compiler for its execution	It depends on OS for its execution
There is no need of planning to develop a program	Software development requires well planning, organizing and time management
It contains only comments	It contains both comments and documentation
A program requires software in order to run it	Software can run independent of the programs
A computer can work without a program	Computer is useless without a software in it as it requires an OS to function which is system software
Example: Web browsers, malware etc.,	Example: Windows, Android, Adobe reader, MS Office etc., are examples of software

A program is a piece of code or a set of instructions that performs a given task whereas a software is a set of programs used to perform a task, which is more complex than that of a program. Software is more complex and is more essential than that of a program. A computer doesn't work without a software in it.

POINTS TO REMEMBER

1. Software has become a key element in the evolution of computer-based systems. Over the past five decades it has evolved from a specific problem solving and analysis tool to a complete industry in itself. However, some initial stage practices have lead to certain problems today.
2. Software comprises of programs, data and documents.
3. The purpose of software engineering is to provide a framework to develop high quality software.
4. Software engineering is an engineering discipline, which is concerned with all aspects of software production.
5. Software products consist of developed programs and associated documentation. Essential product attributes are maintainability, dependability, efficiency and usability.
6. The software process consists of activities, which are involved, in developing software products. Basic activities are software specification, development, validation and evolution.
7. Methods are organized ways of producing software.
8. They include suggestions for the process to be followed, the notations to be used, and rules governing the system descriptions, which are produced and design guidelines

EXERCISE

1. What is role of Software?
2. Why we need software engineering?
3. What is software engineering? Explain in brief.
4. What are the different types of software?
5. What is the prime objective of software engineering?
6. Define software engineering paradigm.
7. What are the different definitions of Software engineering ?
8. What are the attributes and principles of software engineering?
9. Define the advantages and disadvantages of software engineering.
10. What are different characteristics of software engineering ?
11. Define the components of software engineering.
12. What is the relationship of software engineering with other Discipline.
13. What are the Software Myths ?
14. What do you mean by Software Crisis?
15. Discuss the steps of software evolution.
16. Software Engineering is a art or a Science, Discuss
17. What is the difference between Program and software ?
18. The software differs from hardware as it is more logical in nature and hence, the difference in characteristics." Discuss.
19. Software is easy to change. It is myth? Explain why or why not? Explain with example.
20. What do you think the biggest problem is with software development?