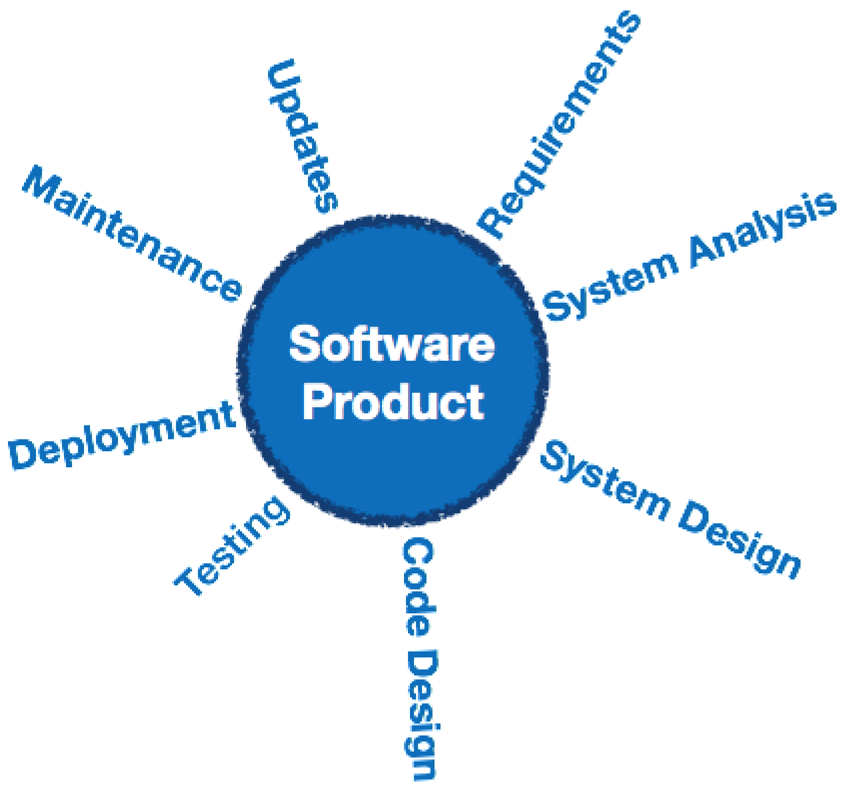
Unit – 1
[Introduction to Software Engineering]

1.Software Engineering :

The term is made of two words, software and 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.

2.Software Engineering Body of Knowledge

- The ***Software Engineering Body of Knowledge*** (**SWEBOK**) is an international standard ISO/IEC TR 19759:2005[1] specifying a guide to the generally accepted Software Engineering Body of Knowledge.
- The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) has been created through cooperation among several professional bodies and members of industry and is published by the IEEE Computer Society (IEEE).
- The standard can be accessed freely from the IEEE Computer Society.[3] In late 2013, SWEBOK V3 was approved for publication and released.[4] In 2016, the IEEE Computer Society kicked off the SWEBoK Evolution effort to develop future iterations of the body of knowledge

3.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.
And yet, **the same questions asked of the lone programmer are being asked when modern computer-based systems are built**:
1)Why does it take so long to get software finished?

2)Why are development costs so high?
3)Why can't we find all the errors before we give the software to customers?
4)Why do we continue to have difficulty in measuring progress as software is being developed?

4.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. The 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.

5.Software myths:

Software Myths : What is software myth in software engineering.

- 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. **Software myths propagate false beliefs and confusion in the minds of management, users and developers.**

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

Management Myths

The members of an organization can acquire all-the information, they require from a manual, which contains standards, procedures, and principles;	Standards are 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 programmerscan 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 controlling the software project internally. The organization invariably suffers when it out sources the software project.

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.

Table User Myths

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. Instead, a complete and formal description of requirements is essential before starting development. Adding requirements at a later stage often requires 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 and extra resources.

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.

Table Developer Myths

Software development is considered complete when the code is delivered.	50% to 70% of all the efforts 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 playa crucial role.
Software engineering requires unnecessary 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

6.The software problem: Cost, schedule and quality, Scale and change:

[Triple constraints of Project management: Cost, schedule and quality]

In the industrial-strength software domain, there are three basic forces at play—cost, schedule, and quality. The software should be produced at reasonable cost, in a reasonable time, and should be of good quality. These three parameters often drive and define a software project.

Cost :

- Industrial-strength software is very expensive primarily due to the fact that software development is extremely labor-intensive. To get an idea of the costs involved, let us consider the current state of practice in the industry. Lines of code (LOC) or thousands of lines of code (KLOC) delivered is by far the most commonly used measure of software size in the industry. As the main cost of producing software is the manpower employed, the cost of developing software is generally measured in terms of person-months of effort spent in development. And productivity is frequently measured in the industry in terms of LOC (or KLOC) per person-month.

  The productivity in the software industry for writing fresh code generally ranges from few hundred to about 1000+ LOC per person-month. This productivity is over the entire development cycle, not just the coding task. Software companies often charge the client for whom they are developing the software between $3000 - $15,000 per person-month. With a productivity of 1000 LOC per person-month, it means that each line of delivered code costs between $3 and $15! And even small projects can easily end up with software of 50,000 LOC. With this productivity, such a software project will cost between $150,000 and $750,000!

Schedule :

- Schedule is another important factor in many projects. Business trends are dictating that the time to market of a product should be reduced; that is, the cycle time from concept to delivery should be small. For software this means that it needs to be developed faster, and within the specified time. Unfortunately, the history of software is full of cases where projects have been substantially late.

- Clearly, therefore, reducing the cost and the cycle time for software development are central goals of software engineering. Productivity in terms of output (KLOC) per person-month can adequately capture both cost and schedule concerns. If productivity is higher, it should be clear that the cost in terms of person-months will be lower (the same work can now be done with fewer person-months). Similarly, if productivity is higher, the potential of developing the software in less time improves—a team of higher productivity will finish a job in less time than a same-size team with lower productivity. (The actual time the project will take, of course, depends also on the number of people allocated to the project.) Hence, pursuit of higher productivity is a basic driving force behind software engineering and a major reason for using the different tools and techniques.

Quality :

- Besides cost and schedule, the other major factor driving software engineering is quality. Today, quality is one of the main mantras, and business strategies are designed around it. Unfortunately, a large number of instances have occurred regarding the unreliability of software—the software often does not do what it is supposed to do or does something it is not supposed to do. Clearly, developing high-quality software is another fundamental goal of software engineering. However, while cost is generally well understood, the concept of quality in the context of software needs further elaboration. The international standard on software product quality [55] suggests that software quality comprises six main attributes, as shown in Figure 1.1.

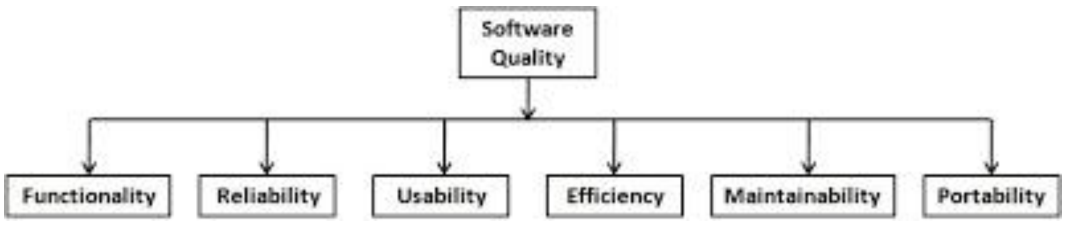


Figure 1.1: Software quality attributes.

These attributes can be defined as follows:

- **Functionality**. The capability to provide functions which meet stated and implied needs when the software is used.
- **Reliability**. The capability to provide failure-free service.
- **Usability**. The capability to be understood, learned, and used.
- **Efficiency**. The capability to provide appropriate performance relative to the amount of resources used.
- **Maintainability**. The capability to be modified for purposes of making corrections, improvements, or adaptation.
- **Portability**. The capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product.

Scale and Change :

Though cost, schedule, and quality are the main driving forces for a project in our problem domain (of industry strength software), there are some other characteristics of the problem domain that also influence the solution approaches employed. We focus on two such characteristics—scale and change.

Scale :

- Most industrial-strength software systems tend to be large and complex, requiring tens of thousands of lines of code. Sizes of some of the well-known software products are given in An example will illustrate this point. Consider the problem of counting people in a room versus taking a census of a country. Both are essentially counting problems. But the methods used for counting people in a room will just not work when taking a census. A different set of methods will have to be used for conducting a census, and the census problem will require considerably more management, organization, and validation, in addition to counting.

- Similarly, methods that one can use to develop programs of a few hundred lines cannot be expected to work when software of a few hundred thousand lines needs to be developed. A different set of methods must be used for developing large software.

- Any software project involves the use of engineering and project management. In small projects, informal methods for development and management can be used. However, for large projects, both have to be much more rigorous, as illustrated in Figure 1.2. In other words, to successfully execute a project, a proper method for engineering the system has to be employed and the project has to be tightly managed to make sure that cost, schedule, and quality are under control. Large scale is a key characteristic of the problem domain and the solution approaches should employ tools and techniques that have the ability to build large software systems

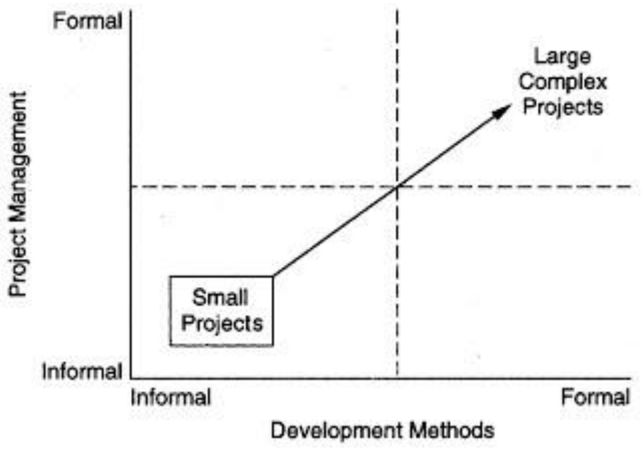


Figure 1.2: The problem of scale.

Change :

- Change is another characteristic of the problem domain which the approaches for development must handle. As the complete set of requirements for the system is generally not known (often cannot be known at the start of the project) or stated, as development proceeds and time passes, additional requirements are identified, which need to be incorporated in the software being developed. This need for changes requires that methods for development embrace change and accommodate it efficiently. Change requests can be quite disruptive to a project, and if not handled properly, can consume up to 30 to 40% of the development cost [14].

- As discussed above, software has to be changed even after it has been deployed. Though traditionally changes in software during maintenance have been distinguished from changes that occur while the development is taking place, these lines are blurring, as fundamentally the changes in both of these scenarios are similar—existing source code needs to be changed due to some changes in the requirements or due to some defects that need to be removed.

- Overall, as the world changes faster, software has to change faster, even while under development. Changes in requirements are therefore a characteristic of the problem domain. In today's world, approaches that cannot accept and accommodate change are of little use—they can solve only those few problems that are change resistant.

7. Principles of Software Engineering :

Seven principles have been determined which form a reasonably independent and complete set. These are:

(1) Manage using a phased life-cycle plan.

(2) Perform continuous validation.

(3) Maintain disciplined product control.

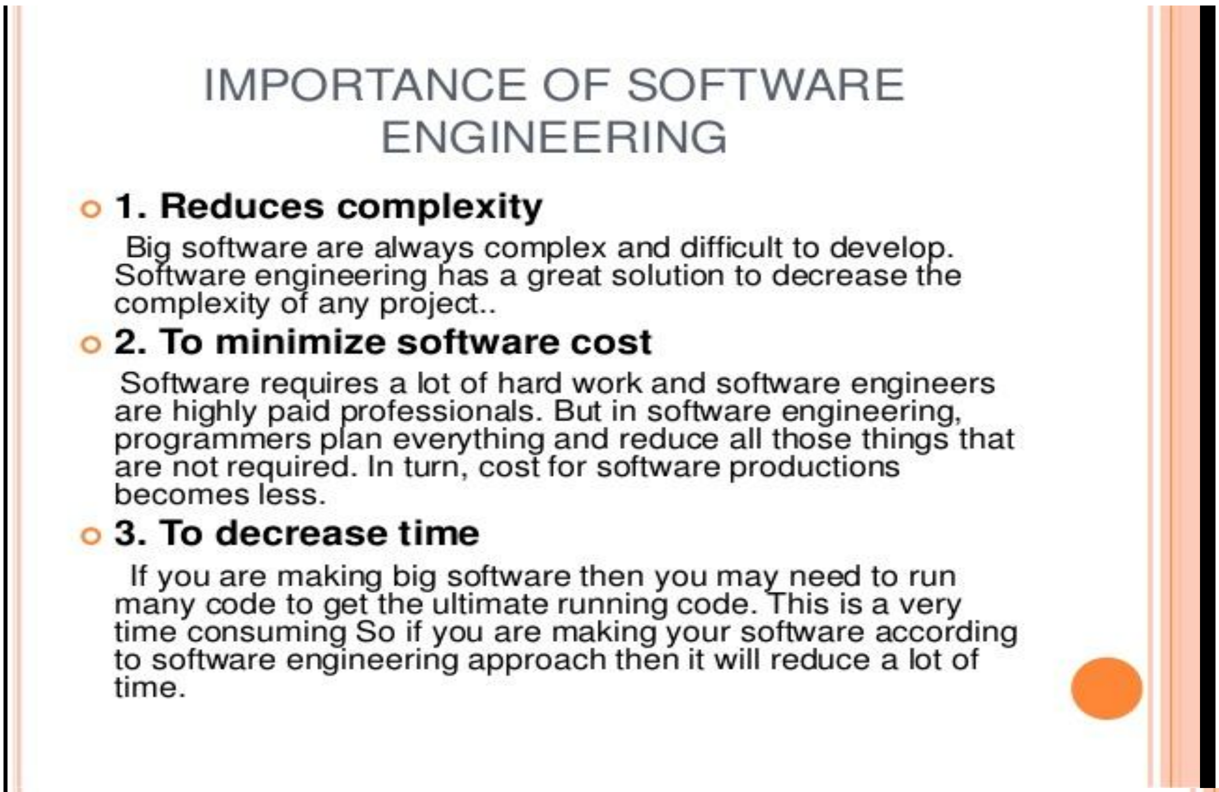
(4) Use modern programming practices.

(5) Maintain clear accountability for results.

(6) Use better and fewer people.

(7) Maintain a commitment to improve the process.

8.



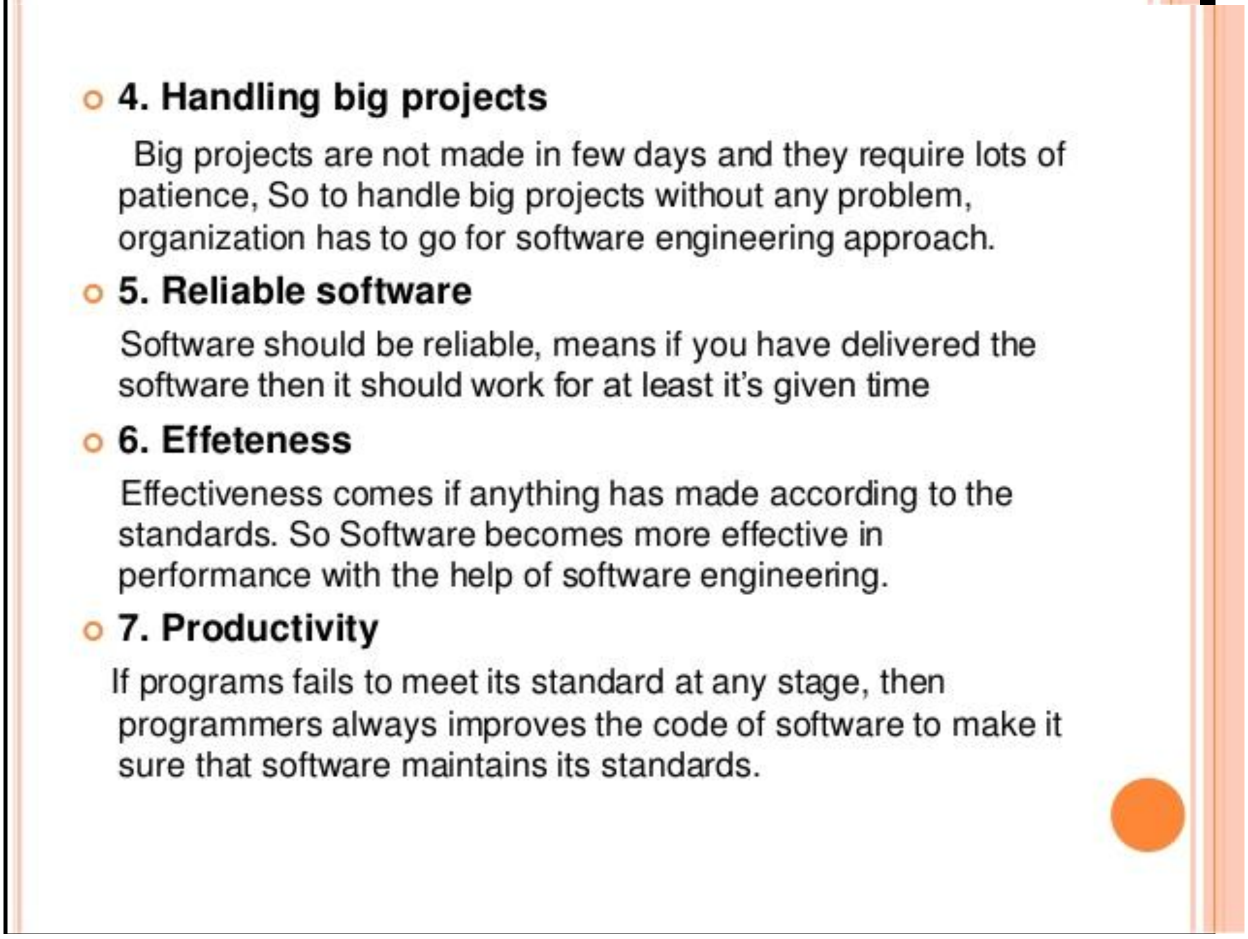
IMPORTANCE OF SOFTWARE ENGINEERING

- **1. Reduces complexity**

  Big software are always complex and difficult to develop. Software engineering has a great solution to decrease the complexity of any project..
- **2. To minimize software cost**

  Software requires a lot of hard work and software engineers are highly paid professionals. But in software engineering, programmers plan everything and reduce all those things that are not required. In turn, cost for software productions becomes less.
- **3. To decrease time**

  If you are making big software then you may need to run many code to get the ultimate running code. This is a very time consuming So if you are making your software according to software engineering approach then it will reduce a lot of time.



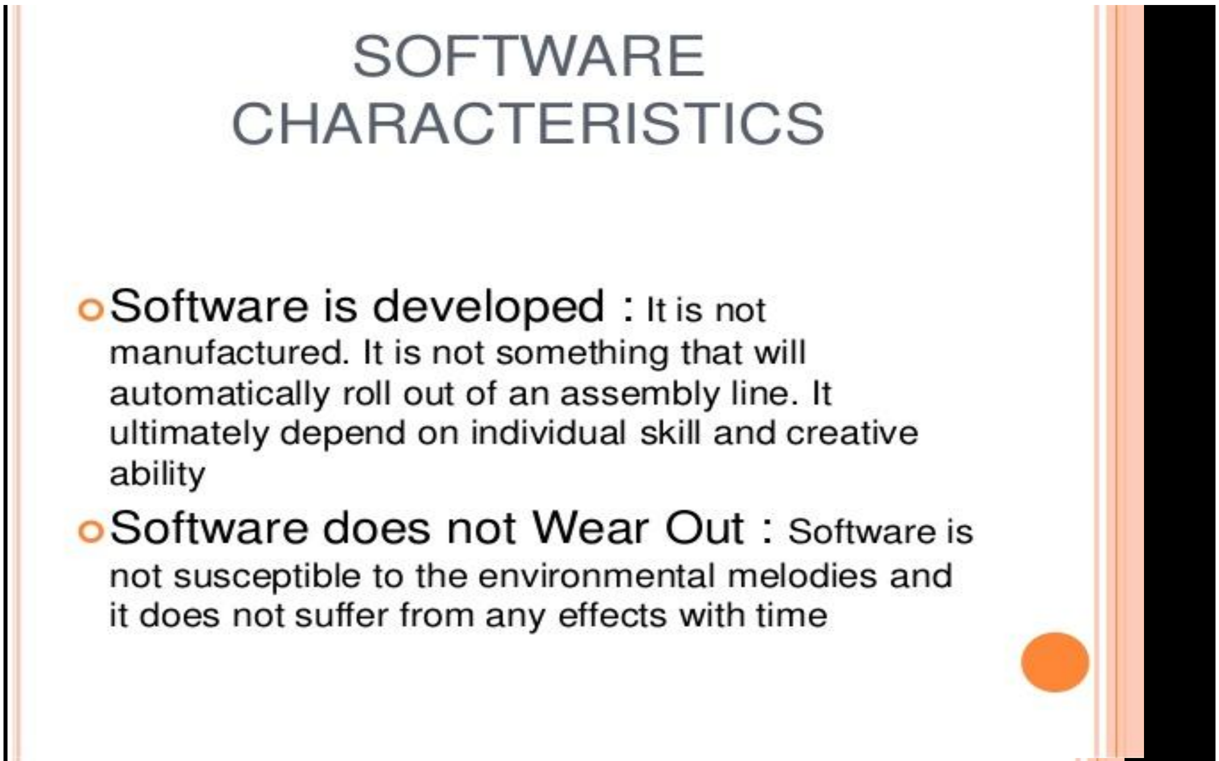
- **4. Handling big projects**

  Big projects are not made in few days and they require lots of patience, So to handle big projects without any problem, organization has to go for software engineering approach.
- **5. Reliable software**

  Software should be reliable, means if you have delivered the software then it should work for at least it's given time
- **6. Effeteness**

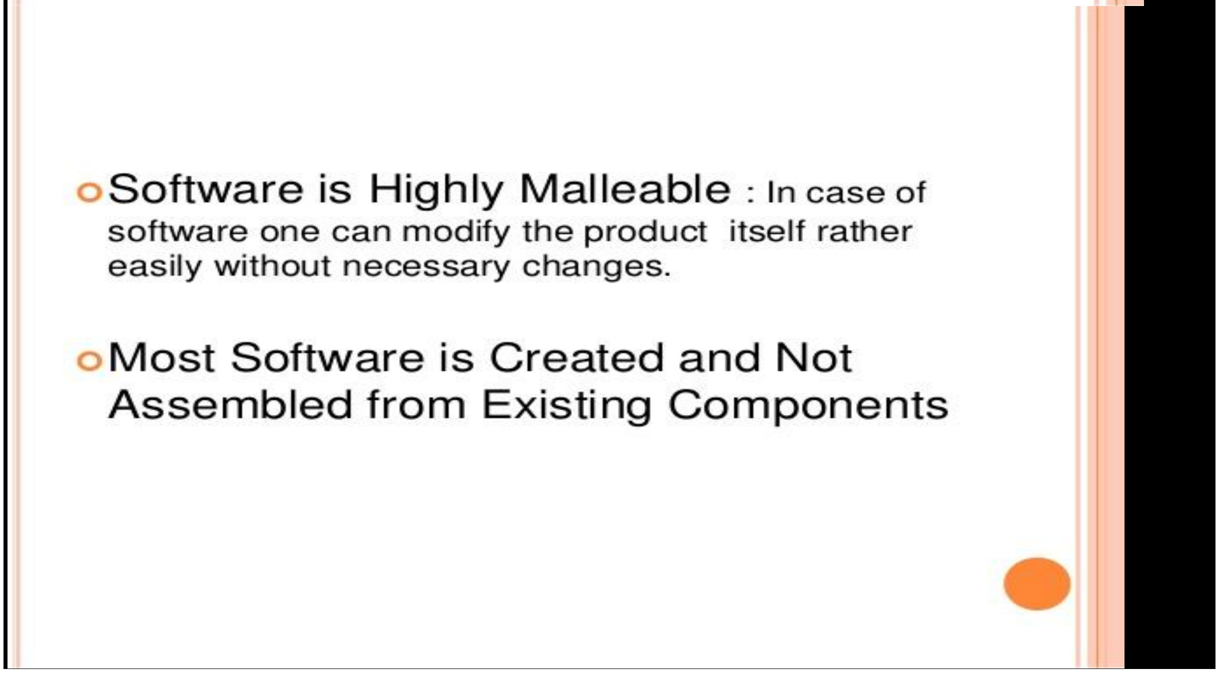
  Effectiveness comes if anything has made according to the standards. So Software becomes more effective in performance with the help of software engineering.
- **7. Productivity**

  If programs fails to meet its standard at any stage, then programmers always improves the code of software to make it sure that software maintains its standards.

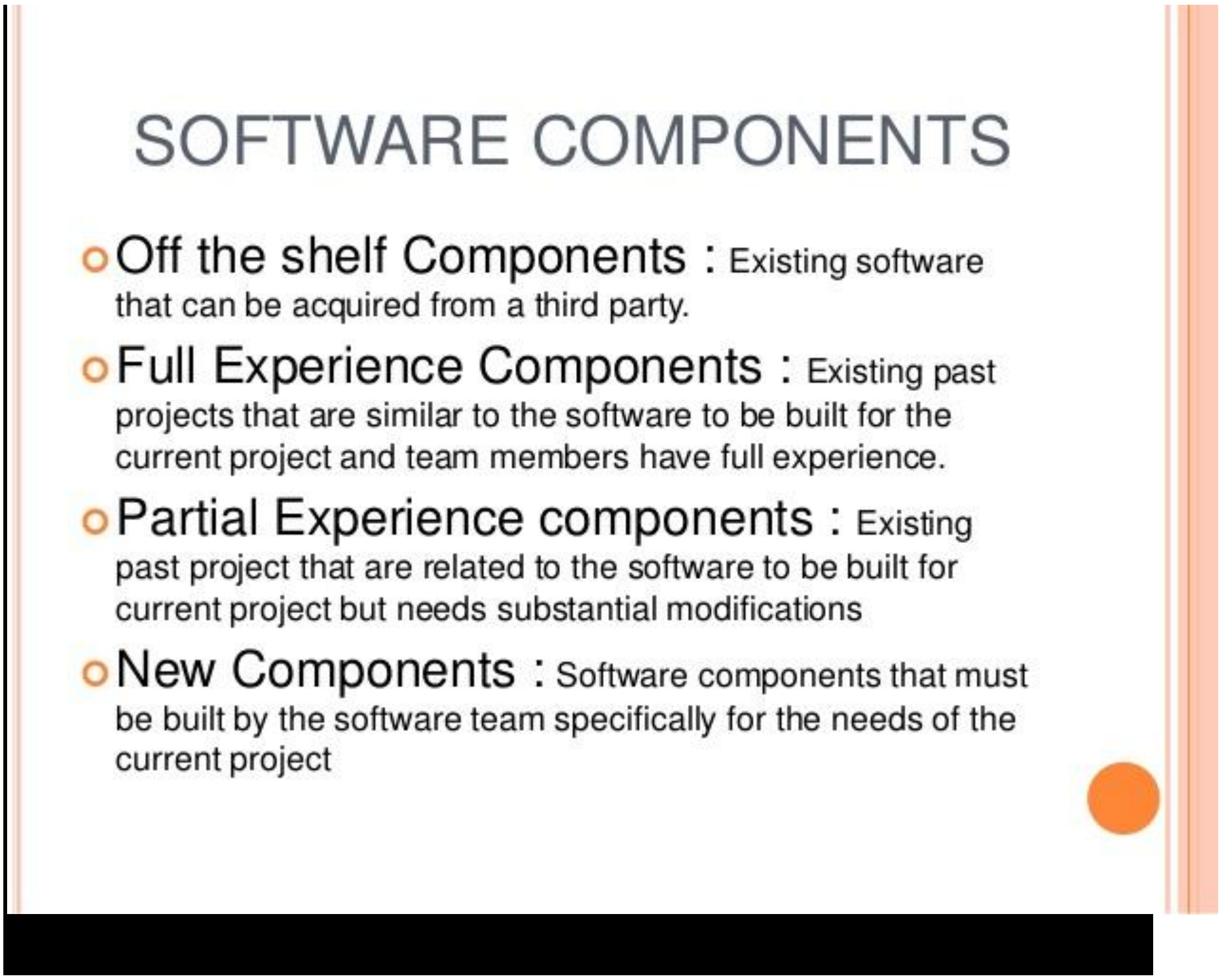


SOFTWARE CHARACTERISTICS

- **Software is developed :** It is not manufactured. It is not something that will automatically roll out of an assembly line. It ultimately depend on individual skill and creative ability
- **Software does not Wear Out :** Software is not susceptible to the environmental melodies and it does not suffer from any effects with time



- **Software is Highly Malleable** : In case of software one can modify the product itself rather easily without necessary changes.
- **Most Software is Created and Not Assembled from Existing Components**



SOFTWARE COMPONENTS

- **Off the shelf Components :** Existing software that can be acquired from a third party.
- **Full Experience Components :** Existing past projects that are similar to the software to be built for the current project and team members have full experience.
- **Partial Experience components :** Existing past project that are related to the software to be built for current project but needs substantial modifications
- **New Components :** Software components that must be built by the software team specifically for the needs of the current project

9.Software Process

A software process (also knows as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

Any software process must include the following four activities:

1. **Software specification** (or requirements engineering): Define the main functionalities of the software and the constrains around them.
2. S**oftware design and implementation**: The software is to be designed and programmed.
3. **Software verification and validation**: The software must conforms to it's specification and meets the customer needs.
4. **Software evolution** (software maintenance): The software is being modified to meet customer and market requirements changes.

In practice, they include sub-activities such as requirements validation, architectural design, unit testing, …etc.

There are also **supporting activities** such as configuration and change management, quality assurance, project management, user experience.

10.Software Process Framework:

A process framework establishes the foundation for a complete *software process* by identifying a small *number of framework activities* that are applicable to all software projects, regardless of size or complexity. It also includes a set of *umbrella activities* that are applicable across the entire software process. Some most applicable framework activities are described below.

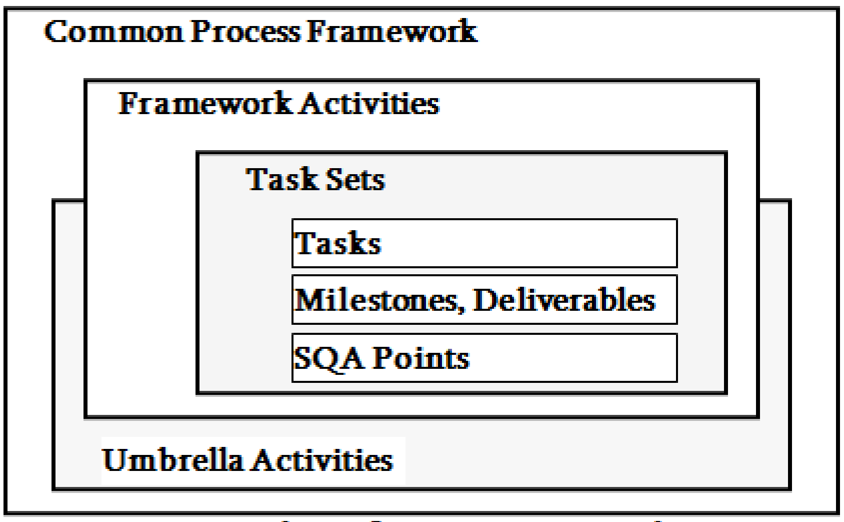


Figure: Chart of Process Framework

11.Elements of software process:

They are different elements of software process.

1. Communication:

This activity involves heavy communication with customers and other stakeholders in order to gather requirements and other related activities.

2. Planning:

Here a plan to be followed will be created which will describe the technical tasks to be conducted, risks, required resources, work schedule etc.

3. Modeling:

A model will be created to better understand the requirements and design to achieve these requirements.

4. Construction:

Here the code will be generated and tested.

5.Deployment:

Here, a complete or partially complete version of the software is represented to the customers to evaluate and they give feedbacks based on the evaluation.

12.Q) Is software engineering applicable when WebApps are built? If so, how might it be modified to accommodate the unique characteristics of **WebApps**?

Ans)Yes,**software engineering** is **applicable, when WebApps are built** because it is a layered technology and consists of Tools, Methods, Process, and A quality focus.

13.Hardware characteristics are completely different from software characteristics. Justify

Ans)Differences between Hardware and Software Development

- Software is easier to change than hardware. The cost of change is much higher for hardware than for software.
- Software products evolve through multiple releases by adding new features and re-writing existing logic to support the new features. Hardware products consist of physical components that cannot be "refactored" after manufacturing, and cannot add new capabilities that require hardware changes.
- Designs for new hardware is often based upon earlier-generation products, but commonly rely on next-generation components not yet present.
- Hardware designs are constrained by the need to incorporate standard parts.
- Specialized hardware components can have much longer lead times for acquisition than is true for software.
- Hardware design is driven by architectural decisions. More of the architectural work must be done up front compared to software products.
- The cost of development for software products is relatively flat over time. However, the cost of hardware development rises rapidly towards the end of the development cycle. Testing software commonly requires developing thousands of test cases. Hardware testing involves far fewer tests.

- Software testing is done by specialized Quality Assurance (QA) engineers, while hardware testing is commonly done by the engineers who are creating the product.
- Hardware must be designed and tested to work over a range of time and environmental conditions, which is not the case for software.
- Hardware development incorporates four parallel, synchronized projects:

UNIT 1

[QUESTIONS]

1. Explain about The evolving role of software.
2. How software changed from day to day.

   [or]

   Explain about the changing nature of software.

   [or]

   Explain about applications of software.[same answer for the any of the question asked]

3. What are the common software myths? Explain.
4. What are the main Software problems during its development? What are its major disadvantages?

   [The **software problem:** Cost, schedule and quality, Scale and Change. You can write about these when asked this question.]

5. What is software Engineering? Explain software Engineering Book of Knowledge.
6. Explain the Principles of Software Engineering.
7. Explain software process. What are elements of software process.
8. Hardware characteristics are completely different from software characteristics. Justify.
9. Sketch the common framework for software process and explain.
10. What are software components? Explain software Characteristics.
11. Explain the importance of software engineering.