Department of Computer Science and Engineering
Subject Notes
IT-6002- Software Engineering
UNIT-I

Introduction

Software: -
Software is nothing but collection of computer programs and related documents that are planned to provide desired features, functionalities and better performance.

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: The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.
Software product classified in 2 classes:
1. **Generic software**: developed to solution whose requirements are very common fairly stable and well understood by software engineer.
2. **Custom software**: developed for a single customer according to their specification.

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.
Large software - It is easier to build a wall than to a house or building, likewise, as the size of software become 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 the cost of software remains high if proper process is not adapted.
Dynamic Nature- The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
Quality Management- Better process of software development provides better and quality software product.

Software Engineering Goals
Readability (understood by those who maintain it)
Correctness
Reliability (high performance)
Re usability
Extensibility (ability to perform its operations)
Flexibility
Efficiency

Software Crisis

- Many software projects failed.
- Many software projects late, over budget, providing unreliable software that is expensive to maintain.
- Many software projects produced software which did not satisfy the requirements of the customer.
- Complexities of software projects increased as hardware Capability increased.
- Larger software system is more difficult and expensive to maintain.
- Demand of new software increased faster than ability to generate new software.

Characteristics of good software: -

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:
Operational
Transitional
Maintenance
Well-engineered and crafted software is expected to have the following characteristics:

Operational: -

This tells us how well software works in operations. It can be measured on:
Budget
Usability
Efficiency
Correctness
Functionality
Dependability
Security
Safety

Transitional: -

This aspect is important when the software is moved from one platform to another:
Portability
Interoperability
Reusability
Adaptability

Maintenance: -

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:
Modularity
Maintainability
Flexibility
Scalability
In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

Software Problem and Prospects:-

Software engineering is the systematic approach to the development, operation, maintenance, and retirement of software. There are few fundamental problems that are:-

- The Problem of scale: A fundamental problem of software engineering is the problem of scale; the methods that are used for developing small systems generally do not scale up to large systems. A different set of methods has to be used for developing large software.
- Cost, schedule and quality: The cost of developing a system is the cost of the resources used for the system, which, in the case of software, are the manpower, hardware, software, and the other support resources.
- The Problem of consistency: Though high quality, low cost and small cycle time are the primary objectives of any project, for an organization there is another goal: consistency. An organization involved in software development does not just want low cost and high quality for a project, but it wants these consistently.

Software Development Process Model: -

Software process can be defining as the structured set of activates that are required to develop the software system.

To solve actual problems in an industry setting, a software engineer or a team of engineers must incorporate a development strategy that encompasses the process, methods, and tools layers. This strategy is often referred to as a process model or a software engineering paradigm.

A process model for software engineering is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.

Goal of Software Process Models: -

The goal of a software process model is to provide guidance for systematically coordinating and controlling the tasks that must be performed in order to achieve the end product and their project objectives. A process model defines the following:

- A set of tasks that need to be performed.
- The inputs to and output from each task.
- The preconditions and post-conditions for each task.
- The sequence and flow of these tasks.

We might ask whether a software development process is necessary if there is only one person developing the software. The answer is that it depends. If the software development process is viewed as only a coordinating and controlling agent, then there is no need since there is only one person. However, if the process is viewed as prescriptive road map for generating various intermediate deliverables in addition to the executable code- for

Characteristics of Software Process: -

Software is often the single largest cost item in a computer-based application. Though software is a product, it is different from other physical products.

i. Software costs are concentrated in engineering (analysis and design) and not in production.
ii. Cost of software is not dependent on volume of production.
iii. Software does not wear out (in the physical sense).
iv. Software has no replacement (spare) parts.
v. Software maintenance is a difficult problem and is very different from hardware (physical product) maintenance.
vi. Most software is custom-built.
vii. Many legal issues are involved (e.g. inter-actual property rights, liability).

Software Product: -

A software product, user interface must be carefully designed and implemented because developers of that

product and users of that product are totally different. In case of a program, very little documentation is expected, but a software product must be well documented. A program can be developed according to the programmer's individual style of development, but a software product must be developed using the accepted software engineering principles.

Various Operational Characteristics of software are:

- **Correctness:** The software which we are making should meet all the specifications stated by the customer.
- **Usability/Learn-ability:** The amount of efforts or time required to learn how to use the software should be less. This makes the software user-friendly even for IT-illiterate people.
- **Integrity:** Just like medicines have side-effects, in the same way software may have aside-effect i.e. it may affect the working of another application. But quality software should not have side effects.
- **Reliability:** The software product should not have any defects. Not only this, it shouldn't fail while execution.
- **Efficiency:** This characteristic relates to the way software uses the available resources. The software should make effective use of the storage space and execute command as per desired timing requirements.
- **Security:** With the increase in security threats nowadays, this factor is gaining importance. The software shouldn't have ill effects on data / hardware. Proper measures should be taken to keep data secure from external threats.
- **Safety:** The software should not be hazardous to the environment/life.

Difference between software process and software process: -

Software Process	**Software Product**
Processes are developed by individual user and it is used for personal use.	It is developed by multiple users and it is used by large number of people or customers.
Process may be small in size and possessing limited functionality.	It consists of multiple program codes; relate documents such as SRS, designing documents, user manuals, test cases.
Process is generally developed by process engineers.	Process is generally developed by process engineers. Therefore systematic approach of developing software product must be applied.
Software product relies on software process for its stability quality and control Only one person uses the process, hence lack of user interface	It is important than software product. Multiuser no lack of user interface.

Software Development Life Cycle/Process model/ Software Development Life Cycle: -

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product. it is a team of engineers must incorporate a development strategy that encompasses the process, method and tools layers. Each phase has various activities to develop the software product. It also specifies the order in which each phase must be executed.

A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed.

The software development paradigm helps developer to select a strategy to develop the software. A software development paradigm has its own set of tools, methods and procedures, which are expressed clearly and defines software development life cycle.

Definition: Software Development Life Cycle (SDLC) is a process used by software industry to design, develop and test high quality software. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.
SDLC is the acronym of Software Development Life Cycle. It is also called as Software development process. The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

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

BASIC ACTIVITIES THAT CAN BE CARIED OUT IN LIFE CYCLE MODEL ARE:
A few of software development paradigms or process models are defined in Fig:-1.1:

1. **Waterfall model or linear sequential model or classic life cycle model: -**

Sometimes called the classic life cycle or the waterfall model, the linear sequential model suggests a systematic, sequential approach5 to software development that begins at the system level and progresses through analysis, design, coding, testing, and maintenance.

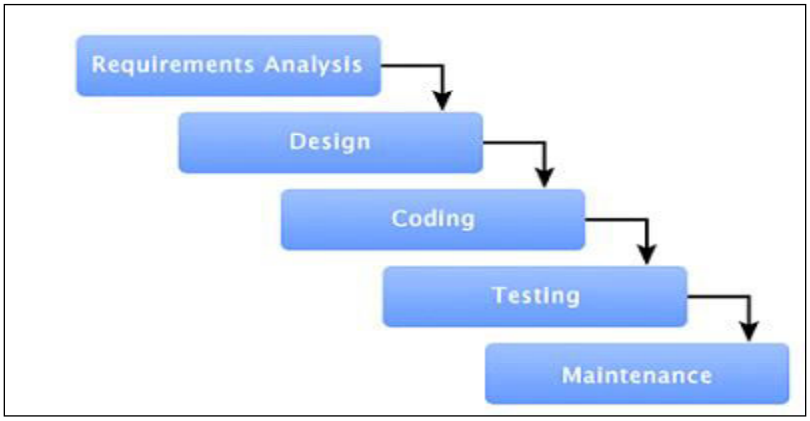

Fig 1.1

Software requirements analysis: The requirements gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.
Design: Software design is actually a multi-step process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration.
Code generation: The design must be translated into a machine-readable form. The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.
Testing: Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

Maintenance: Software will undoubtedly undergo change after it is delivered to the customer (a possible exception is embedded software). Change will occur because errors have been encountered, because the software must be adapted to accommodate changes in its external environment (e.g., a change required because of a new operating system or peripheral device), or because the customer requires functional or performance enhancements. Software support/maintenance reapplies each of the preceding phases to an existing program rather than a new one.

Advantages of waterfall model: -

This model is simple and easy to understand and use.
Waterfall model works well for smaller projects where requirements are very well understood.
Each phase proceeds sequentially.
Documentation is produced at every stage of the software's development. This makes understanding the product designing procedure, simpler.
After every major stage of software coding, testing is done to check the correct running of the code. help us to control schedules and budgets.

Disadvantages of waterfall model: -

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.
High amounts of risk and uncertainty.
Customer can see working model of the project only at the end. after reviewing of the working model if the customer gets dissatisfied then it causes serious problem.
You cannot go back a step if the design phase has gone wrong, things can get very complicated in the implementation phase.

Other Models

Prototype Model: -

A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system. The Fig:-1.2 is shown as below-

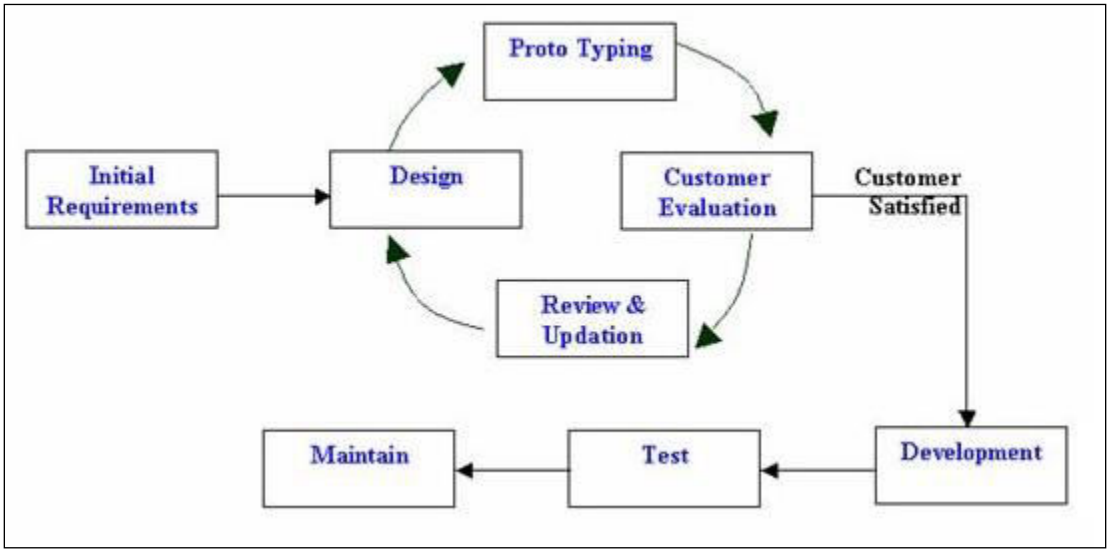

Fig 1.2

Need for a prototype in software development: -

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

- how the screens might look like
- how the user interface would behave
- how the system would produce outputs

This is something similar to what the architectural designers of a building do; they show a prototype of the building to their customer. The customer can evaluate whether he likes it or not and the changes that he would need in the actual product. A similar thing happens in the case of a software product and its prototyping model.

Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product.

A prototyping model can be used when technical solutions are unclear to the development team.

A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

A prototype of the actual product is preferred in situations such as:

- user requirements are not complete
- technical issues are not clear

Two most popular prototyping approaches are: -

Throw away prototyping approaches

Evolutionary prototyping approaches

Throwaway prototyping: -

Also called close-ended prototyping. Throwaway or Rapid Prototyping refers to the creation of a model that will eventually be discarded rather than becoming part of the final delivered software. After preliminary requirements gathering is accomplished, a simple working model of the system is constructed to visually show the users what their requirements may look like when they are implemented into a finished system.

Evolutionary prototyping: -

Evolutionary Prototyping (also known as breadboard prototyping) is quite different from Throwaway Prototyping. The main goal when using Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built. When developing a system using Evolutionary Prototyping, the system is continually refined and rebuilt. Evolutionary prototyping acknowledges that we do not understand all the requirements and builds only those that are well understood. This technique allows the development team to add features, or make changes that couldn't be conceived during the requirements and design phase.

Evolutionary Prototypes have an advantage over Throwaway Prototypes in that they are functional systems. Although they may not have all the features the users have planned, they may be used on an interim basis until the final system is delivered.

It is not unusual within a prototyping environment for the user to put an initial prototype to practical use while

waiting for a more developed version. The user may decide that a 'flawed' system is better than no system at all. In Evolutionary Prototyping, developers can focus themselves to develop parts of the system that they understand instead of working on developing a whole system. To minimize risk, the developer does not implement poorly understood features. The partial system is sent to customer sites. As users work with the system, they detect opportunities for new features and give requests for these features to developers. Developers then take these enhancement requests along with their own and use sound configuration-management practices to change the software-requirements specification, update the design, recode and retest.

Advantages of Prototyping Model: -

When prototype is shown to the user, he gets a proper clarity and 'feel' of the functionality of the software and he can suggest changes and modifications and increasing user confidence.

When client is not confident about the developer's capabilities, he asks for a small prototype to be built. Based on this model, he judges capabilities of developer.

It reduces risk of failure, as potential risks can be identified early and mitigation steps can be taken.

Iteration between development team and client provides a very good and conductive environment during project.

Disadvantages of Prototyping Model: -

Once we get proper requirements from client after showing prototype model, it may be of no use. That is why, sometimes we refer to the prototype as "Throw-away" prototype.

It is a slow process.

Too much involvement of client is not always preferred by the developer.

Too many changes can disturb the rhythm of the development team.

RAPID APPLICATION MODEL

Rapid application development (RAD) is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

Since there is no detailed pre-planning, it makes it easier to incorporate the changes within the development process. RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype. The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a “high-speed” adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods (e.g., 60 to 90 days). Used primarily for information systems applications, the RAD approach encompasses the following phases shown in Fig:-1.3:

Business modeling. The information flow among business functions is modeled in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

Data modeling. The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called *attributes*) of each object are identified and the relationships between these objects defined.

Process modeling. The data objects defined in the data modeling phase are transformed to achieve the

information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

Application generation. RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

Testing and turnover. Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.

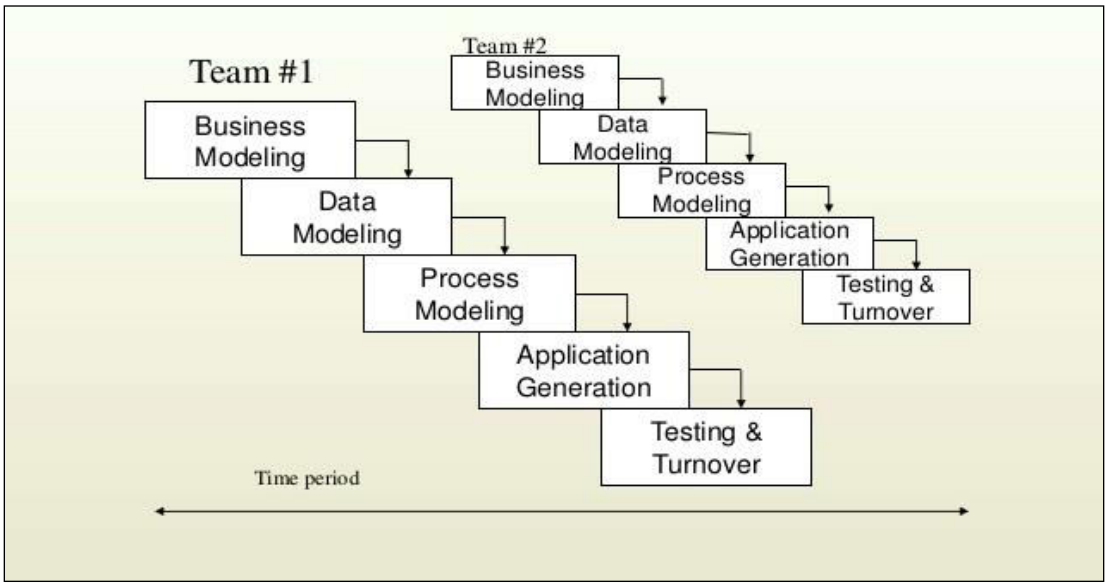

Fig 1.3

Like all process models, the **RAD approach has drawbacks**:

➔ For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.

➔ RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system complete in a much-abbreviated time frame. If commitment is lacking from either constituency, RAD projects will fail.

➔ Not all types of applications are appropriate for RAD. If a system cannot be properly modularized, building the components necessary for RAD will be problematic. If high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.

➔ RAD is not appropriate when technical risks are high. This occurs when a new application makes heavy use of new technology or when the new software requires a high degree of interoperability with existing computer programs.

Advantages of the RAD model:

- Reduced development time.
- Increases re usability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

Disadvantages of RAD model:

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

When to use RAD model:

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

Evolutionary Process Model: -

There is growing recognition that software, like all complex systems, evolves over a period of time. Business and product requirements often change as development proceeds, making a straight path to an end product unrealistic; tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure; a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined. In these and similar situations, software engineers need a process model that has been explicitly designed to accommodate a product that evolves over time.

It is based on the idea of enveloping an initial implementation a give further detailed and explanation to user for refinement through many versions until good enough to be used or accepted system has been developed. Advantage is the user get chance to experiment with partially developed software much before the complete version of the system released. Core module gets tested thoroughly thereby reducing chance of errors in the core modules to final product. it is difficult to divide to the problem into several versions that would be acceptable to the customer and which can be incrementally implemented and delivered.

The linear sequential model is designed for straight-line development. In essence, this waterfall approach assumes that a complete system will be delivered after the linear sequence is completed. The prototyping model is designed to assist the customer (or developer) in understanding requirements.

In general, it is not designed to deliver a production system. The evolutionary nature of software is not considered in either of these classic software engineering paradigms.

Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software.

Iterative Model design:

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Following is the pictorial representation of Iterative and Incremental model shown in Fig:-1.4:

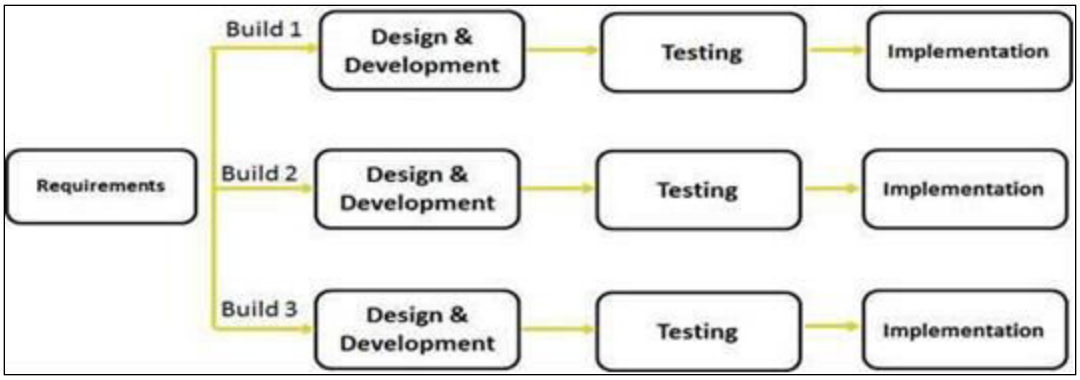

Fig 1.4

Iterative Model Application:

Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. This model is most often used in the following scenarios:

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high-risk features and goals which may change in the future.

Evolutionary Process Model is of 2 types

- Incremental Model And
- Spiral Model

Incremental Model:

The incremental model combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping. the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable “increment” of the software. For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment. It should be noted that the process flow for any increment can incorporate the prototyping paradigm. The Fig. 1.5 is shown as below:-

The first increment is often a core product where the basic requirements are addressed and the supplementary features are added in the next increments. The core product is used and evaluated by the client. Once the core product is evaluated by the client there is plan development for the next increment. Thus, in every increment the needs of the client are kept in mind and more features and functions are added and the core product is updated. This process continues till the complete product is produced.

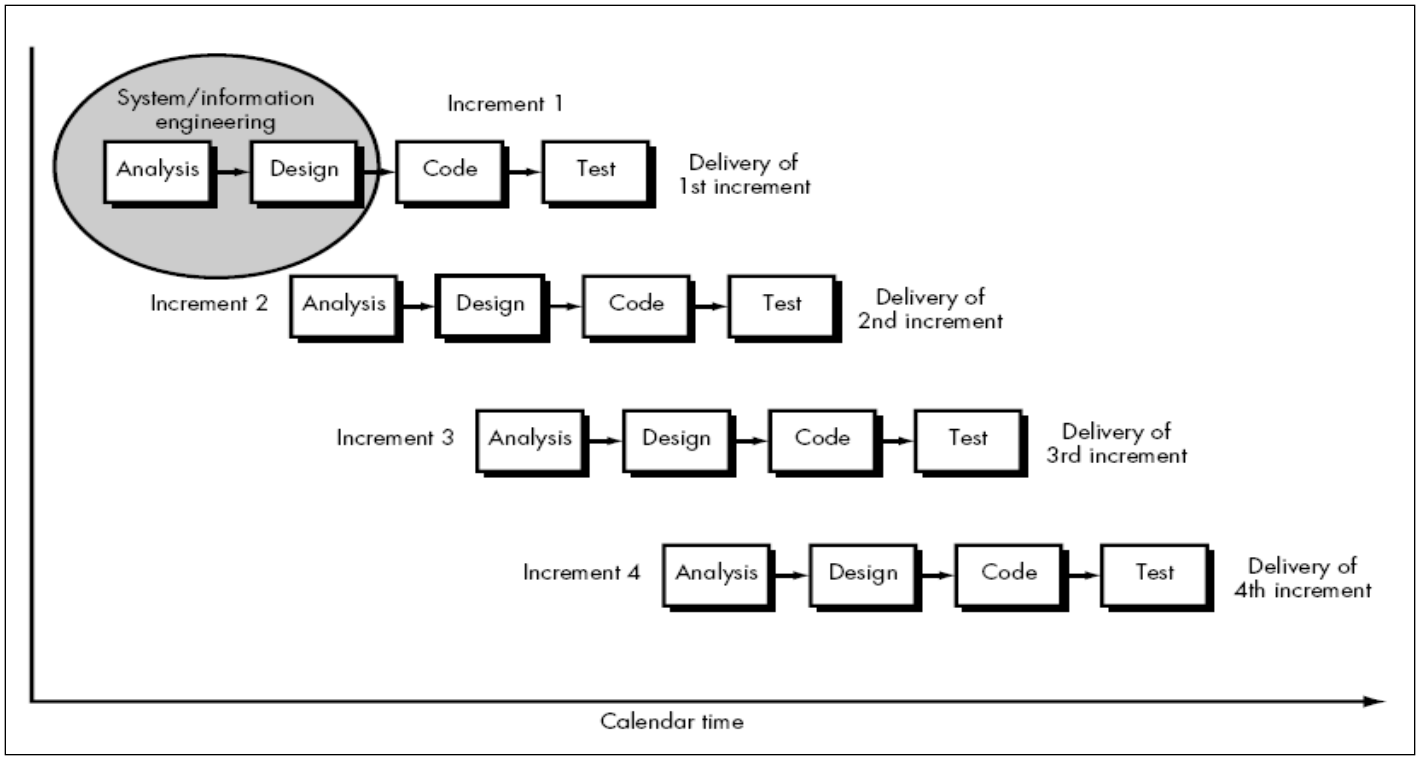

Fig 1.5

Advantages of Incremental model: -

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.
- There is low risk for overall project failure.
- Customer does not have to wait until the entire system is delivered.

Disadvantages of Incremental model: -

- Needs good planning and design at the management a technical level.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.
- Time foundation create problem to complete the project.

When to use the Incremental model:

- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available
- There are some high-risk features and goals.

Spiral Model (Boehm's Model)

The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a number of framework activities, also called task regions. Project entry point axis is defined this axis represents starting point for different types of project. Every framework activities represent one section of the spiral path. As the development process starts, the software team perform activities that are indirect by a path around the spiral model in a clockwise direction. It begins at the center of spiral model.

Typically, there are between three and six task regions. In blow figure depicts a spiral model that contains six task regions:

- **Customer communication**—tasks required to establish effective communication between developer and customer.
- **Planning**—tasks required to define resources, time lines, and other project related information.
- **Risk analysis**—tasks required to assess both technical and management risks.
- **Engineering**—tasks required to build one or more representations of the application.
- **Construction and release**—tasks required to construct, test, install, and provide user support(e.g., documentation and training).
- **Customer evaluation**—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

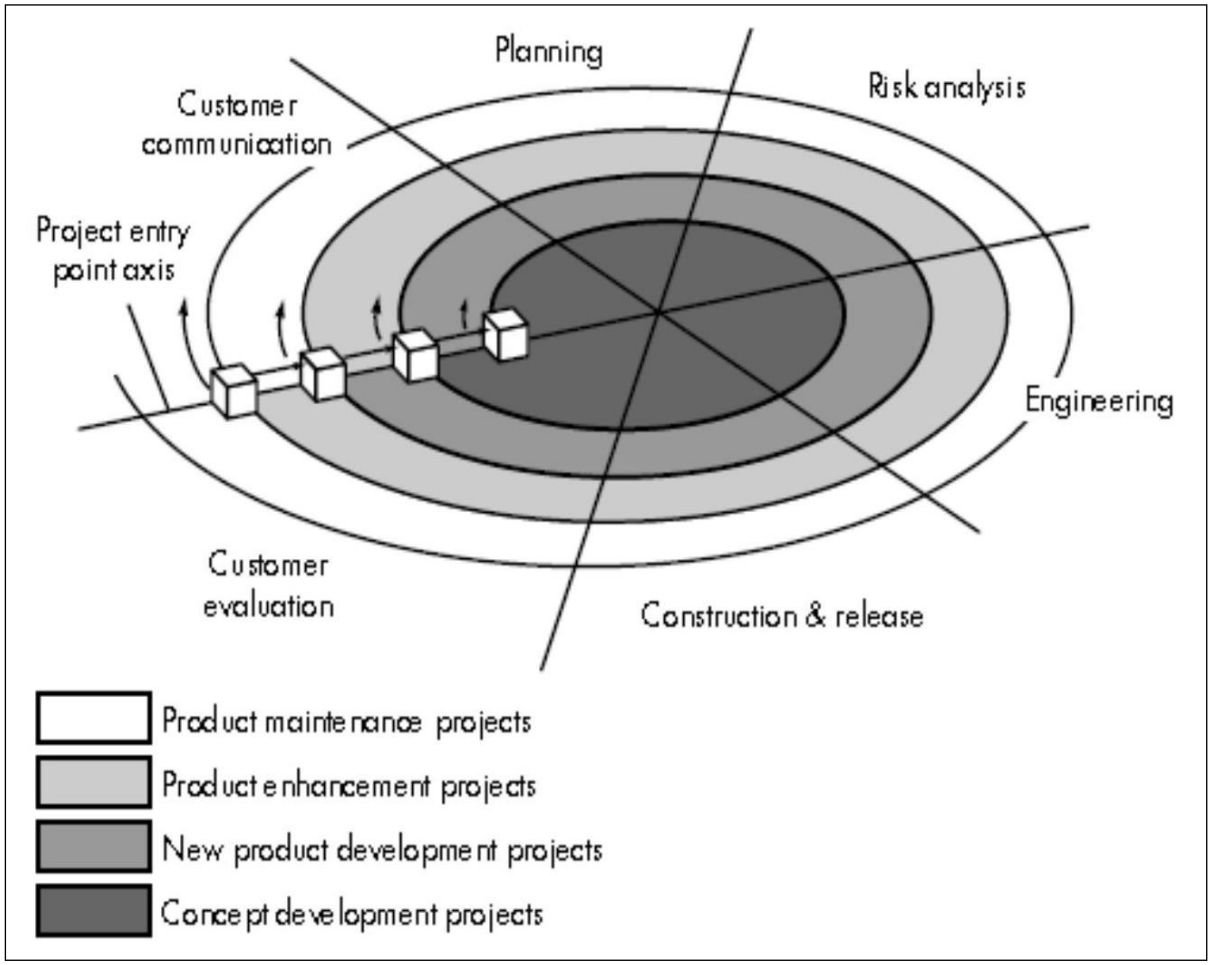

Fig 1.6

Advantages of Spiral model: -

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages of Spiral model: -

Can be a costly model to use.
Risk analysis requires highly specific expertise.
Project's success is highly dependent on the risk analysis phase.
Doesn't work well for smaller projects.

When to use Spiral model:

- When costs and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment unwise because of potential changes to economic priorities.
- Users are unsure of their needs.
- Requirements are complex.
- New product line.
- Significant changes are expected (research and exploration).

Component Assembly Model

The problem is a software development Life Cycle (SDLC) plan called Component Assembly model. Instead of starting over with different codes and languages, developers who use this model tap on the available components and put them together to build a program. Component Assembly Model is an iterative development model. It works like the Prototype model, constantly creating a prototype until software that will cater the need of businesses and consumers are realized.

Component Assembly model has a close resemblance with the Rapid Application Development (RAD) model. This SDLC model uses the available tools and GUIs to build software. With the number of SDKs released today, developers will find it easier to build programs using lesser codes with the help of SDK. Since it has enough time to concentrate on other parts of the programs aside from coding language; RAD concentrates or user inputs and graphical interaction of the user and program.

Component Assembly Model on the other hand uses a lot of previously made components.

CAM doesn't need to use SDKs to develop programs but it will be putting together powerful components. All the developers have to do is to know what the customer wants, look for the components to answer the need and put together the components to create the program.

Component Assembly Model is just like the Prototype model, in which first a prototype is reacted according to the requirements of the customer. Thus, this is one of the most beneficial advantages of component assembly model as it saves lots of time during the software development program.

Component Assembly Model is just like the Prototype model, in which first a prototype is created according to the requirements of the customer and sent to the user for evaluation to get the feedback for the modifications to be made and the same procedure is repeated until the software will cater the need of businesses and consumers is realized. Thus, it is also an iterative development model.

Component Assembly model has been developed to answer the problems faced during the Software Development Life Cycle (SDLC). Instead of searching for different codes and languages, the developers using this model opt for the available components and use them to make an efficient program. Component Assembly Model is an iterative development model that works like the Prototype model and keeps developing

a prototype on the basis of the user feedback until the prototype resembles the specifications provided by the customer and the business.

Moreover, Component Assembly model resembles to the Rapid Application Development (RAD) model and uses the available resources and GUIs to create a software program. Today, a number of SDKs are available that makes it easier for the developers to design a program using less number of codes with the help of SDK. This method has ample of time to concentrate on the other components of the program apart from the coding language, user input and graphical interaction of both user and software program.

In addition to that, a Component Assembly Model uses a number of previously made components and does not need the use of SDK for creating a program but puts the powerful components together to develop an effective an efficient program. Thus, this is one of the most beneficial advantages of component assembly model as it saves lots of time during the software development program.

The developers only need to know the requirements of the customer, look for the useful components that are useful for answering the need of the customer and finally put them together to build a program.

This model work in following manner:

- Identify all required candidate component i.e. classes with the help of application data and algorithm.
- If these candidate component are used in previous software project then thy must be present in library.
- Such preexisting component can be extracted from the library and used for further development.
- But if required component is not presented in the library then build or create the component as per requirement.
- Place the newly created component in library. This makes one iteration of the system.
- Repeat step 1 to 5 for creating 'n' iteration. Where 'n' denotes the no of iterations required to develop complete application.

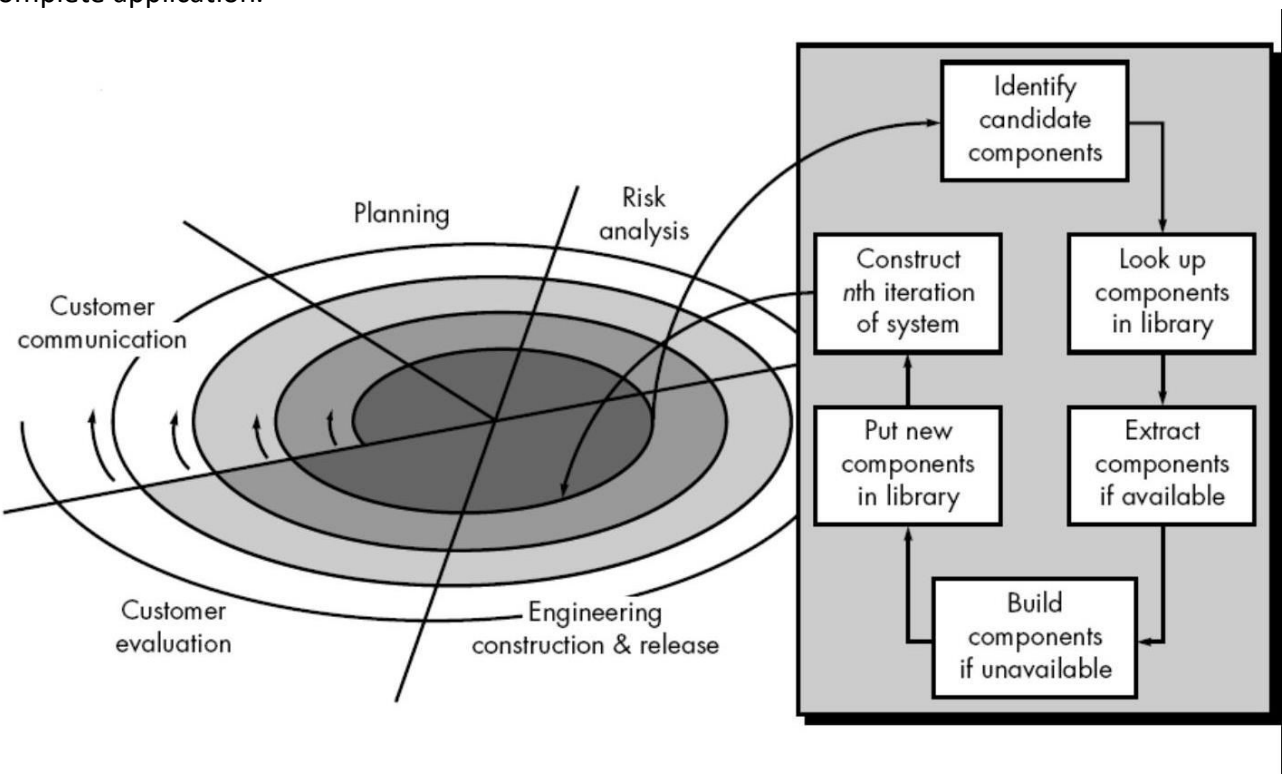

Fig 1.7

Component Assembly Model Characteristics:

- Use of object-oriented technology.
- Components – classes that encapsulate both data and algorithms.
- Components developed to be reusable.
- Paradigm similar to spiral model, but engineering activity involves components.
- System produced by assembling the correct components.

Unified Process

RUP:

The Rational Unified Process (RUP) is an iterative software development process framework created by Rational Software Corporation. RUP is not a single concrete process, but rather an adaptable process framework intended to be tailored by development organizations and software teams. It states that its heart is about successful software development. RUP provides a disciplined approach to assigning tasks and responsibly within a development organization. Its goal is to ensure the production of a high-quality software that meets the need of its end user within a predictable schedule and budget RUP provides each team with member with the guidelines, templates and tool mentors necessary for the entire team to take advantage of the available resources.

Stands for "Rational Unified Process." RUP is a software development process from Rational, a division of IBM. It divides the development process into four distinct phases that each involves business modeling, analysis and design, implementation, testing, and deployment. The four phases are shown in Fig:-1.8:

1. **Inception** - The idea for the project is stated. The development team determines if the project is worth pursuing and what resources will be needed.
2. **Elaboration** - The project's architecture and required resources are further evaluated. Developers consider possible applications of the software and costs associated with the development.
3. **Construction** - The project is developed and completed. The software is designed, written, and tested.
4. **Transition** - The software is released to the public. Final adjustments or updates are made based on feedback from end users.

The RUP development methodology provides a structured way for companies to envision create software programs. Since it provides a specific plan for each step of the development process, it helps prevent resources from being wasted and reduces unexpected development costs.

Advantages of RUP Software Development: -

1. This is a complete methodology in itself with an emphasis on accurate documentation
2. It is pro-actively able to resolve the project risks associated with the client's evolving requirements requiring careful change request management
3. Less time is required for integration as the process of integration goes on throughout the software development life cycle.
4. The development time required is less due to reuse of components.
5. There is online training and tutorial available for this process.

Disadvantages of RUP Software Development: -

1. The team members need to be expert in their field to develop a software under this methodology.
2. The development process is too complex and disorganized.
3. On cutting edge projects which utilize new technology, the reuse of components will not be possible. Hence the time saving one could have made will be impossible to fulfill.
4. Integration throughout the process of software development, in theory sounds a good thing. But on particularly big projects with multiple development streams it will only add to the confusion and cause more issues during the stages of testing.

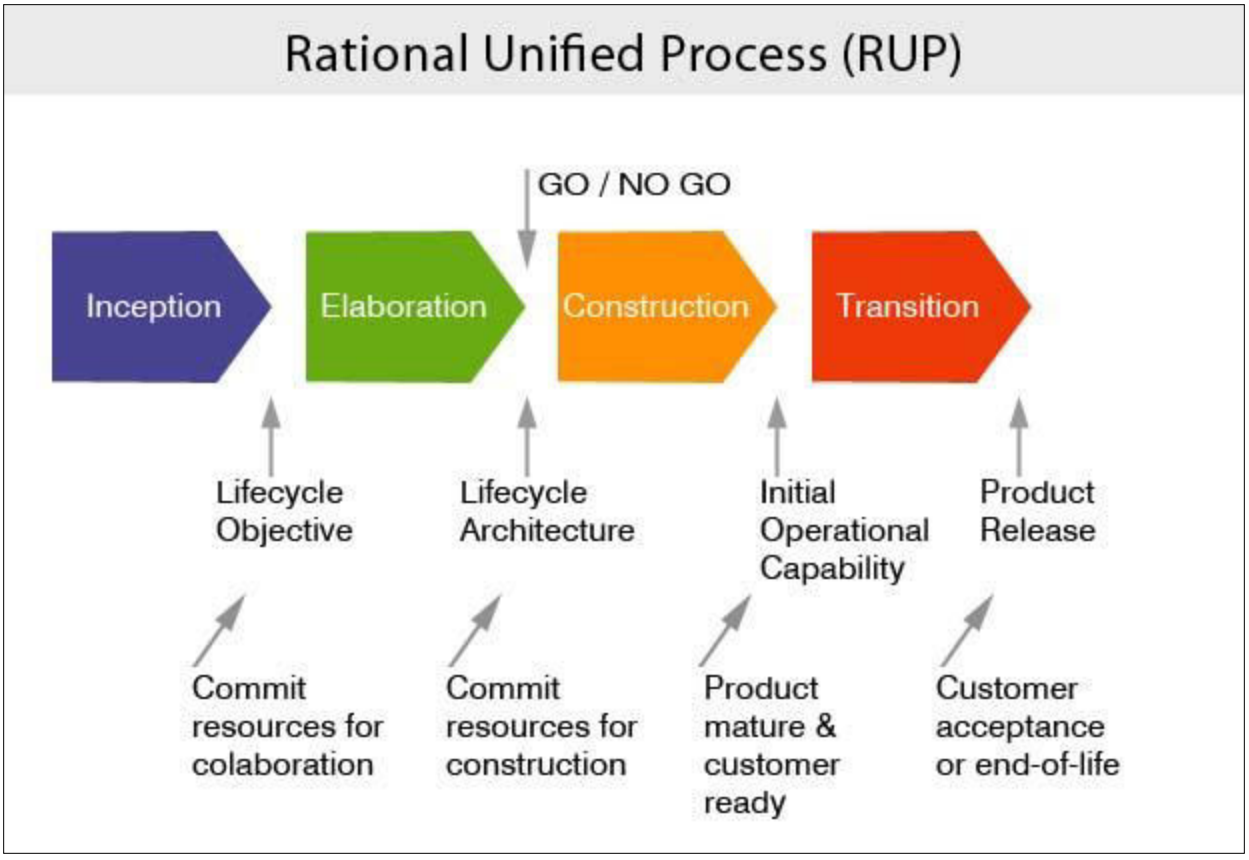

Fig 1.8

Capability Maturity Model (CMM): -

The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach different levels of process maturity. To determine an organization's current state of process maturity, the

SEI uses an assessment that results in a five-point grading scheme. The grading scheme determines compliance with a capability maturity model (CMM) that defines key activities required at different levels of process maturity. The SEI approach provides a measure of the global effectiveness of a company's software engineering practices and establishes five process maturity levels that are defined in Fig:-1.9:

Level 1: Initial. The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

Level 2: Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

Level 3: Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

Level 4: Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level 5: Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

The five levels defined by the SEI were derived as a consequence of evaluating responses to the SEI assessment questionnaire that is based on the CMM. The results of the questionnaire are distilled to a single numerical grade that provides an indication of an organization's process maturity.

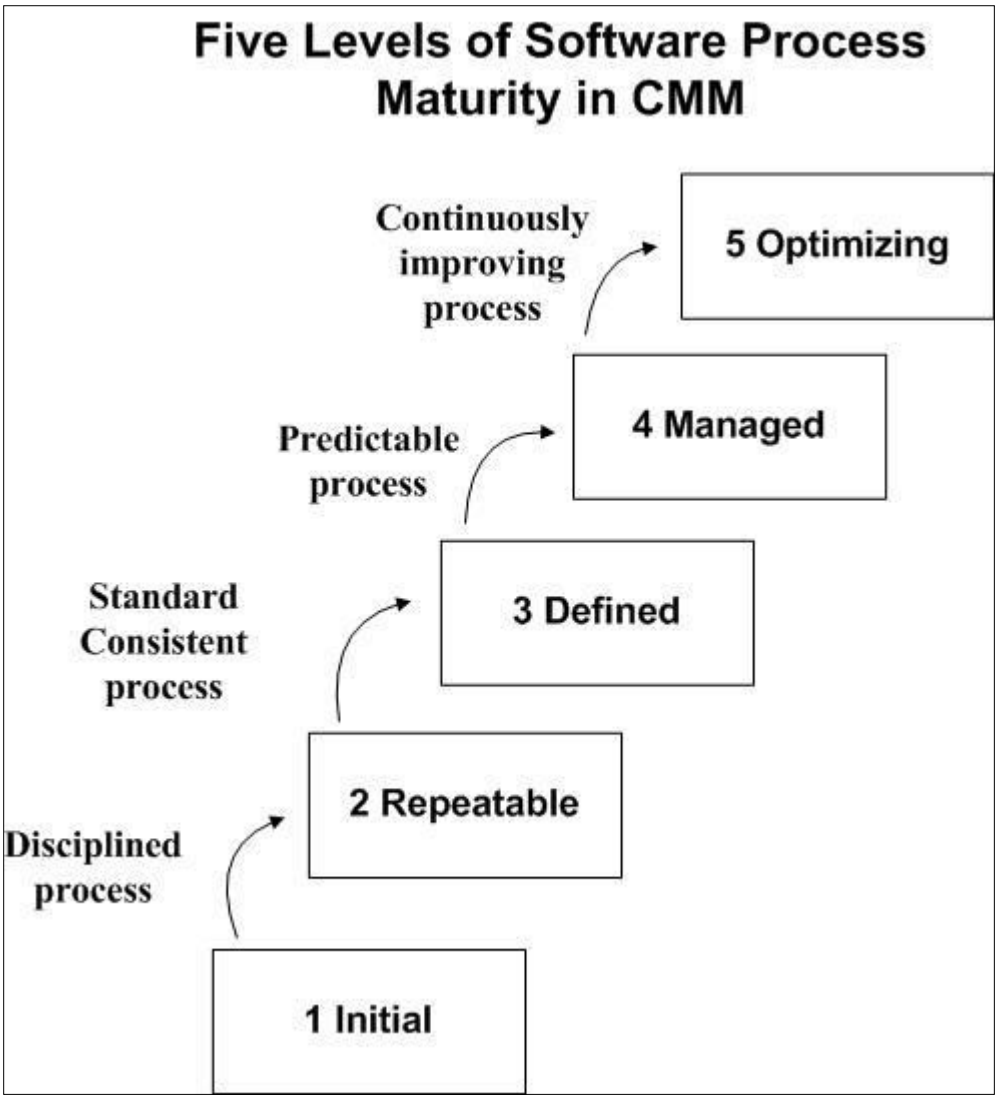

Fig 1.9

Capability Maturity Model (CMM)

The SEI has associated **key process areas** (KPAs) with each of the maturity levels. The KPAs describe those software engineering functions (e.g., software project planning, requirements management) that must be present to satisfy good practice at a particular level. Each KPA is described by identifying the following characteristics:

- *Goals*—the overall objectives that the KPA must achieve.
- *Commitments*—requirements (imposed on the organization) that must be met to achieve the goals or provide proof of intent to comply with the goals.
- *Abilities*—those things that must be in place (organizationally and technically to enable the organization to meet the commitments.
- *Activities*—the specific tasks required to achieve the KPA function.
- *Methods for monitoring implementation*—the manner in which the activities are monitored as they are put into place.
- *Methods for verifying implementation*—the manner in which proper practice for the KPA can be verified.

Process maturity level 2: -

- Software configuration management
- Software quality assurance
- Software subcontract management

- Software project tracking and oversight
- Software project planning
- Requirements management

Process maturity level 3: -

- Peer reviews
- Intergroup coordination
- Software product engineering
- Integrated software management
- Training program
- Organization process definition
- Organization process focus

Process maturity level 4: -

- Software quality management
- Quantitative process management

Process maturity level 5: -

- Process change management
- Technology change management
- Defect prevention

Unified Process Agile Development Model

Agile Process:-The word 'agile' means able to think quickly and clearly.
In business, 'agile' is used for describing ways of planning and doing work wherein it is understood that making changes as needed is an important part of the job.

Agile development model is also a type of Incremental model. Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly tested to ensure software quality is maintained. It is used for time critical applications. Extreme Programming (XP) is currently one of the most well known agile development life cycle model.

Advantages of Agile model:

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

Disadvantages of Agile model:

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development

process. Hence it has no place for newbie programmers, unless combined with experienced resources.

Extreme Programming

Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

Extreme Programming is based on the following values −

- Communication
- Simplicity
- Feedback
- Courage
- Respect

Extreme Programming takes the effective principles and practices to extreme levels.

- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.