Unit 1

Chapter-1

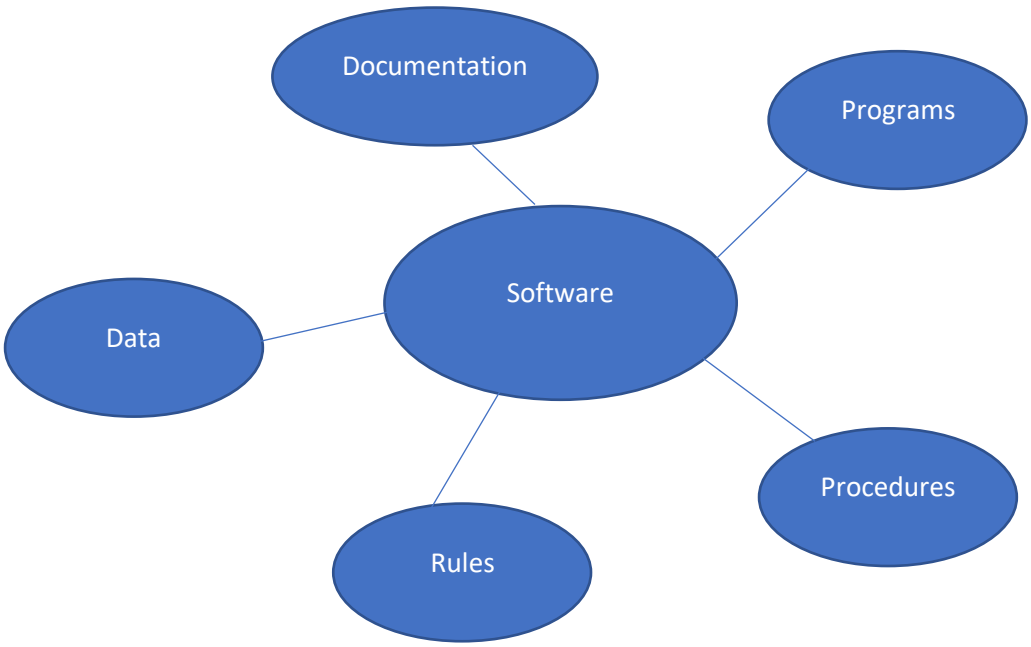
Software and Software Engineering:

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.

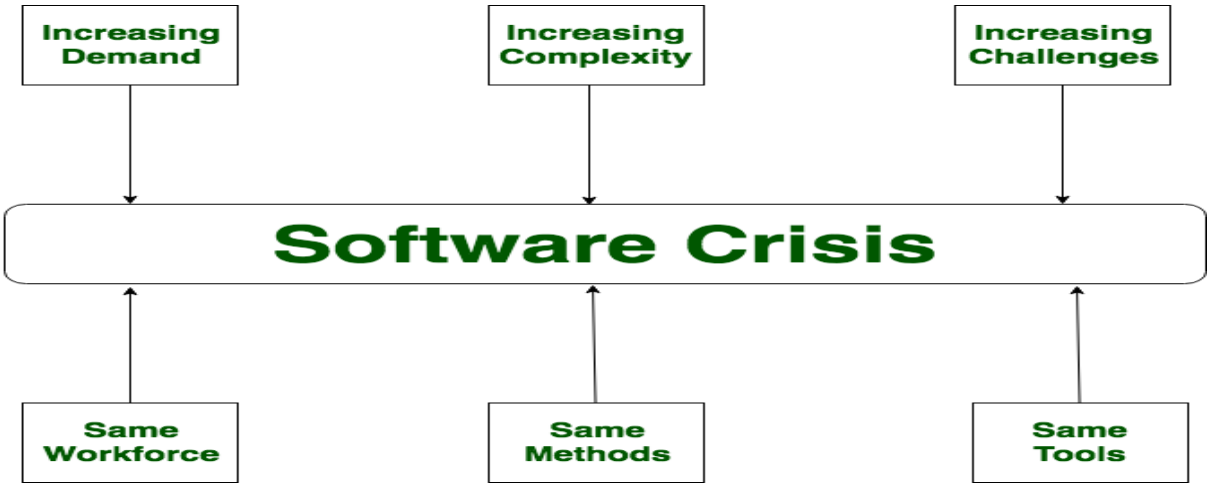


Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

software engineering is a systematic, disciplined, and quantifiable approach.

Software Crisis

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



Causes of Software Crisis:

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

Let's now understand **which factors are contributing to the software crisis.**

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

Solution of Software Crisis:

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

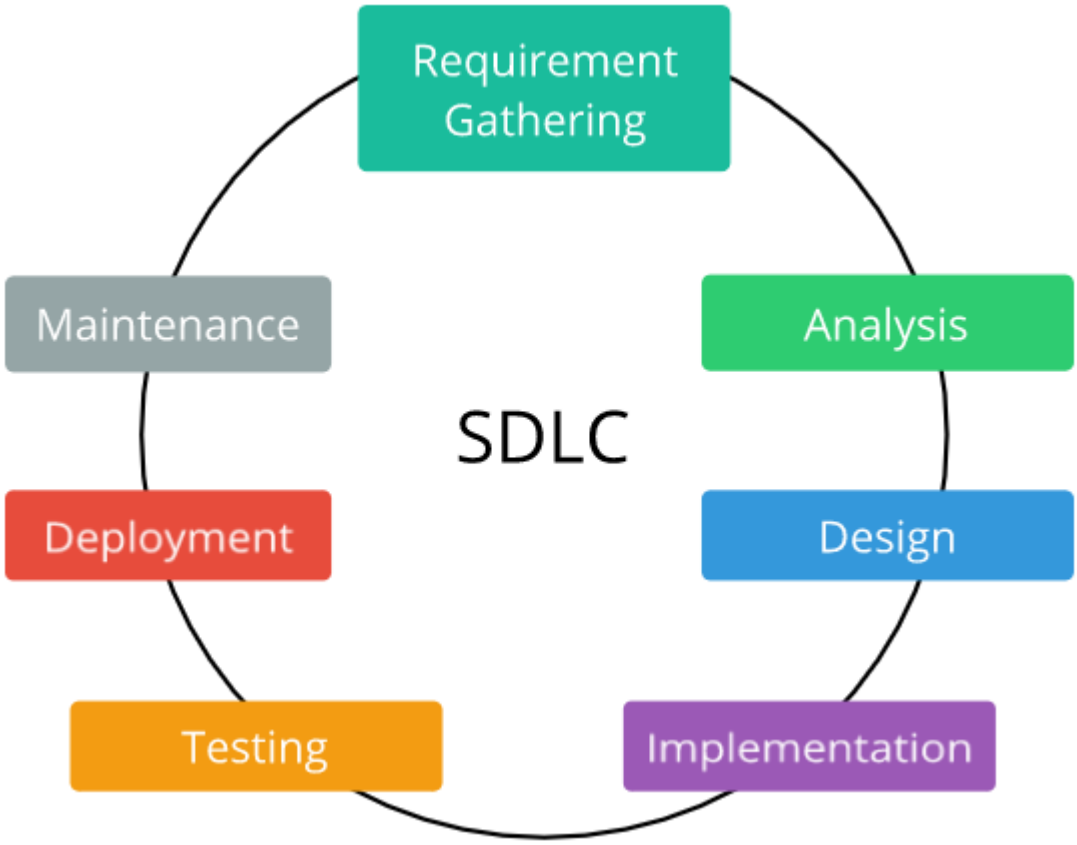
- Reduction in software over budget.
- The quality of software must be high.
- Less time is needed for a software project.
- Experienced and skilled people working over the software project.
- Software must be delivered.
- Software must meet user requirements.

Software Development Lifecycle

SDLC (Software Development Life Cycle) is essentially a process that consists of a series of planned activities to develop, alter, or manage the software or pieces of software. SDLC is a structure to develop a software product that defines the process of planning, implementation, testing, documentation, deployment, and ongoing maintenance and support. The methodology within the SDLC process can vary across industry and organization, but the steps or phases remain same.

Following are the phases of SDLC

1. Requirement gathering
2. Analysis
3. Design
4. Implementation (Coding)
5. Testing
6. Deployment
7. Maintenance



Requirement gathering

This is a fundamental phase of SDLC where user requirement or business requirement is gathered. The requirement is often gathered as an answer to the following questions.

- What is the purpose of developing software?
- Who is going to use the software?
- How the end user will use the software?
- What should be the input for software?
- What will be the output of the software?
- Where will the software be used?

Requirements are usually written in natural language and supplemented by diagrams for better understanding. This phase may include Functional and Non-functional requirements.

- A functional requirement defines a function of a system or its components. A function is described as a set of inputs, the behavior, and outputs.

- The non-functional requirements define system attributes, such as Performance, Scalability, Capacity, Availability, Reliability, Recoverability, Maintainability, and Serviceability.

Analysis

Analysis represents the “What” (What are the actual requirements?) and “How” (How are requirements accomplished independently?) phase. After requirement gathering, the requirement will be analyzed for validity and fulfillment.

The analysis phase starts with the document delivered by the requirement gathering phase and maps it into a full architecture. Detailed information on a programming language, the environment to accomplish the task, team, resources, platforms and algorithms etc. are established in this phase.

Design

In this phase, all systems and software designs are prepared from the specification which comes from the analysis phase. The design phase clearly defines all modules of a system with the help of relational diagrams. Testers will also define their **“***Test Strategy***”** (like what to test and how to test) in this phase.

The clear determination of how the system will look and how it will function can be done in a detailed manner.

Implementation (Coding)

Given the architecture document from the design phase and the requirement document from the analysis phase, the team should implement or build exactly what has been requested through programming code. The main focus of the developer is code here. High-level programming languages are used for coding (such as C#, JAVA, PHP etc). In this phase, the system can be ready to deploy in the user’s live environment.

Testing

Simply everyone knows the quality of the system is very important and the testing phase helps to create good quality. At this phase the whole system is tested to make sure that the system actually fulfills the entire requirement gathered in the Requirement Phase.

As everyone knows, it is hard to find one's own mistake and a fresh eye can find it better. So, the testing team finds defects, flaws, errors or bugs that can be made by the developer and resends it to the respective phase to fix it. All the Functional Testing like Unit Testing, Integration Testing, System Testing, and Non-Functional Testing will be done at this phase.

Deployment

Once the system is developed and tested properly, it will be ready to deploy, which means ready to deliver to the user. The system is not released fully in the first place or we can say that release can be done in a limited segment first and tested. It is called “Beta Version”. If any bugs are found at this stage, it will be reported to the engineering team and after solving it the final deployment will take place.

Maintenance

Once the system is delivered to the customer, the maintenance phase starts. While using the system, if the user gets any problem it will be solved in this phase by the organizations’ support team. The team at this stage makes sure that the released system will not make any problems for the users who are using it.

SDLC Models

There are various Software Development Life Cycle Models that have been defined according to different requirements. Each model has its unique series of steps or approaches to successfully accomplish user requirements.

Following are some popular SDLC models/ Process Models,

- Waterfall Model
- Spiral Model
- V Model
- Incremental Model
- RAD Model and etc..

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.

Characteristics of Software

There're many characters for good software. They include the following.

- Software is developed or engineered, but not Manufactured.
- Software does not wear out.
- Software is custer built Customer.

- **Correctness**

Correctness is important for good software. There shouldn't be faults with specification, design or implementation.

- **Usability**

Users should be able to learn and use a system easily.

- **Efficiency**

The less resource a piece of software uses, the better. Processor, memory and disk space usage should be minimized.

- **Reliability**

A system that can perform the required functions stably is important. Failures should be as rare as possible.

- **Integrity**

Security should be taken into account. Our software should let attackers access unauthorized resources.

Also, data validation is important so that bad data can’t be saved into the system.

- **Adaptability**

A system that can be used without modification for different situations it’s good.

- **Accuracy**

The accuracy of its outputs is good. This measures if the software outputs the right results for users.

- **Robustness**

If a system is still working after getting invalid inputs and stressful environmental conditions, then it’s good for our system.

- **Maintainability**

The ease in which an existing system can be changed is important. The easier that we can make changes, the better.

- **Portability**

A system that operates in different environments from which it's originally designed makes it good.

- **Reusability**

The more reusable parts that a piece of software has, the better.

Using reusable parts means that we don't have to reuse them from scratch.

- **Readability**

Easy to read code is easy to change code. If we understand them faster, then we can make changes faster and in a less error-prone way.

- **Testability**

Making our software system testable is critical. If our code is easy to write unit tests for, then that's good.

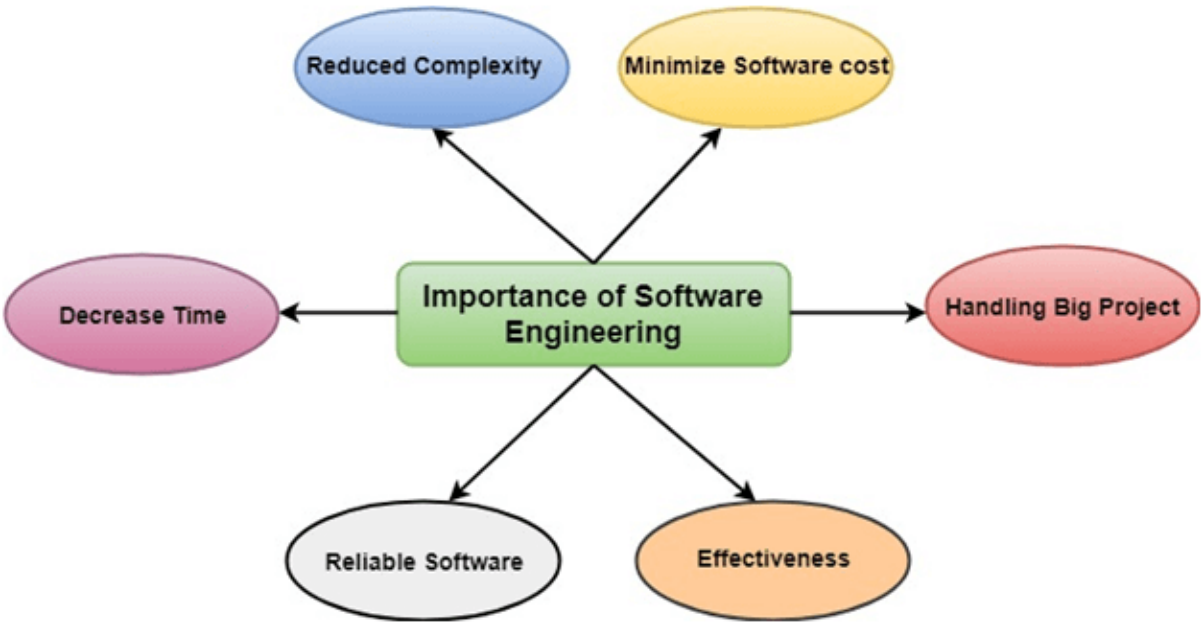
- **Understandability**

The ability for us to understand our system in a global view or at the detailed code level is important.

Easier to understand systems are more coherent.

With good software, it's harder for us to create defects, and it's faster to make changes.

Importance of Software Engineering



The importance of Software engineering is as follows:

1. **Reduces complexity:**

   Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.

2. **To minimize software cost:**

   Software needs a lot of hardwork and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.

3. **To decrease time:**

   Anything that is not made according to the project always wastes time. And if you are making great software, then you may

need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So if you are making your software according to the software engineering method, then it will decrease a lot of time.

4. **Handling big projects:**

Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So to handle a big project without any problem, the company has to go for a software engineering method.

5. **Reliable software:**

Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.

6. **Effectiveness:**

Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So Software becomes more effective in the act with the help of software engineering.

THE CHANGING NATURE OF SOFTWARE:

The 7 broad categories of computer software present continuing challenges for software engineers:

* System software
* Application software
* Engineering/scientific software
* Embedded software
* Product-line software
* Web-applications software
* Artificial intelligence software.

System software:

System software is a collection of programs written to service other programs. The systems software is characterized by heavy interaction with computer hardware heavy usage by multiple users concurrent operation that requires scheduling, resource sharing, and sophisticated process management complex data structures multiple external interfaces

E.g. compilers, editors and file management utilities.

Application software:

Application software consists of standalone programs that solve a specific business need. It facilitates business operations or management/technical decision making. It is used to control business functions in real-time

E.g. point-of-sale transaction processing, real-time manufacturing process control.

Engineering/Scientific software:

Engineering and scientific applicationsrange -from astronomy to volcanology - from automotive stress analysis to space shuttle orbital dynamics - from molecular biology to automated manufacturing

E.g. computer aided design, system simulation and other interactive applications.

Embedded software: Embedded software resides within a product or system and is used to implement and control features and functions for the end-user and for the system itself. It can perform limited and esoteric functions or provide significant function and control capability. SOFTWARE ENGINEERING Page 3

E.g. Digital functions in automobile, dashboard displays, braking systems etc.

Product-line software:

Designed to provide a specific capability for use by many different customers, product-line software can focus on a limited and esoteric market place or address mass consumer markets

E.g. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications

Web-applications:

WebApps are evolving into sophisticated computing environments that not only provide standalone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications.

Artificial intelligence software:

AI software makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Application within this area includes robotics, expert systems, pattern recognition, artificial neural networks, theorem proving, and game playing.

Legacy Software

Legacy software are older programs that are developed decades ago.

The Quality of the legacy software is poor. because it has inextensible design, convoluted code, poor and nonexistent documentation, test cases and results that are not achieved. A legacy system's older

Main characteristics of legacy systems :

- They are implemented on old technologies and platforms.
- Outdated development, design, and architecture approaches are used.
- No unit and integration tests.
- The system is difficult to make changes to.
- The system breaks down unexpectedly.
- Bad unreadable code that calls into question the operation of the entire system.
- Routine operations are not automated, which periodically leads to the same type of errors and increases the bus factor, which is the level of specific knowledge that certain team members have. The higher this factor, the more difficult it becomes to continue developing the project after those team members are replaced by others.
- System and infrastructure not properly documented.

Advantages of Legacy Systems

1. Familiarity

It is a system that everyone is familiar with, so there are no roadblocks in their operation. Nobody feels stuck while trying to

execute a specific function and need assistance to complete a specific task. All the team members know how they can access to various records & in which tab they are present. This helps them filtering out, performing daily tasks, and generating reports with ease and quickly.

2. Steadiness

Business steadiness is essential. So a legacy system that works and keeps every one of the teams on the same page is good for the organization. Whenever a new team member joins the organization, the system provides him/her with enough trained resources to help him/her become comfortable with the company's technology.

3. Efficiency

There are many newer & more powerful devices and applications. But legacy system upgrades are helpful if the costs of opting for a new and developed system are high. Moreover, if you are not getting for what you are paying, there is no need to replace irrespective of the problems with legacy systems.

Disadvantages of Legacy systems

1. **Cost of maintenance**

Maintaining a system may ultimately become more expensive than replacing the hardware and software. Some businesses invest much money into a system that they propose to get the maximum **ROI**. But over a period of time, that technology is no longer the best in class, and newer versions on the market could do things better.
Because of the initial investment into the newly outdated system, firms may decide to stick with their share.

2. **Lack of understanding & security issues with legacy systems**

Maintaining and expanding systems can be difficult because of the lack of understanding of their implementation and execution. Security

issues with legacy systems may lead to vulnerabilities, and this puts the system at risk of being bargained.

3. **IT specialists retiring**

IT workers are retiring, and younger professionals do not know the appropriate dealing with legacy systems. This means that when these legacy professionals retire, they can be very hard to be replaced. This is enough to make stuff even worse. Hardware makers are, in many cases, no longer proposing support for their legacy products.

4. **Mobile growth on hold**

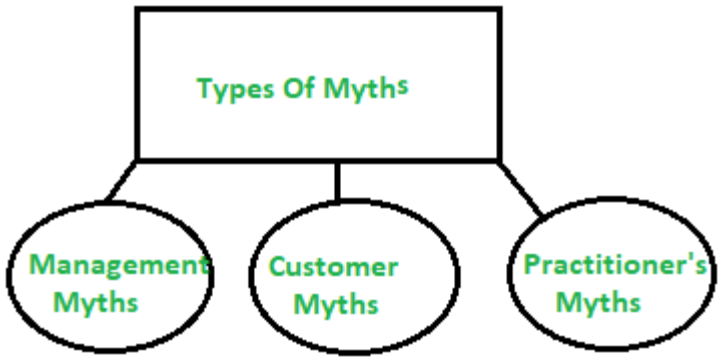
Ask yourself, do you always switch on a laptop/PC, or do you favor to use a mobile device, for instance, a tablet or a smartphone? Legacy systems risk in dealing with rapidly expanding mobile services, as well as being incapable of offering the competencies necessary to propose services on social networks and other similar platforms.

5. **Compatibility Issues**

Legacy systems typically support file and data setups up to a point and evolve afar what the legacy system can lever, which frequently only takes a few years or so. It means business wedges using older setups that their customers, followers, or dealers may not be able to use for the long-term.

Software Myths

Software Myths:
Most, experienced experts have seen myths or superstitions (false beliefs or interpretations) misleading attitudes (naked users) which creates Major problems for management and technical people. The opposite Types of software-related myths are listed below.



Types of Software Myths

(i)Management Myths:

Manages with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality.

Myth 1:

We have all the standards and procedures available for software development i.e. the software developer has all the reqd.

Fact:

- Software experts do not know that there are all of them levels.
- Such practices may or may not be expired at present / modern software engineering methods.
- And all existing processes are incomplete.

Myth 2:

The addition of the latest hardware programs will improve the software development.

Fact:

- The role of the latest hardware is not very high on standard software development; instead (CASE) Engineering tools help the computer they are more important than hardware to produce quality and productivity.
- Hence, the hardware resources are misused.

Myth 3:

- Managers think that, with the addition of more people and program planners to Software development can help meet project deadlines (If lagging behind).

Fact:

- Software development is not, the process of doing things like production; here the addition of people in previous stages can reduce the time it will be used for productive development, as the newcomers would take time existing developers of definitions and understanding of the file project. However, planned additions are organized and organized It can help complete the project.

(ii)Customer Myths:

The customer can be the direct users of the software, the technical team, marketing / sales department, or other company. Customer has myths

Leading to false expectations (customer) & that's why you create dissatisfaction with the developer.

Myth 1 :

A general statement of intent is enough to start writing plans (software development) and details of objectives can be done over time.

Fact:

- Official and detailed description of the database function, ethical performance, communication, structural issues and the verification process are important.
- It is happening that the complete communication between the customer and the developer is required.

Myth 2 :

- Project requirements continue to change, but change can be easy location due to the flexible nature of the software.

Fact:

- Changes were made to the final stages of software development but cost to make those changes grow through the latest stages of Development.
- A detailed analysis of user needs should be done to minimize change requirement.

(iii)Practitioner's Myths:

Myths that are still believed by software practitioners: during the early days of software, programming was viewed as an art from old ways and attitudes die hard.

Myths 1 :
They believe that their work has been completed with the writing of the plan and they received it to work.

Fact:

- It is true that every 60-80% effort goes into the maintenance phase (as of the latter software release). Efforts are required, where the product is available first delivered to customers.

Myths 2:
There is no other way to achieve system quality, behind it done running.

Fact:

- Systematic review of project technology is the quality of effective software verification method. These updates are quality filters and more accessible than test.

Myth 3:
An operating system is the only product that can be successfully exported project.

Fact:

- A working system is not enough, it is just the right document brochures and booklets are also reqd. To provide for guidance & software support.

Myth4:

Engineering software will enable us to build powerful and unnecessary document & always delay us.

Fact:

- Software engineering does not deal with text building, rather while creating better quality leads to reduced recycling & this is being studied for rapid product delivery.

Unit -1 Chapter-2

Software Process and Process Models

Software Process

A software process (also known 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 constraints around them.
2. S**oftware design and implementation**: The software is to be designed and programmed.
3. **Software verification and validation**: The software must conform to its specification and meet the customer's needs.
4. **Software evolution** (software maintenance): The software is being modified to meet customer and market requirements changes.

Software engineering - Layered technology

Software Engineering is a layered technology. It is the application of principles uses in the field of engineering, which usually deals with

physical systems, to the design, development, testing, deployment and management of systems.

The main objective of software engineering layers is to help software developers obtain high-quality software. There are four types of layers in Software Engineering such as – **Tools, methods, process, A quality focus**.

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

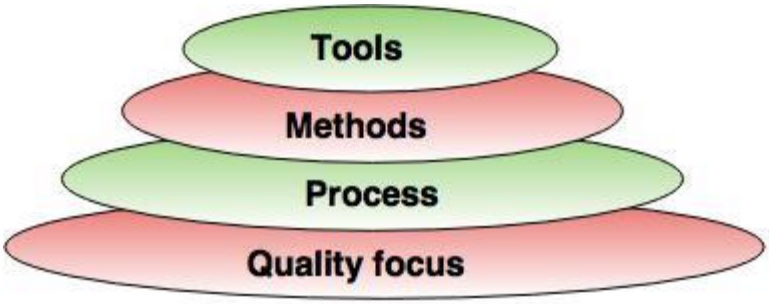


Fig. - Software Engineering Layers

The layered technology consists of:

1. Quality focus:

It is a culture that ultimately leads to the development of software engineering.

The characteristics of good quality software are:

- Correctness of the functions required to be performed by the software.
- Maintainability of the software
- Integrity i.e. providing security so that the unauthorized user cannot access information or data.
- Usability i.e. the efforts required to use or operate the software.

2. Process

A process defines who is doing what, when and how to reach a certain goal. The software process forms the basis for management control of software projects. It establishes the context in which technical methods are applied for work products are produced.

- It is the base layer or foundation layer for the software engineering.
- The software process is the key to keep all levels together.
- It defines a framework that includes different activities and tasks.
- In short, it covers all activities, actions and tasks required to be carried out for software development.

3. Methods

- The method provides the answers of all 'how-to' that are asked during the process.
- It provides the technical way to implement the software.
- It includes collection of tasks starting from communication, requirement analysis, analysis and design modelling, program construction, testing and support.

4. Tools

- The software engineering tool is an automated support for the software development.
- The tools are integrated i.e the information created by one tool can be used by the other tool.
- **For example:** The Microsoft publisher can be used as a web designing tool.

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 their size or complexity.
- In addition, the process framework encompasses a set of activities that are applicable across the entire software process.
- Referring the following figure, each framework activity is populated by an asset of software engineering actions- a

collection of related tasks that produces a major software engineering work product (e.g. design is a software engineering action).

- Each action is populated with individual work tasks that accomplish some part of the work implied by the action.

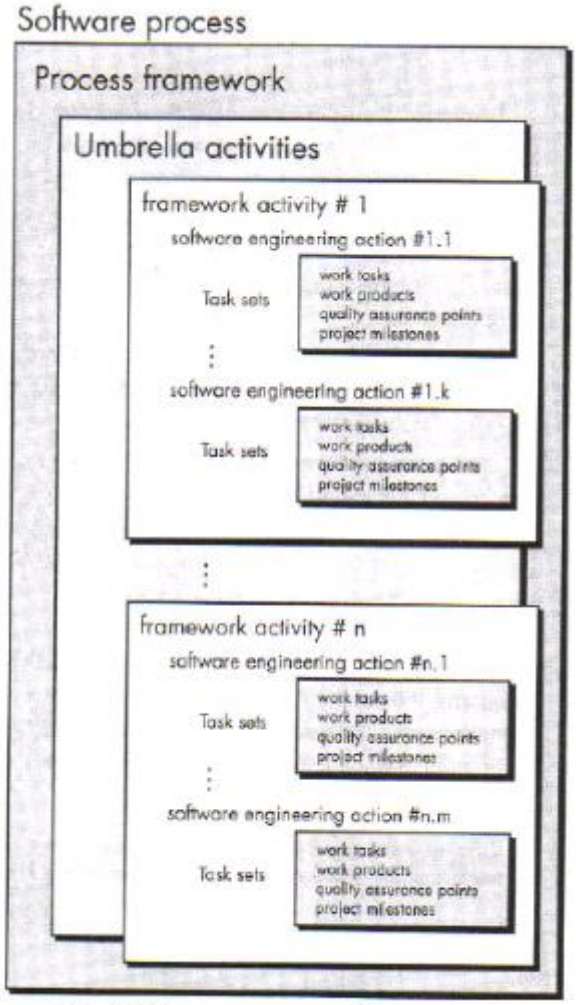


Fig. Software process framework

- The following generic process (used as a basis for the description of process models in subsequent chapters) is applicable to the vast majority of software projects:

1. Communication

This framework activity involves heavy communication and collaboration with the customer (and other stakeholders) and encompasses requirements gathering and other related activities.

2. Planning

This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

3. Modelling

This activity encompasses the creation of models the developer and the customer to better understand software requirements and the design that will achieve those requirements.

4. Construction

This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.

5. Deployment

The software (as a complete entity or as a partially completed increment) is delivered to the customers who evaluates the delivered product and provide feedback based on the evaluation.

UMBRELLA ACTIVITIES:

The following are the set of Umbrella Activities.

1) Software project tracking and control – allows the software team to assess progress against the project plan and take necessary action to maintain schedule.

2) Risk Management - assesses risks that may effect the outcome of the project or the quality of the product.

3) Software Quality Assurance - defines and conducts the activities required to ensure software quality.

4) Formal Technical Reviews - assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next action or activity.

5) Measurement - define and collects process, project and product measures that assist the team in delivering software that needs customer's needs, can be used in conjunction with all other framework and umbrella activities.

6) Software configuration management - manages the effects of change throughout the software process.

7) Reusability management - defines criteria for work product reuse and establishes mechanisms to achieve reusable components.

8) Work Product preparation and production - encompasses the activities required to create work products such as models, document, logs, forms and lists.

Software Engineering Institute Capability Maturity Model (SEICMM) / CMMI

The Capability Maturity Model (CMM) is a procedure used to develop and refine an organization's software development process.

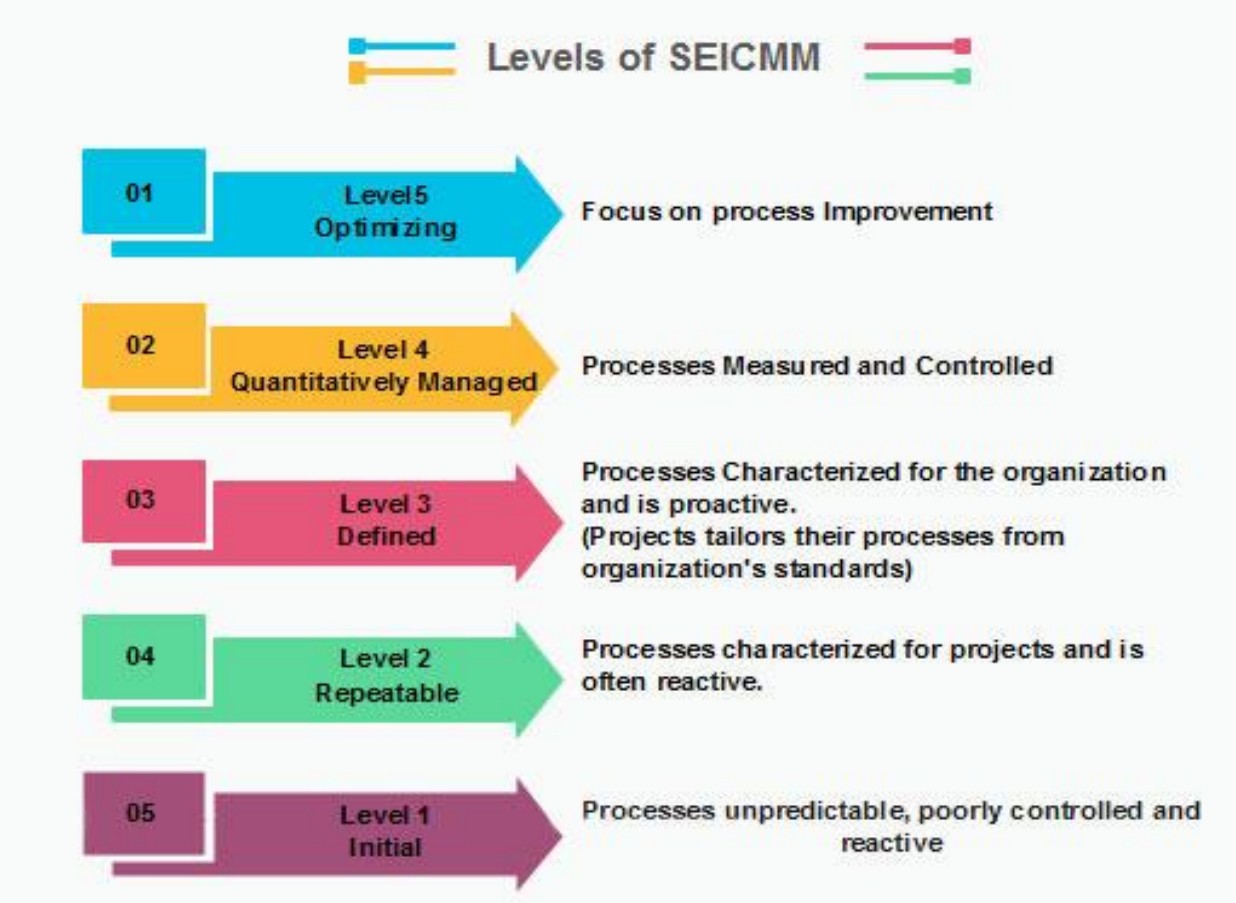
The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes.

CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center promote by the U.S. Department of Defense (DOD).

Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.

In CMMI models with a staged representation, there are five maturity levels designated by the numbers 1 through 5

1. Initial
2. Repeatable
3. Defined
4. Quantitatively Managed
5. Optimizing



Level 1: Initial

At maturity level 1, processes are usually ad hoc. The organization usually does not provide a stable environment. Success in these organizations depends on the competence and heroics of the people in the organization and not on the use of proven processes.

Maturity level 1 organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects.

Maturity level 1 organizations are characterized by a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes.

Level 2: Repeatable

At this level, the fundamental project management practices like tracking cost and schedule are established. maturity level 2 helps to ensure that existing practices are retained during times of stress. When

these practices are in place, projects are performed and managed according to their documented plans.

Level 3: Defined

At maturity level 3, an organization has achieved all the **specific** and **generic goals** of the process areas assigned to maturity levels 2 and 3.

At maturity level 3, processes are well characterized and understood, and are described in standards, procedures, tools, and methods.

At this level, the methods for both management and development activities are defined and documented. There is a common organization-wide understanding of operations, roles, and responsibilities. The ways through defined, the process and product qualities are not measured.

Level 4: Managed

At maturity level 4, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, and 4 and the **generic goals** assigned to maturity levels 2 and 3.

At maturity level 4 Subprocesses are selected that significantly contribute to overall process performance. These selected subprocesses are controlled using statistical and other quantitative techniques.

A critical distinction between maturity level 3 and maturity level 4 is the predictability of process performance. At maturity level 4, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable. At maturity level 3, processes are only qualitatively predictable.

Level 5: Optimizing

At maturity level 5, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, 4, and 5 and the **generic goals** assigned to maturity levels 2 and 3.

Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes.

Maturity level 5 focuses on continually improving process performance through both incremental and innovative technological improvements.

Quantitative process-improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement.

The Process and Product

Process:

Process is a set of sequence steps that have to be followed to create a project. The main purpose of a process is to improve the quality of the project. The process serves as a template that can be used through the creation of its examples and is used to direct the project.

Product:

In the context of software engineering, Product includes any software manufactured based on the customer's request. This can be a problem solving software or computer based system. It can also be said that this is the result of a project.

Let’s see the difference between Product and Process:-

S.NO	Product	Process
1.	Product is the final production of the project.	While the process is a set of sequence steps that have to be followed to create a project.
2.	A product focuses on the final result.	Whereas the process is focused on completing each step being developed.
3.	In the case of products, the firm guidelines are followed.	In contrast, the process consistently follows guidelines.
4.	A product tends to be short-term.	Whereas the process tends to be long-term.
5.	The main goal of the product is to complete the work successfully.	While the purpose of the process is to make the quality of the project better.

software process models

Software processes are the activities for designing, implementing, and testing a software system.

A software process model is an abstraction of the software development process. The models specify the stages and order of a process. So, think of this as a representation of the **order of activities** of the process and the **sequence** in which they are performed.

***NOTE* The goal of a software process model is to provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible.**

There are many kinds of process models for meeting different requirements. We refer to these as **SDLC models** (Software Development Life Cycle models). The most popular and important SDLC models are as follows:

- Waterfall model
- V model
- Incremental model
- RAD model
- Spiral model

Waterfall Model

The waterfall model is a **sequential, plan driven-process** where you must plan and schedule all your activities before starting the project. Each activity in the waterfall model is represented as a separate phase arranged in linear order.

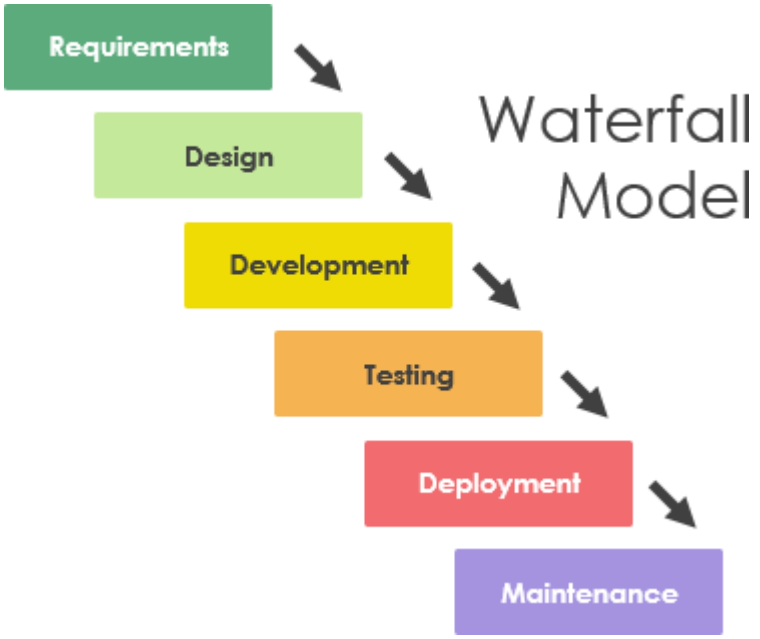
This model has five phases:

- Requirements
- Design
- Implementation
- Testing
- Deployment
- Maintenance

The steps always follow in this order and do not overlap. The developer must complete every phase before the next phase begins. This model is named "**Waterfall Model**", because its diagrammatic representation resembles a cascade of waterfalls.

The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of

the previous one and corresponds to a specialisation of tasks. The approach is typical for certain areas of engineering design.



The waterfall model is easy to understand and follow. It doesn't require a lot of customer involvement after the specification is done. Since it's inflexible, it can't adapt to changes. There is no way to see or try the software until the last phase.

The waterfall model has a rigid structure, so it should be used in cases where the requirements are understood completely and unlikely to radically change.

1.Requirements analysis and specification phase:

The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. It describes the "what" of the system to be produced and not "how."In this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.

2. Design Phase:

This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

3. Implementation and unit testing:

During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD. During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

4. Integration and System Testing: This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

5. Operation and maintenance phase: Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

When to use SDLC Waterfall Model?

Some Circumstances where the use of the Waterfall model is most suited are:

- When the requirements are constant and not changed regularly.
- A project is short
- The situation is calm
- Where the tools and technology used is consistent and is not changing

- When resources are well prepared and are available to use.

Advantages of Waterfall model

- This model is simple to implement also the number of resources that are required for it is minimal.
- The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
- The start and end points for each phase is fixed, which makes it easy to cover progress.
- The release date for the complete product, as well as its final cost, can be determined before development.
- It gives easy to control and clarity for the customer due to a strict reporting system.

Disadvantages of Waterfall model

- In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
- This model cannot accept the changes in requirements during development.
- It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
- Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

INCREMENTAL PROCESS MODELS:

i) The incremental model

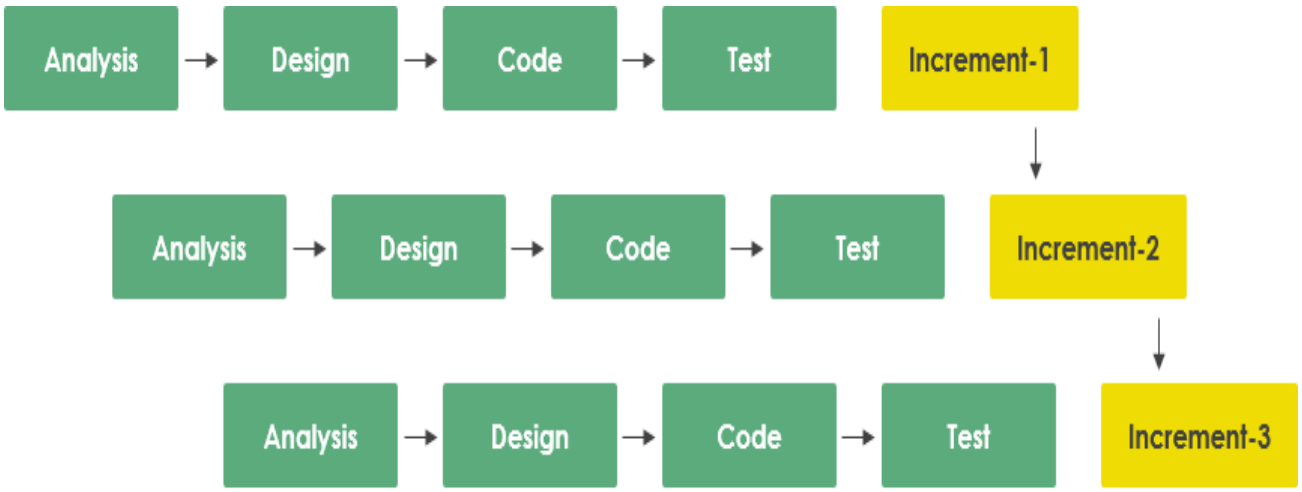
ii) The RAD model

(i) Incremental Model

The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished.

In this 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.

It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements. Each iteration passes through the requirements, design, coding and testing phases.



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. 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

4. 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.

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.

(ii) RAD (Rapid Application Development) Model

Rapid Application Development (RAD) is an incremental software process model that emphasizes a short development cycle.

The RAD model is a "high-speed" adaption of the waterfall model, in which rapid development is achieved by using a component base construction approach.

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)

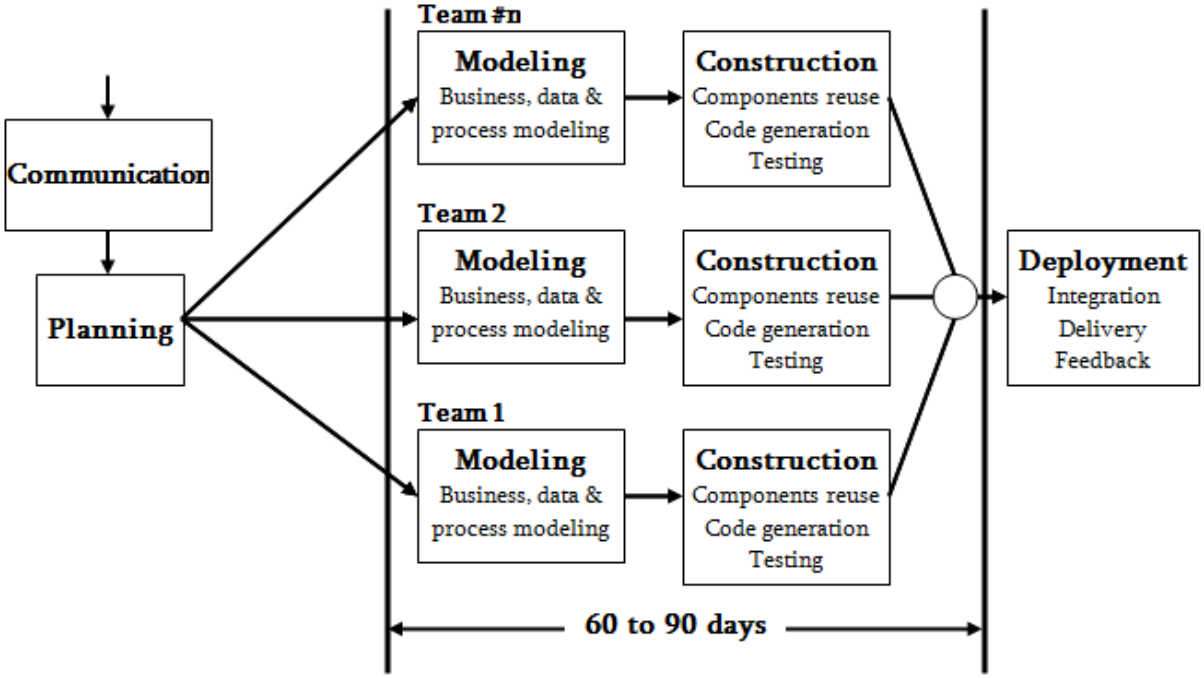
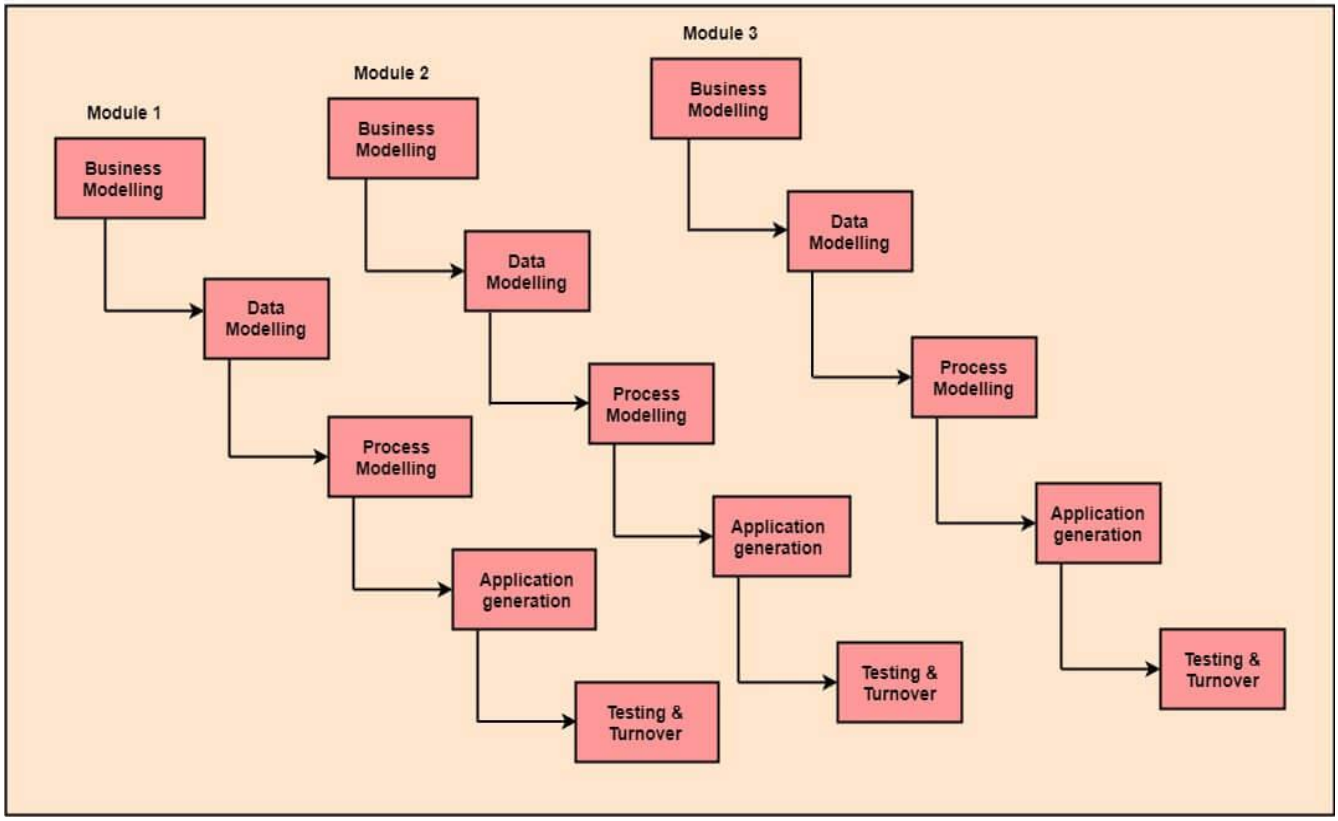


Figure : Flowchart of RAD model

Fig: RAD Model



The RAD approach maps into the generic framework activities:

Communication:

works to understand the business problem and the information characteristics that the software must accommodate.

Planning:

Is essential because multiple software teams works in parallel on different system functions.

Modeling:

encompasses three major phases- business modeling, data modelling and process modeling- and establishes design representation that serve existing software components and the application of automatic code generation.

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.

Construction:

Application generation:

RAD assumes the use of fourth generation techniques i.e, automated tools are used to facilitate construction of the software.

Testing :

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.

Deployment:

Establishes a basis for subsequent iterations**.**

When to use RAD Model?

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known.
- When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

Advantage of RAD Model

- This model is flexible for change.
- In this model, changes are adoptable.
- Each phase in RAD brings highest priority functionality to the customer.
- It reduced development time.
- It increases the reusability of features.

Disadvantage of RAD Model

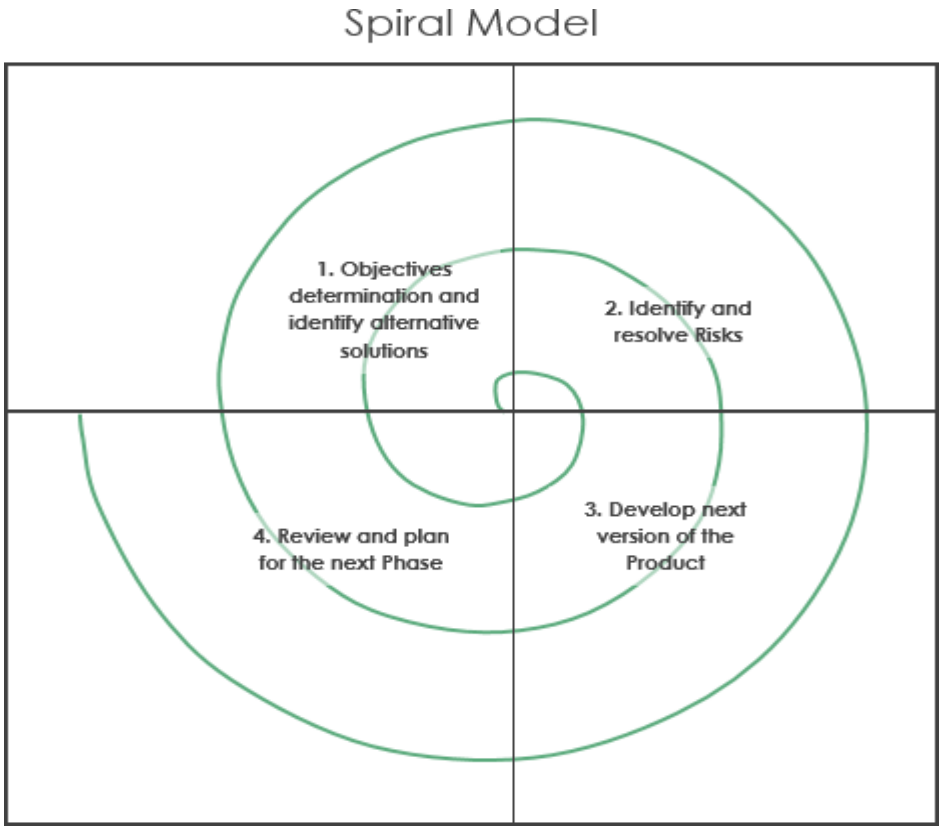
- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.
- On the high technical risk, it's not suitable.
- Required user involvement.

Spiral model

The spiral model, first described by Barry Boehm in 1986, is a risk-driven software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model. A spiral model looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. This model supports risk handling, and the project is delivered in

loops. Each loop of the spiral is called a Phase of the software development process.

The initial phase of the spiral model in the early stages of Waterfall Life Cycle that is needed to develop a software product. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using a spiral model.



Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.

2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

When to use Spiral Model?

- When deliverance is required to be frequent.
- When the project is large
- When requirements are unclear and complex
- When changes may require at any time
- Large and high budget projects

Advantages

- High amount of risk analysis
- Useful for large and mission-critical projects.

Disadvantages

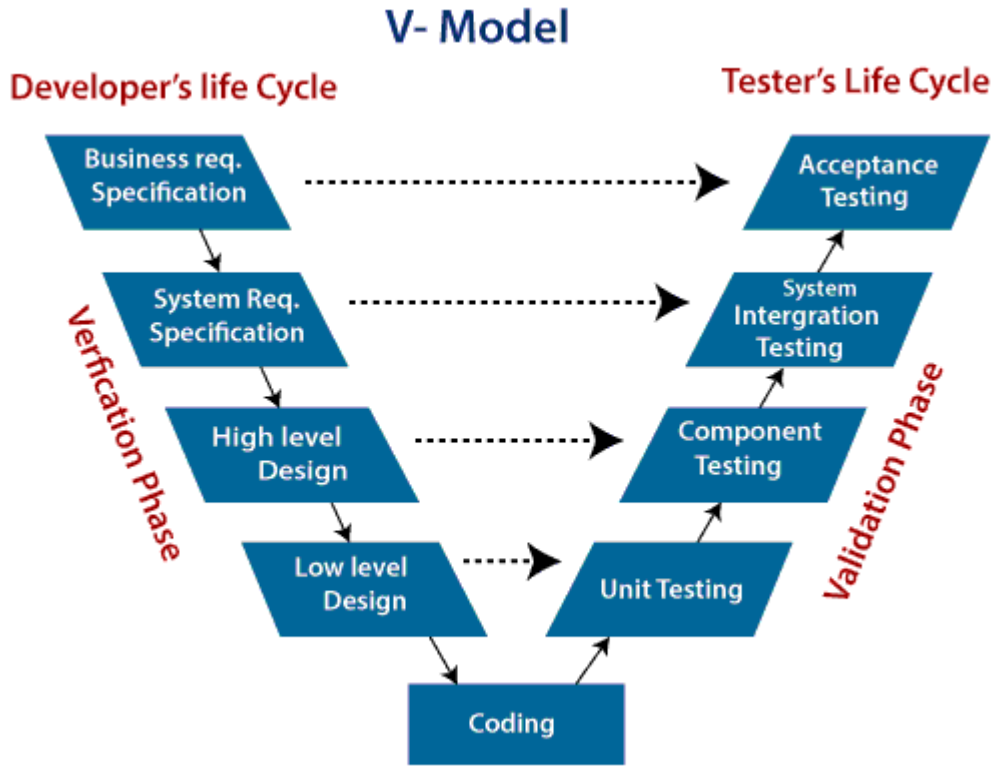
- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.

V -Model

The V-model represents a development process that may be considered an extension of the waterfall model and is an example of the more general V-model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the

typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.



Verification: It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

Validation: It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process

to determine whether the software meets the customer expectations and requirements.

So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation process is joined by coding phase in V-shape. Thus it is known as V-Model.

There are the various phases of Verification Phase of V-model:

1. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

There are the various phases of Validation Phase of V-model:

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is

the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.

2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client?s business team. System Test ensures that expectations from an application developer are met.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

When to use V-Model?

- When the requirement is well defined and not ambiguous.
- The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Advantage (Pros) of V-Model:

1. Easy to Understand.
2. Testing Methods like planning, test designing happens well before coding.
3. This saves a lot of time. Hence a higher chance of success over the waterfall model.
4. Avoids the downward flow of the defects.
5. Works well for small plans where requirements are easily understood.

Disadvantage (Cons) of V-Model:

1. Very rigid and least flexible.
2. Not a good for a complex project.
3. Software is developed during the implementation stage, so no early prototypes of the software are produced.
4. If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

The Component Base Development Software Process Model

The component-based development (CBD) model incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software. However, the component-based development model composes applications from prepackaged software components, called classes.

Modeling and construction activities begin with the identification of candidate components. Regardless of the technology that has been used to create the components, the model incorporates the following steps:

1. Available component-based products are researched and evaluated for the application domain in question.
2. Component integration issues are considered.
3. A software architecture is designed to accommodate the components.
4. Components are integrated into the architecture.
5. Comprehensive testing is conducted to ensure proper functionality.

The component-based development model leads to software reuse, and re-usability provides software engineers with a number of measurable benefits.

Advantages:

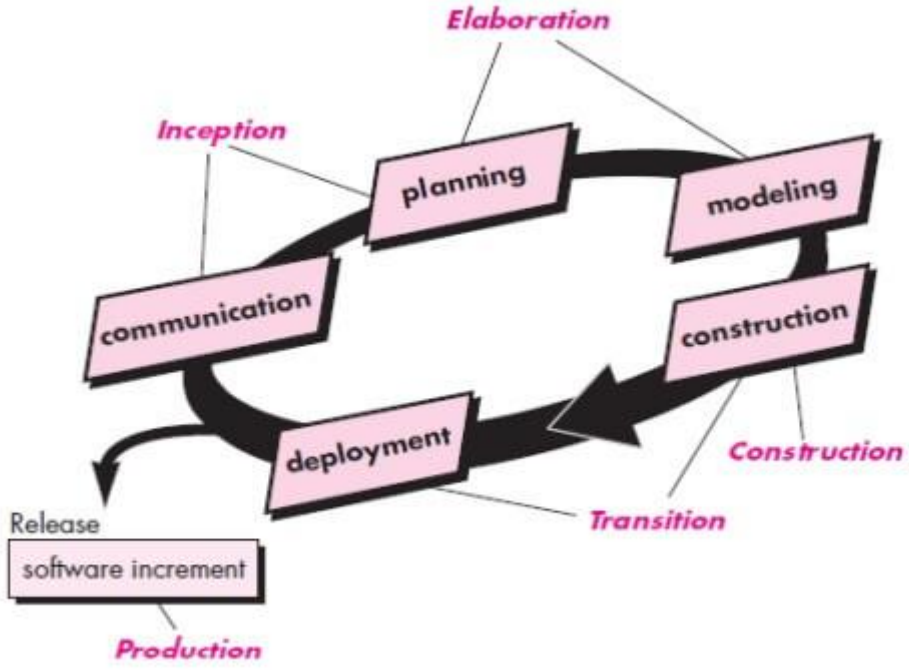
1. Software reuse

2. can achieve a reduction in development cycle time

3. Reduction in project cost

The Unified Process in Software Engineering

The Unified Process or Rational Unified Process (RUP) is a framework for object oriented software engineering using UML. This is a use-case driven, architecture centric, iterative and incremental software development model.



The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system. It emphasizes the important role of software

architecture and “helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse”.

It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

Phases of the Unified Process

This process divides the development process into five phases:

- Inception
- Elaboration
- Conception
- Transition
- Production

Inception Phase

The Inception Phase of UP includes both customer communication and planning activities. By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed, and a plan for the iterative, incremental nature of the project is developed.

Elaboration Phase

The Elaboration Phase encompasses the planning and modeling activities of the generic process model. Elaboration refines and expands the primary use-cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software – the use-case model, the analysis model, the design model, the implementation model and the deployment model.

Construction Phase

The construction phase of the UP is identical to the construction activity defined in the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each usecase operational for end-users. As components are developed unit tests are designed and executed for each component. Integration testing and acceptance testing are carried out using use-cases to derive required test cases.

Transition Phase

The Transition Phase of the UP encompasses the later stages of the generic construction activity and the first part of the generic deployment activity. Software is given to end-users for beta testing. The software team creates necessary support information (user manual, installation manual etc.). At the end of transition phase, the software increment becomes a usable software release.

Production Phase

The production phase of the UP coincides with the deployment activity of the generic process. During this phase, the on going use of the software is monitored, support for operating environment is provided, and defect reports and requests for change are submitted and evaluated.

It is likely that at the same time the construction, transition and production phases are being conducted, work may have already begun on the next software increment. This means that the unified process do not occur in a sequence, but rather with staggered concurrency.

Advantages of Unified Process Model

- It cover the complete Software Development Life Cycle.
- The best Practices for Software development are Supported by unified process model.
- It encourage designing in UML.

Disadvantages of Unified Process Model

- Could be more complex to implement.
- Heavy Weight Process.
- Expert are need for it.
- Risk can’t be determined.

The End