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

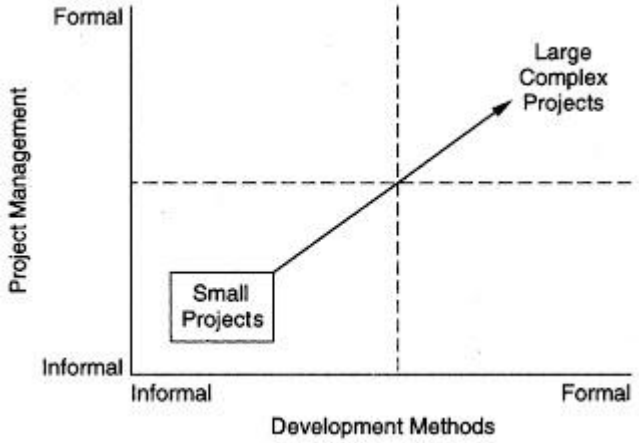


Figure 1.2: The problem of scale.

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.

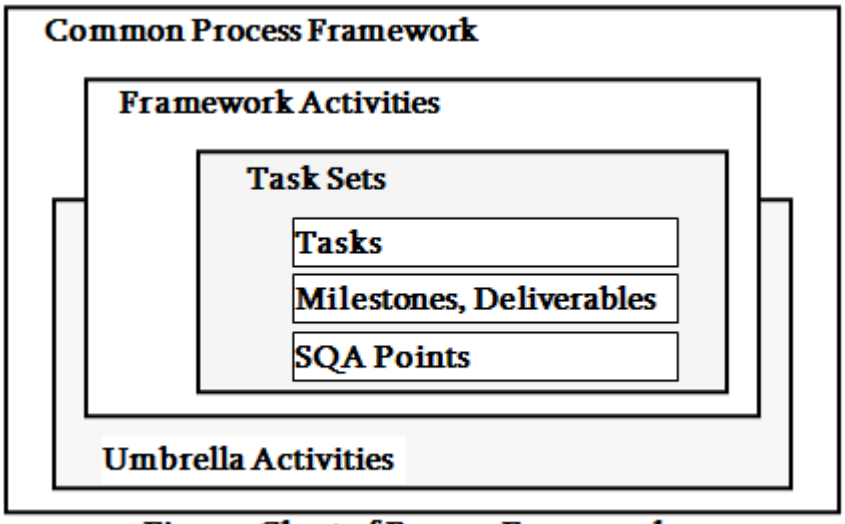


Figure: Chart of Process Framework

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.

Layered Technology

Software engineering is fully a layered technology, to develop software we need to go from one layer to another. All the layers are connected and each layer demands the fulfillment of the previous layer.

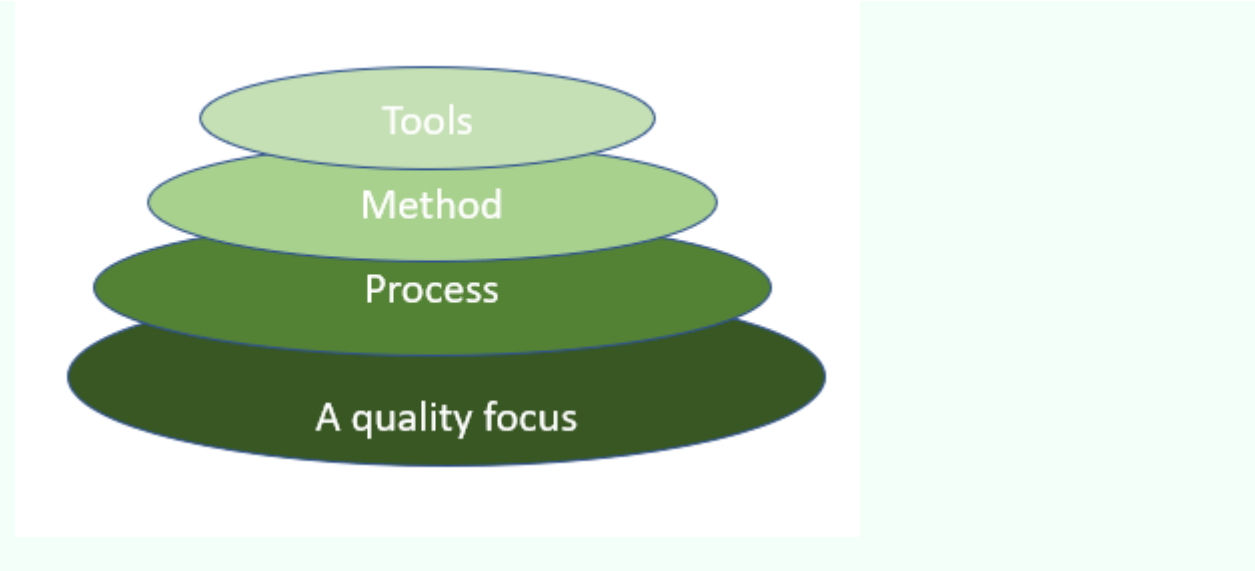
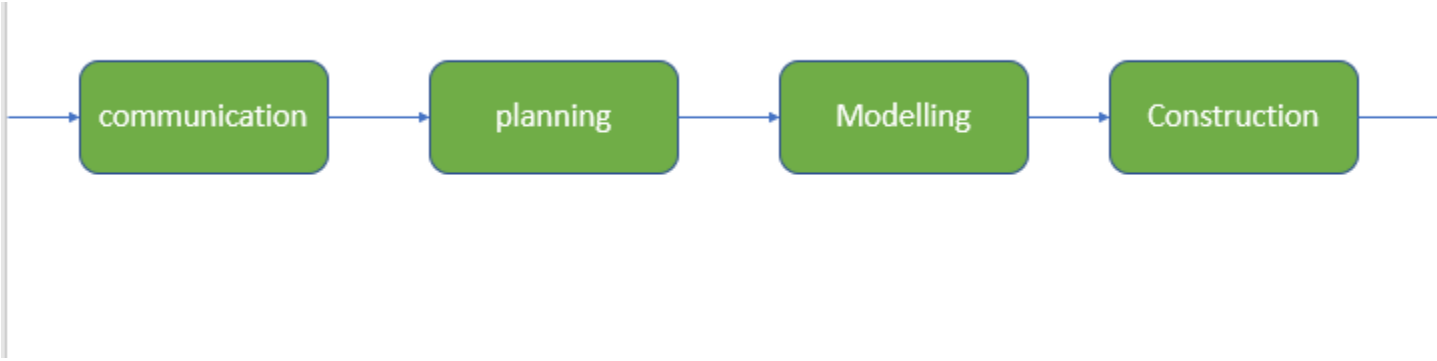


Fig: The diagram shows the layers of software development

Layered technology is divided into four parts:

1. A quality focus: It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.

2. Process: It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time. Process defines a framework that must be established for the effective delivery of software engineering technology. The software process covers all the activities, actions, and tasks required to be carried out for software development.



Process activities are listed below:-

- **Communication:** It is the first and foremost thing for the development of software. Communication is necessary to know the actual demand of the client.
- **Planning:** It basically means drawing a map for reduced the complication of development.
- **Modeling:** In this process, a model is created according to the client for better understanding.
- **Construction:** It includes the coding and testing of the problem.
- **Deployment:-** It includes the delivery of software to the client for evaluation and feedback.

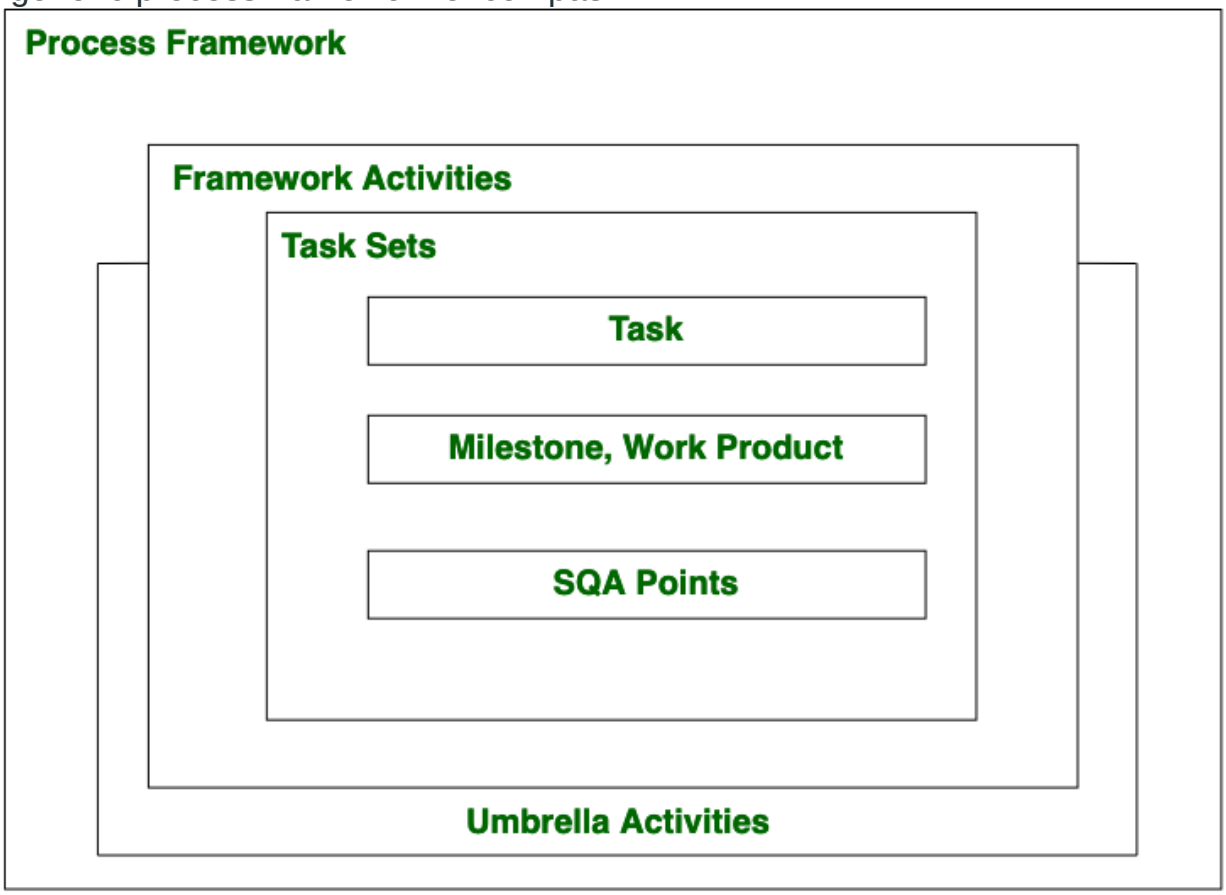
3. Method: During the process of software development the answers to all "how-to-do" questions are given by method. It has the information of all the tasks which includes communication, requirement analysis, design modeling, program construction, testing, and support.

4. Tools: Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

Software Process Framework

Framework is a Standard way to build and deploy applications. **Software Process Framework** is the foundation of complete software engineering process. Software process framework includes set of all umbrella activities. It also includes number of framework activities that are applicable to all software projects.

A generic process framework encompas



Software Process Framework

ses five activities which are given below one by one:

1. **Communication:**
   In this activity, heavy communication with customers and other stakeholders, as well as requirement gathering is done.

2. **Planning:**
   In this activity, we discuss the technical related tasks, work schedule, risks, required resources, etc.
3. **Modeling:**
   Modeling is about building representations of things in the ‘real world’. In modeling activity, a product’s model is created in order to better understand the requirements.
4. **Construction:**
   In software engineering, construction is the application of set of procedures that are needed to assemble the product. In this activity, we generate the code and test the product in order to make better product.
5. **Deployment:**
   In this activity, a complete or non-complete product or software is represented to the customers to evaluate and give feedback. On the basis of their feedback, we modify the product for supply of better product.

Umbrella activities include:

- Risk Management
- Software Quality Assurance (SQA)
- Software Configuration Management (SCM)
- Measurement
- Formal Technical Reviews (FTR)

Capability maturity model (CMM)

CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.

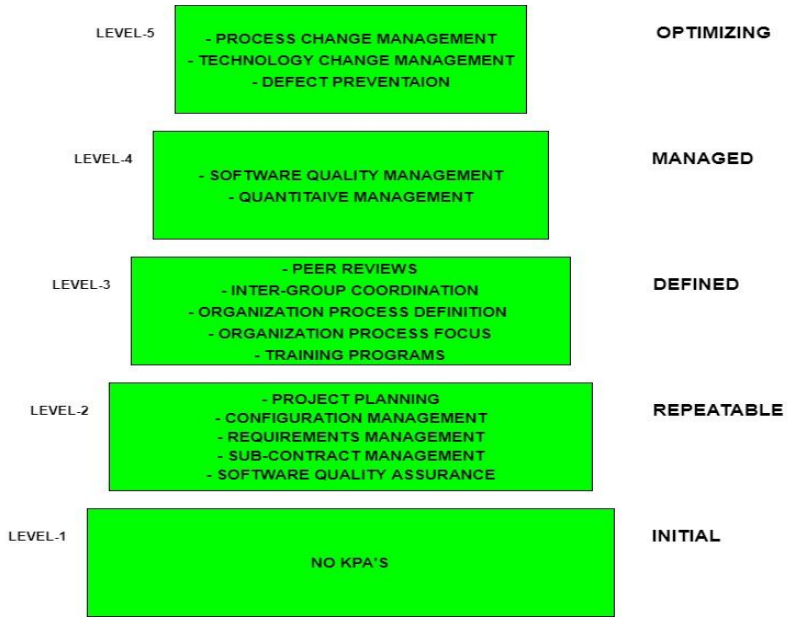
- It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to further enhance the maturity of the process used to develop those software products.
- It is based on profound feedback and development practices adopted by the most successful organizations worldwide.
- This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.
- Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

Shortcomings of SEI/CMM:

- It encourages the achievement of a higher maturity level in some cases by displacing the true mission, which is improving the process and overall software quality.
- It only helps if it is put into place early in the software development process.
- It has no formal theoretical basis and in fact is based on the experience of very knowledgeable people.
- It does not have good empirical support and this same empirical support could also be constructed to support other models.

Key Process Areas (KPA's):

Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity. Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, workv products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.



Level-1: Initial –

- No KPA's defined.
- Processes followed are Adhoc and immature and are not well defined.
- Unstable environment for software development.
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.
- **Project Planning-** It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for the successful completion of good quality software.
- **Configuration Management-** The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- **Requirements Management-** It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- **Subcontract Management-** It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
- **Software Quality Assurance-** It guarantees a good quality software product by following certain rules and quality standard guidelines while developing.

Level-3: Defined –

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well-defined integrated set of project-specific software engineering and management processes.
- **Peer Reviews-** In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- **Intergroup Coordination-** It consists of planned interactions between different development teams to ensure efficient and proper fulfillment of customer needs.
- **Organization Process Definition-** Its key focus is on the development and maintenance of the standard development processes.
- **Organization Process Focus-** It includes activities and practices that should be followed to improve the process capabilities of an organization.
- **Training Programs-** It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed –

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.
- **Software Quality Management-** It includes the establishment of plans and strategies to develop quantitative analysis and understanding of the product's quality.
- **Quantitative Management-** It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing –

- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques, and evaluation of software processes is done to prevent recurrence of known defects.
- **Process Change Management-** Its focus is on the continuous improvement of the organization's software processes to improve productivity, quality, and cycle time for the software product.
- **Technology Change Management-** It consists of the identification and use of new technologies to improve product quality and decrease product development time.
- **Defect Prevention-** It focuses on the identification of causes of defects and prevents them from recurring in future projects by improving project-defined processes.

Process Patterns

As the software team moves through the software process they encounter problems. It would be very useful if solutions to these problems were readily available so that problems can be resolved quickly. Process-related problems which are encountered during software engineering work, it identifies the encountered problem and in which environment it is found, then it will suggest proven solutions to problem, they all are described by process pattern. By solving problems a software team can construct a process that best meets needs of a project.

Uses of the process pattern :
At any level of abstraction, patterns can be defined. They can be used to describe a problem and solution associated with framework activity in some situations. While in other situations patterns can be used to describe a problem and solution associated with a complete process model.

- **Pattern Name –**
  Meaningful name must be given to a pattern within context of software process (e.g. Technical Reviews).
- **Forces –**
  The issues that make problem visible and may affect its solution also environment in which pattern is encountered.

Type :
It is of three types :

1. **Stage pattern –**
   Problems associated with a framework activity for process are described by stage pattern. Establishing Communication might be an example of a staged pattern. This pattern would incorporate task pattern Requirements Gathering and others.
2. **Task-pattern –**
   Problems associated with a software engineering action or work task and relevant to successful software engineering practice (e.g., Requirements Gathering is a task pattern) are defined by task-pattern.
3. **Phase pattern –**
   Even when the overall flow of activities is iterative in nature, it defines sequence of framework activities that occurs within process. Spiral Model or Prototyping might be an example of a phase pattern.

Initial Context :
Conditions under which the pattern applies are described by initial context. Prior to the initiation of the pattern :

1. What organizational or term-related activities have already occurred?
2. Entry state for the process?
3. Software engineering information or project information already exists?

For example, the Planning pattern requires that :

- Collaborative communication has been established between customers and software engineers.
- Successful completion of a number of task patterns for the communication pattern has occurred.
- The project constraints, basic requirements, and the project scope are known.

Problem :

Any specific problem is to be solved by pattern.

Solution –

Describes how to implement pattern successfully. This section describes how initial state of process is modified as a consequence of initiation of pattern.

Resulting Context :

Once the pattern has been successfully implemented, it describes conditions. Upon completion of pattern :

1. Organizational or term-related activities must have occurred?
2. What should be the exit state for the process?
3. What software engineering information has been developed?

Related pattern :

Provide a list of all process patterns that are directly related to this one. It can be represented n a hierarchy or in some other diagrammatic form.

Known uses and Examples –

In which the pattern is applicable, it indicates the specific instances. For example, communication is mandatory at the beginning of every software project, is recommended throughout the software project, and is mandatory once the deployment activity is underway.

Personal and Team Process Model

The best software process is personal and team process model one that is close to the people who will be doing the work. Watts Humphrey proposed two process models. Models “Personal Software Process (PSP)” and “Team Software Process (TSP).” Both require hard work, training, and coordination, but both are achievable.

Personal Software Process (PSP)

The Personal Software Process (PSP) emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product. In addition PSP makes the practitioner responsible for project planning and empowers

the practitioner to control the quality of all software work products that are developed. The PSP model defines five framework activities:

- ***Planning.*** This activity isolates requirements and develops both size and resource estimates. In addition, defects estimate (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.
- ***High level design.*** External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.
- ***High level design review.*** Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.
- ***Development.*** The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.
- ***Postmortem.*** Using the measures and metrics collected, the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

PSP stresses the need to identify errors early and, just as important, to understand the types of errors that you are likely to make. PSP represents a disciplined, metrics based approach to software engineering that may lead to culture shock for many practitioners.

Team Software Process (TSP)

Watts Humphrey extended the lessons learned from the introduction of PSP and proposed a Team Software Process (TSP). The goal of TSP is to build a "self directed" project team that organizes itself to produce high quality software.

Humphrey defines the following objectives for TSP:

- *Build self directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPTs) of 3 to about 20 engineers.*

- *Show managers how to coach and motivate their teams and how to help them sustain peak performance*
- *Accelerate software process improvement by making CMM23 Level 5 behavior normal and expected.*
- *Provide improvement guidance to high-maturity organizations.*
- *Facilitate university teaching of industrial-grade team skills.*

A self directed team has a consistent understanding of its overall goals and objectives; defines roles and responsibilities for each team member; tracks quantitative project data (about productivity and quality); identifies a team process that is appropriate for the project and a strategy for implementing the process; defines local standards that are applicable to the team's software engineering work; continually assesses risk and reacts to it; and tracks, manages, and reports project status.

TSP defines the following framework activities: project launch, high level design, implementation, personal and team process model, integration and test, and postmortem. TSP makes use of a wide variety of scripts, forms, and standards that serve to guide team members in their work. "Scripts" define specific process activities (project launch, design, implementation, integration and system testing, postmortem) and other more detailed work functions (development planning, requirements development, software configuration management, unit test) that are part of the team process.

Waterfall Model:

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The Waterfall model is the earliest SDLC approach that was used for software development.

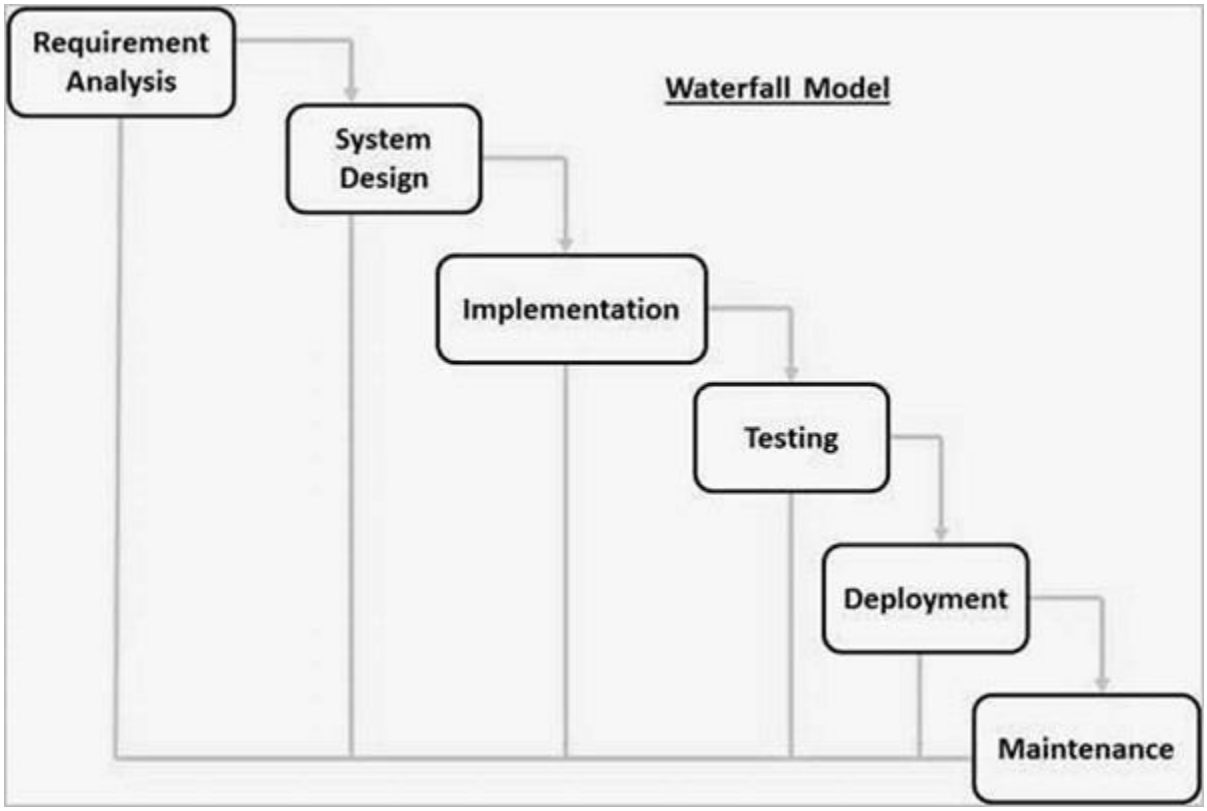
The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of

software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are −

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.

- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows −

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Incremental Model

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

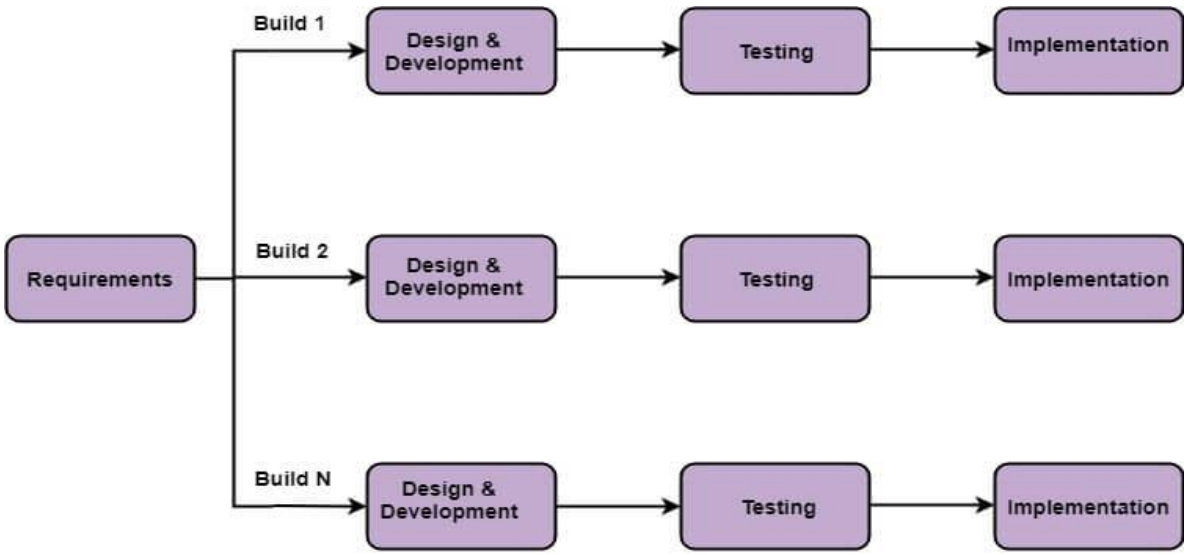


Fig: Incremental Model

The various phases of incremental model are as follows:

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

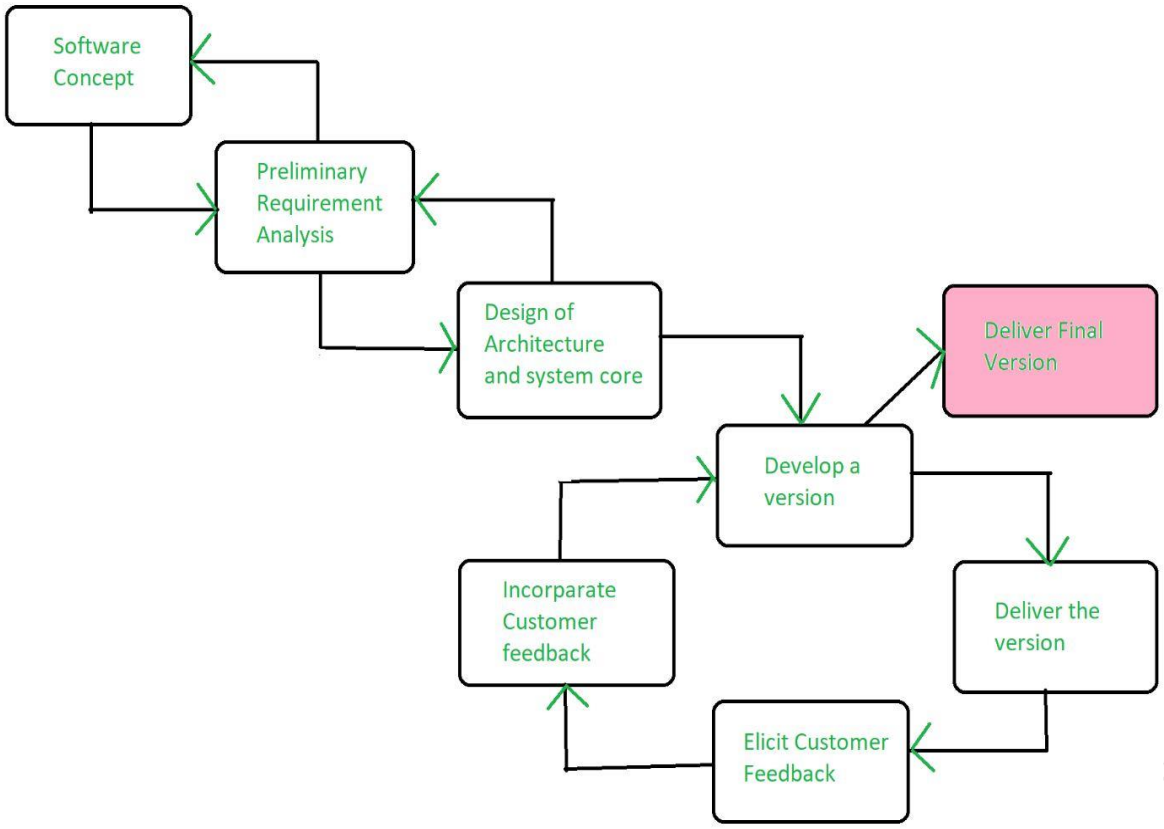
Disadvantage of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

Evolutionary Model

Evolutionary model is a combination of Iterative and Incremental model of software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done.
It is better for software products that have their feature sets redefined during development because of user feedback and other factors. The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.

Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan or process. Therefore, the software product evolves with time.
All the models have the disadvantage that the duration of time from start of the project to the delivery time of a solution is very high. Evolutionary model solves this problem in a different approach.



Evolutionary model suggests breaking down of work into smaller chunks, prioritizing them and then delivering those chunks to the customer one by one. The number of chunks is huge and is the number of deliveries made to the customer. The main advantage is that the customer's confidence increases as he constantly gets quantifiable goods or services from the beginning of the project to verify and validate his requirements. The model allows for changing requirements as well as all work in broken down into maintainable work chunks.

Application of Evolutionary Model:

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
2. Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Advantages:

- In evolutionary model, a user gets a chance to experiment partially developed system.
- It reduces the error because the core modules get tested thoroughly.

Disadvantages:

- Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

Unified Process ModelDefinition:

The **unified process model** (or **UPM**) is an iterative, incremental, architecture-centric, and use-case driven approach to software development. Let's first take a look at the use-case driven approach.

Use-Case Driven Approach

A **use-case** defines the interaction between two or more entities. The list of requirements specified by a customer are converted to functional requirements by a business analyst and generally referred to as use-cases. A use-case describes the operation of a software as interactions between the customer and the system, resulting in a specific output or a measurable return. For example, the online cake shop can be specified in terms of use cases such as 'add cake to cart,' 'change the quantity of added cakes in cart,' 'cake order checkout,' and so on. Each use case represents a significant functionality and could be considered for an iteration.

Architecture-Centric Approach

Now, let's take a closer look at the **architecture-centric approach**. Using this approach, you'd be creating a blueprint of the organization of the software system. It would include taking into account the different technologies, programming languages, operating systems, development and release environments, server capabilities, and other such areas for developing the software.

Iterative and Incremental Approach

And finally, let's take a closer look at the **iterative and incremental approach**.

Using an iterative and incremental approach means treating each iteration as a mini-project. Therefore, you'd develop the software as a number of small mini-projects, working in cycles. You'd develop small working versions of the software at the end of each cycle. Each iteration would add some functionality to the software according to the requirements specified by the customer.

Now that we saw the distinctive characteristics of the unified process model, let's take a look at the process steps involved.

Unified Process Model Phases

We'll go through the four different phases, one at a time, here:

1. **Inception**: The inception phase is similar to the requirements collection and analysis stage of the waterfall model of software development. In this phase, you'd collect requirements from the customer and analyze the project's feasibility, its cost, risks, and profits.

2. **Elaboration**: In this phase, you'd be expanding upon the activities undertaken in the inception phase. The major goals of this phase include creating fully functional requirements (use-cases) and creating a detailed architecture for fulfillment of the requirements. You'd also prepare a business case document for the customer.

3. **Construction**: In this phase, you'd be writing actual code and implementing the features for each iteration. You'd be rolling out the first iteration of the software depending on the key use-cases that make up the core functionalities of the software system.

4. **Transition**: In this phase, you'd be rolling out the next iterations to the customer and fixing bugs for previous releases. You would also deploy builds of the software to the customer.