# UNIT 1

**UNIT 1 SOFTWARE PROCESS AND PROJECT MANAGEMENT** **9 Hrs.**

The evolving role of software – the changing nature of software- Life cycle models - Water fall - Incremental - Spiral - Evolutionary - Prototyping – Concurrent development – Specialised process models - Verification - Validation - Life cycle process - Development process - System engineering hierarchy - Introduction to CMM - Levels of CMM.

## Evolving Role of Software:

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.

**Fritz Bauer, a German computer scientist, defines software engineering as:**

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

## Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as software evolution. This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements,

# UNIT 1

Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

## Software Evolution Laws

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

- **S-type (static-type) -** This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.
- **P-type (practical-type) -** This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.
- **E-type (embedded-type) -** This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

## E-Type software evolution

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

- **Continuing change -** An E-type software system must continue to adapt to the real world changes, else it becomes progressively less useful.
- **Increasing complexity -** As an E-type software system evolves, its complexity tends to increase unless work is done to maintain or reduce it.

- **Conservation of familiarity -** The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc. must be retained at any cost, to implement the changes in the system.
- **Continuing growth-** In order for an E-type system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business.
- **Reducing quality -** An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment.
- **Feedback systems-** The E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.
- **Self-regulation -** E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
- **Organizational stability -** The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.

## Software Paradigms

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another:

Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

## Software Development Paradigm:

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirement gathering
- Software design
- Programming

# UNIT 1

## Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

## Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes –

- Coding
- Testing
- Integration

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 he 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.

# UNIT 1

## 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 Development Lifecycle (Sdlc)

It is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software that meets customer expectations. The system development should be complete in the pre-defined time frame and cost. SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC life cycle has its own process and deliverables that feed into the next phase. SDLC stands for Software Development Lifecycle.

Here, are prime reasons why SDLC is important for developing a software system.

- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

## SDLC Phases

The entire SDLC process divided into the following stages:


- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study:
- Phase 3: Design:

- Phase 4: Coding:
- Phase 5: Testing:
- Phase 6: Installation/Deployment:
- Phase 7: Maintenance:

In this tutorial, I have explained all these phases

**Phase 1: Requirement collection and analysis:**

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

**Phase 2: Feasibility study:**

Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

**There are mainly five types of feasibilities checks:**

- **Economic:** Can we complete the project within the budget or not?
- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software
- **Schedule:** Decide that the project can be completed within the given schedule or not.

**Phase 3: Design:**

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design(LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

**Phase 4: Coding:**

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

**Phase 5: Testing:**

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

**Phase 6: Installation/Deployment:**

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

**Phase 7: Maintenance:**

Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing - bugs are reported because of some scenarios which are not tested at all
- Upgrade - Upgrading the application to the newer versions of the Software
- Enhancement - Adding some new features into the existing software

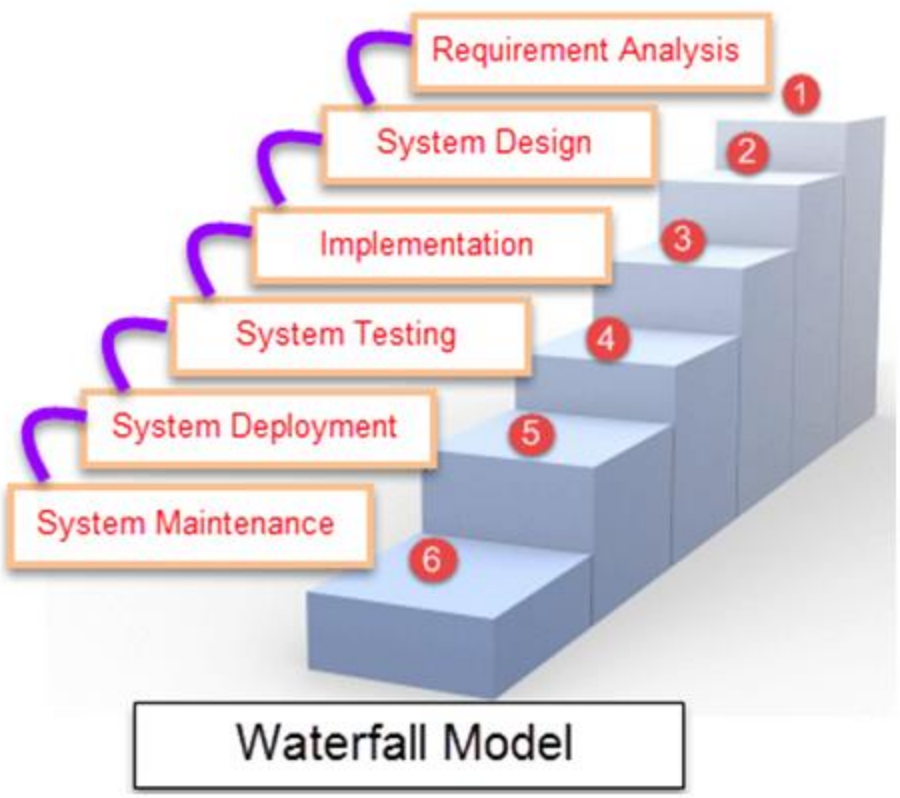
The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

**Waterfall model**

The waterfall is a widely accepted SDLC model. In this approach, the whole process of the software development is divided into various phases. In this SDLC model, the outcome of one phase acts as the input for the next phase.

This SDLC model is documentation-intensive, with earlier phases documenting what need be performed in the subsequent phases.

# UNIT 1




## Different Phases of Waterfall Model in Software Engineering

| Different phases | Activities performed in each stage |
|---|---|
| Requirement Gathering stage | • During this phase, detailed requirements of the software system to be developed are gathered from client |
| Design Stage | • Plan the programming language, for Example Java, PHP, .net<br>• or database like Oracle, MySQL, etc.<br>• Or other high-level technical details of the project |
| Built Stage | • After design stage, it is built stage, that is nothing but coding the software |
| Test Stage | • In this phase, you test the software to verify that it is built as per the specifications given by the client. |

| | |
|---|---|
| **Deployment stage** | • Deploy the application in the respective environment |
| **Maintenance stage** | • Once your system is ready to use, you may later require change the code as per customer request |

Waterfall model can be used when

- Requirements are not changing frequently
- Application is not complicated and big
- Project is short
- Requirement is clear
- Environment is stable
- Technology and tools used are not dynamic and is stable
- Resources are available and trained

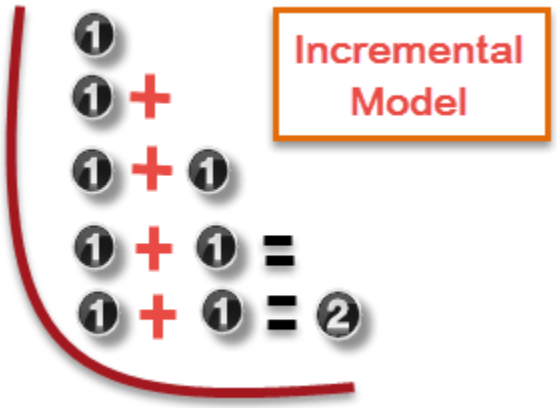
**Advantages and Disadvantages of Waterfall-Model**

| Advantages | Dis-Advantages |
|---|---|
| • Before the next phase of development, each phase must be completed | • Error can be fixed only during the phase |
| • Suited for smaller projects where requirements are well defined | • It is not desirable for complex project where requirement changes frequently |
| • They should perform quality assurance test (Verification and Validation) before completing each stage | • Testing period comes quite late in the developmental process |

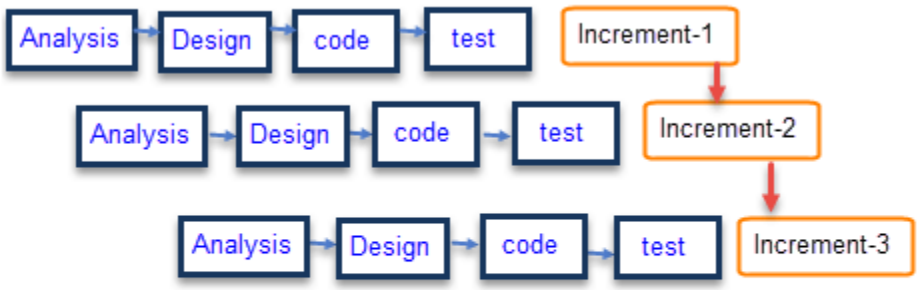
## Incremental Approach

The incremental model is not a separate model. It is essentially a series of waterfall cycles. The requirements are divided into groups at the start of the project. For

each group, the SDLC model is followed to develop software. The SDLC process is repeated, with each release adding more functionality until all requirements are met. In this method, every cycle act as the maintenance phase for the previous software release. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.




Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.




Incremental Model

The system is put into production when the first increment is delivered. The first increment is often a core product where the basic requirements are addressed, and supplementary features are added in the next increments. Once the core product is analyzed by the client, there is plan development for the next increment.

# UNIT 1

## Characteristics of an Incremental module includes

- System development is broken down into many mini development projects
- Partial systems are successively built to produce a final total system
- Highest priority requirement is tackled first
- Once the requirement is developed, requirement for that increment are frozen

| Incremental Phases | Activities performed in incremental phases |
|---|---|
| Requirement Analysis | • Requirement and specification of the software are collected |
| Design | • Some high-end function are designed during this stage |
| Code | • Coding of software is done during this stage |
| Test | • Once the system is deployed, it goes through the testing phase |

## When to use Incremental models?

- Requirements of the system are clearly understood
- When demand for an early release of a product arises
- When software engineering team are not very well skilled or trained
- When high-risk features and goals are involved
- Such methodology is more in use for web application and product based companies

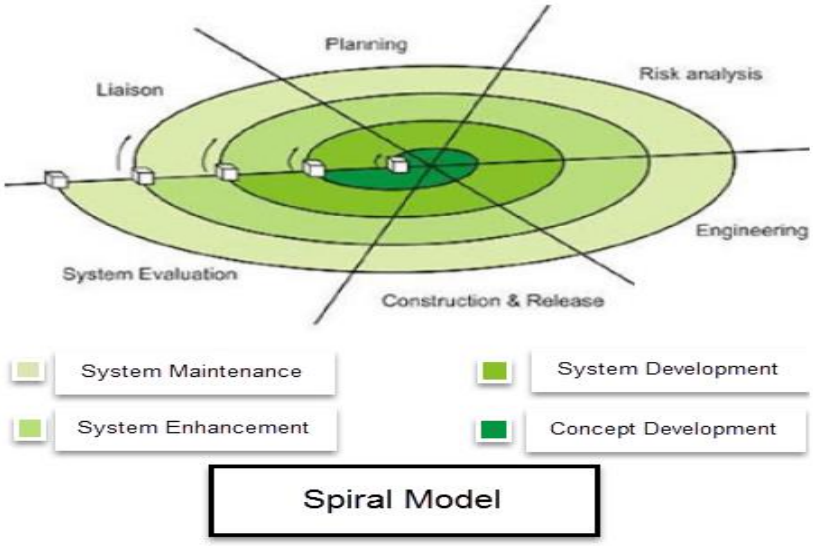
## Advantages and Disadvantages of Incremental Model

| Advantages | Disadvantages |
|---|---|
| • The software will be generated quickly during the software life cycle | • It requires a good planning designing |
| • It is flexible and less expensive to change requirements and scope | • Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle |

## Spiral Model

The spiral model is a risk-driven process model. This SDLC model helps the team to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc.

This model adopts the best features of the prototyping model and the waterfall model. The spiral methodology is a combination of rapid prototyping and concurrency in design and development activities.



## Spiral Model Phases

| Spiral Model Phases | Activities performed during phase |
|---|---|
| Planning | • It includes estimating the cost, schedule and resources for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer |
| Risk Analysis | • Identification of potential risk is done while risk mitigation strategy is planned and finalized |
| Engineering | • It includes testing, coding and deploying software at the customer site |
| Evaluation | • Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost overrun |

### Use of Spiral Methodology:

- When project is large
- When releases are required to be frequent
- When creation of a prototype is applicable
- When risk and costs evaluation is important
- For medium to high-risk projects
- When requirements are unclear and complex
- When changes may require at any time
- When long term project commitment is not feasible due to changes in economic priorities

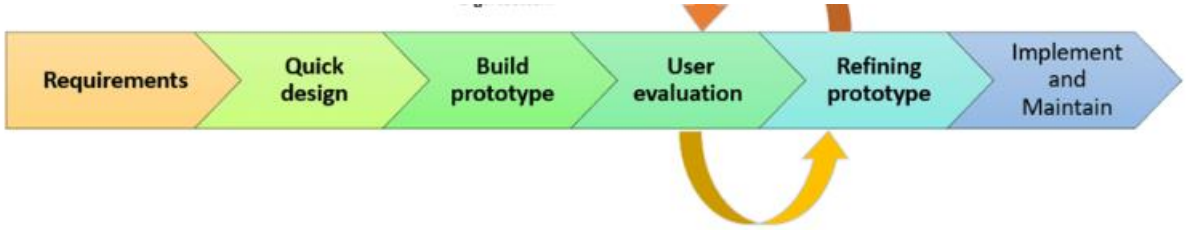
## Advantages and Disadvantages of Spiral Model

| Advantages | Disadvantages |
|---|---|
| • Additional functionality or changes can be done at a later stage | • Risk of not meeting the schedule or budget |
| • Cost estimation becomes easy as the prototype building is done in small fragments | • It works best for large projects only also demands risk assessment expertise |
| • Continuous or repeated development helps in risk management | • For its smooth operation spiral model protocol needs to be followed strictly |
| • Development is fast and features are added in a systematic way | • Documentation is more as it has intermediate phases |

## Software Prototyping Model

Prototype methodology is defined as a Software Development model in which a prototype is built, test, and then reworked when needed until an acceptable prototype is achieved. It also creates a base to produce the final system.

Software prototyping model works best in scenarios where the project's requirement are not known. It is an iterative, trial, and error method which take place between the developer and the client.




Prototyping Model has following six SDLC phases as follow:

# UNIT 1

## Step 1: Requirements gathering and analysis

A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

## Step 2: Quick design

The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

## Step 3: Build a Prototype

In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system.

## Step 4: Initial user evaluation

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

## Step 5: Refining prototype

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

## Step 6: Implement Product and Maintain

Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

# UNIT 1

## Types of Prototyping Models

Four types of Prototyping models are:

1. Rapid Throwaway prototypes
2. Evolutionary prototype
3. Incremental prototype
4. Extreme prototype

## Rapid Throwaway Prototype

Rapid throwaway is based on the preliminary requirement. It is quickly developed to show how the requirement will look visually. The customer's feedback helps drives changes to the requirement, and the prototype is again created until the requirement is baselined.

In this method, a developed prototype will be discarded and will not be a part of the ultimately accepted prototype. This technique is useful for exploring ideas and getting instant feedback for customer requirements.

## Evolutionary Prototyping

Here, the prototype developed is incrementally refined based on customer's feedback until it is finally accepted. It helps you to save time as well as effort. That's because developing a prototype from scratch for every interaction of the process can sometimes be very frustrating.

This model is helpful for a project which uses a new technology that is not well understood. It is also used for a complex project where every functionality must be checked once. It is helpful when the requirement is not stable or not understood clearly at the initial stage.

## Incremental Prototyping

In incremental Prototyping, the final product is decimated into different small prototypes and developed individually. Eventually, the different prototypes are merged into a single product. This method is helpful to reduce the feedback time between the user and the application development team.

## Extreme Prototyping:

# UNIT 1

Extreme prototyping method is mostly used for web development. It is consists of three sequential phases.

1. Basic prototype with all the existing page is present in the HTML format.
2. You can simulate data process using a prototype services layer.
3. The services are implemented and integrated into the final prototype.

## Best practices of Prototyping

Here, are a few things which you should watch for during the prototyping process:

- You should use Prototyping when the requirements are unclear
- It is important to perform planned and controlled Prototyping.
- Regular meetings are vital to keep the project on time and avoid costly delays.
- The users and the designers should be aware of the prototyping issues and pitfalls.
- At a very early stage, you need to approve a prototype and only then allow the team to move to the next step.
- In software prototyping method, you should never be afraid to change earlier decisions if new ideas need to be deployed.
- You should select the appropriate step size for each version.
- Implement important features early on so that if you run out of the time, you still have a worthwhile system

## Advantages of the Prototyping Model

Here, are important pros/benefits of using Prototyping models:

- Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process.
- Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity.
- Helps team member to communicate effectively
- Customer satisfaction exists because the customer can feel the product at a very early stage.
- There will be hardly any chance of software rejection.
- Quicker user feedback helps you to achieve better software development solutions.

- Allows the client to compare if the software code matches the software specification.
- It helps you to find out the missing functionality in the system.
- It also identifies the complex or difficult functions.
- Encourages innovation and flexible designing.
- It is a straightforward model, so it is easy to understand.
- No need for specialized experts to build the model
- The prototype serves as a basis for deriving a system specification.
- The prototype helps to gain a better understanding of the customer's needs.
- Prototypes can be changed and even discarded.
- A prototype also serves as the basis for operational specifications.
- Prototypes may offer early training for future users of the software system.

## Disadvantages of the Prototyping Model

Here, are important cons/drawbacks of prototyping model:

- Prototyping is a slow and time taking process.
- The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.
- Prototyping may encourage excessive change requests.
- Some times customers may not be willing to participate in the iteration cycle for the longer time duration.
- There may be far too many variations in software requirements when each time the prototype is evaluated by the customer.
- Poor documentation because the requirements of the customers are changing.
- It is very difficult for software developers to accommodate all the changes demanded by the clients.
- After seeing an early prototype model, the customers may think that the actual product will be delivered to him soon.
- The client may lose interest in the final product when he or she is not happy with the initial prototype.
- Developers who want to build prototypes quickly may end up building sub-standard development solutions.

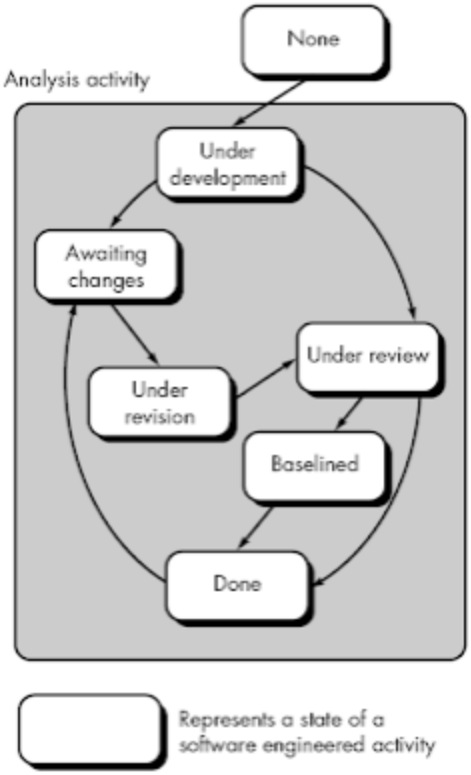
# UNIT 1

## Concurrent Development:

The concurrent development model, sometimes called concurrent engineering, has been described in the following manner by Davis and Sitaram :

Project managers who track project status in terms of the major phases [of the classic life cycle] have no idea of the status of their projects. These are examples of trying to track extremely complex sets of activities using overly simple models. Note that although . . . [a large] project is in the coding phase, there are personnel on the project involved in activities typically associated with many phases of development simultaneously. For example, . . personnel are writing requirements, designing, coding, testing, and integration testing [all at the same time]. Software engineering process models by Humphrey and Kellner have shown the concurrency that exists for activities occurring during any one phase. Kellner's more recent work uses statecharts [a notation that represents the states of a process] to represent the concurrent relationship existent among activities associated with a specific event (e.g., a requirements change during late development), but fails to capture the richness of concurrency that exists across all software development and management activities in the project. . . . Most software development process models are driven by time; the later it is, the later in the development process you are. [A concurrent process model] is driven by user needs, management decisions, and review results.

The concurrent process model can be represented schematically as a series of major technical activities, tasks, and their associated states. For example, the engineering activity defined for the spiral model is accomplished by invoking the following tasks: prototyping and/or analysis modeling, requirements specification, and design.




The activity—analysis—may be in any one of the states noted at any given time. Similarly, other activities (e.g., design or customer communication) can be represented in an analogous manner. All activities exist concurrently but reside in different states. For example, early in a project the customer communication activity (not shown in the figure) has completed its first iteration and exists in the awaiting changes state. The analysis activity (which existed in the none state while initial customer communication was completed) now makes a transition into the under development state. If, however, the customer indicates that changes in requirements must be made, the analysis activity moves from the under development state into the awaiting changes state.

The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities. For example,during early stages of design, an inconsistency in the analysis model is uncovered. This generates the event analysis model correction which will trigger the analysis activity from the done state into the awaiting changes state.

The concurrent process model is often used as the paradigm for the development of client/server applications. A client/server system is composed of a set of functional

components. When applied to client/server, the concurrent process model defines activities in two dimensions : a system dimension and a component dimension. System level issues are addressed using three activities: design, assembly, and use. The component dimension is addressed with two activities: design and realization.

Concurrency is achieved in two ways:

(1) system and component activities occur simultaneously and can be modeled using the state-oriented approach described previously;

(2) a typical client/server application is implemented with many components, each of which can be designed and realized concurrently.

In reality, the concurrent process model is applicable to all types of software development and provides an accurate picture of the current state of a project. Rather thanconfining software engineering activities to a sequence of events, it defines a network of activities. Each activity on the network exists simultaneously with other activities. Events generated within a given activity or at some other place in the activity network trigger transitions among the states of an activity.

**Specialized Process Models.**

Special process models take on many of the characteristics of one or more of the conventional models. However, specialized models tend to be applied when a narrowly defined software engineering approach is chosen.

**Component-Based Development.**

Commercial off-the-shelf (COST) software components, developed by vendors who offer them as products, can be used when software is to be built. These components provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software. The component-based development 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 model composes applications from prepackaged software components. Modeling and construction activities begin with the identification of candidate components. These components can be designed as either conventional software modules or object-oriented classes or packages of classes. Regardless of the technology that is used to create the components, the component-based development model incorporates the following steps (implemented using an evolutionary approach):

• Available component-based products are researched and evaluated for the application domain in question. http://wikistudent.ws/Unisa

• Component integration issues are considered.

• A software architecture is designed to accommodate the components.

• Components are integrated into the architecture.

**Comprehensive testing** is conducted to ensure proper functionality. The component-based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefits. Based on studies of reusability component-based development can lead to reduction in development cycle time, reduction in project cost and increase in productivity. Although these results are a function of the robustness of the component library, there is little question that the component-based development model provides significant advantages for software engineers.

**The Formal Methods Model.**

The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software. Formal methods enable a software engineer to specify, develop, and verify a computer-based system by applying rigorous, mathematical notation. A variation on this approach is called clean-room software engineering. When formal methods are used during development, they provide a mechanism for eliminating many of the problems that are difficult to overcome using other software engineering paradigms. Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily, not through ad hoc review, but through the application of mathematical analysis. When formal methods are used during design, they serve as a basis for program verification and therefore enable the software engineer to discover and correct errors that might otherwise go undetected. Although not a mainstream approach, the formal methods model offers the promise of defect-free software. Yet, concern about its applicability in a business environment has been voiced:

• The development of formal models is currently quite time-consuming and expensive.

• Because few software developers have the necessary background to apply formal methods, extensive training is required.

• It is difficult to use the model as a communication mechanism for technically unsophisticated customers.

## Aspect-Oriented Software Development:

Regardless of the software process that is chosen, the builders of complex software invariably implement a set of localized features, functions and information content. These localized software characteristics are modeled as components and then constructed within the context of a system architecture. As modern computer-based systems become more sophisticated and complex, certain concerns, customer required properties or areas of technical interest, span the entire architecture. Some concerns are high-level properties of a system; others affect functions or are systemic. When concerns cut across multiple system functions, features, and information they are often referred to as crosscutting concerns. Aspectual requirements define those crosscutting concerns that have impact across the software architecture. Aspects are mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concerns. Aspect-oriented software development (AOSD), often referred to as aspect-oriented programming (AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects. http://wikistudent.ws/Unisa A distinct aspect-oriented process has not yet matured. However, it is likely that such a process will adopt characteristics of both the spiral and concurrent process models. The evolutionary nature of the spiral is appropriate as aspects are identified and then constructed. The parallel nature of concurrent development is essential because aspects are engineered independently of localized software components and yet, aspects have a direct impact on these components.

## The Unified Process.

In some ways the unified process (UP) is an attempt to draw on the best features and characteristics of conventional software process models, but characterize them in a way that implements many of the best principles of agile software development. 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.

# UNIT 1

## Prescriptive Models or Conventional Models.

Every software engineering organization should describe a unique set of framework activities for the software processes it adopts. It should populate each framework activity with a set of software engineering actions, and define each action in terms of a task set that identifies the work (and work products) to be accomplished to meet the development goals. It should then adapt the resultant process model to accommodate the specific nature of each project, the people who will do the work and the environment in which the work will be conducted. Regardless of the process model that is selected, software engineers have traditionally chosen a generic process framework that encompasses the following framework activities:

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

• **Modeling** – this activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.

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

• **Deployment** – The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation. We call these models prescriptive because they prescribe a set of process elements namely framework activities, software engineering actions, tasks, work products, quality assurance and change control mechanisms for each project. Each process model also prescribes a workflow that is, the manner in which the process elements are interrelated to one another. All software process models can accommodate the generic framework activities, but each applies a different emphasis to these activities and defines a workflow that invokes each framework activity (as well as software engineering actions and tasks) in a different manner.

# UNIT 1

## Verification And Validation:

### Verification:

The verifying process includes checking documents, design, code, and program.

### Validation:

Validation is a dynamic mechanism of Software testing and validates the actual product.

| Verification | Validation |
|---|---|
| • The verifying process includes checking documents, design, code, and program | • It is a dynamic mechanism of testing and validating the actual product |
| • It does ***not*** involve executing the code | • It always involves executing the code |
| • Verification uses methods like reviews, walkthroughs, inspections, and desk-checking etc. | • It uses methods like Black Box Testing, White Box Testing, and non-functional testing |
| • Whether the software conforms to specification is checked | • It checks whether the software meets the requirements and expectations of a customer |
| • It finds bugs early in the development cycle | • It can find bugs that the verification process can not catch |

| | |
|---|---|
| • Target is application and software architecture, specification, complete design, high level, and database design etc. | • Target is an actual product |
| • QA team does verification and make sure that the software is as per the requirement in the SRS document. | • With the involvement of testing team validation is executed on software code. |
| • It comes before validation | • It comes after verification |

**Example of verification and validation**

- In Software Engineering, consider the following specification

***A clickable button with name Submet***

- Verification would check the design doc and correcting the spelling mistake.
- Otherwise, the development team will create a button like

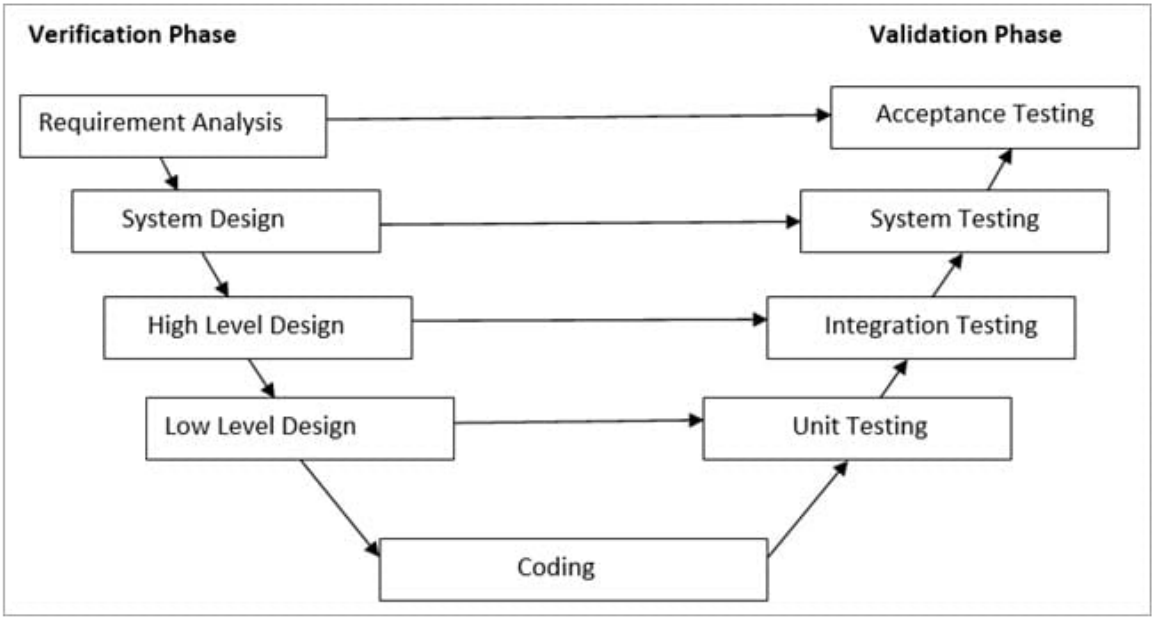

  - So new specification is

***A clickable button with name Submit***

- Once the code is ready, Validation is done. A Validation test found –


- Owing to Validation testing, the development team will make the submit button clickable.

V-Model is also known as Verification and Validation Model. In this model Verification & Validation goes hand in hand i.e. development and testing goes parallel. V model and waterfall model are the same except that the test planning and testing start at an early stage in V-Model.




**a) Verification Phase:**

**(i) Requirement Analysis:**

In this phase, all the required information is gathered & analyzed. Verification activities include reviewing the requirements.

**(ii) System Design:**

Once the requirement is clear, a system is designed i.e. architecture, components of the product are created and documented in a design document.

**(iii) High-Level Design:**

High-level design defines the architecture/design of modules. It defines the functionality between the two modules.

**(iv) Low-Level Design:**

Low-level Design defines the architecture/design of individual components.

**(v) Coding:**

Code development is done in this phase.

**b) Validation Phase:**

**(i) Unit Testing:**

Unit testing is performed using the unit test cases that are designed and is done in the Low-level design phase. Unit testing is performed by the developer itself. It is performed on individual components which lead to early defect detection.

**(ii) Integration Testing:**

Integration testing is performed using integration test cases in High-level Design phase. Integration testing is the testing that is done on integrated modules. It is performed by testers.

**(iii) System Testing:**

System testing is performed in the System Design phase. In this phase, the complete system is tested i.e. the entire system functionality is tested.

**(iv) Acceptance Testing:**

Acceptance testing is associated with the Requirement Analysis phase and is done in the customer's environment.

**Advantages of V – Model:**

- It is a simple and easily understandable model.
- V –model approach is good for smaller projects wherein the requirement is defined and it freezes in the early stage.
- It is a systematic and disciplined model which results in a high-quality product.

**Disadvantages of V-Model:**

- V-shaped model is not good for ongoing projects.
- Requirement change at the later stage would cost too high.

**The System Engineering Hierarchy - System Modeling**

Good system engineering begins with a clear understanding of context - the world view - and then progressively narrows focus until technical details are understood. System engineering encompasses a collection of top-down and bottom-up methods to navigate the hierarchy.

System engineering process begins with a world of view which is refined to focus more fully on a specific domain of interest. Within a specific domain, the need for targeted system elements is analyzed. Finally, the analysis, design, and construction of targeted system element is initiated. Broad context is established at the top of the hierarchy and at the bottom, detailed technical activities are

conducted. It is important for a system engineer narrows the focus of work as one moves downward in the hierarchy.

**System modeling is an important element of system engineering process. System engineering model accomplishes the following:**

- define processes.

- represent behavior of the process.

- define both exogenous and endogenous input to model.

- represent all linkages.

**Some restraining factors that are considered to construct a system model are:**

- Assumptions that reduce number of possible permutations and variations thus enabling a model to reflect the problem in a reasonable manner.

- Simplifications that enable the model to be created in a timely manner.

- Limitations that help to bound the system.

- Constraints that will guide the manner in which the model is created and the approach taken when the model is implemented.

- Preferences that indicate the preferred architecture for all data, functions , and technology.

The resultant system model may call for a completely automated or semi automated or a non automated solution.

**Software engineering lifecycle process:**

**Software Development Lifecycle (Sdlc)**

It is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software that meets customer expectations. The system development should be complete in the pre-defined time frame and cost. SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of

the SDLC life cycle has its own process and deliverables that feed into the next phase. SDLC stands for Software Development Lifecycle.

Here, are prime reasons why SDLC is important for developing a software system.

- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

## SDLC Phases

The entire SDLC process divided into the following stages:


- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study:
- Phase 3: Design:
- Phase 4: Coding:
- Phase 5: Testing:
- Phase 6: Installation/Deployment:
- Phase 7: Maintenance:

In this tutorial, I have explained all these phases

### Phase 1: Requirement collection and analysis:

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

**Phase 2: Feasibility study:**

Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

**There are mainly five types of feasibilities checks:**

- **Economic:** Can we complete the project within the budget or not?
- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software
- **Schedule:** Decide that the project can be completed within the given schedule or not.

**Phase 3: Design:**

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design(LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

**Phase 4: Coding:**

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

**Phase 5: Testing:**

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

**Phase 6: Installation/Deployment:**

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

**Phase 7: Maintenance:**

Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing - bugs are reported because of some scenarios which are not tested at all
- Upgrade - Upgrading the application to the newer versions of the Software
- Enhancement - Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

**Capability Maturity Model:**

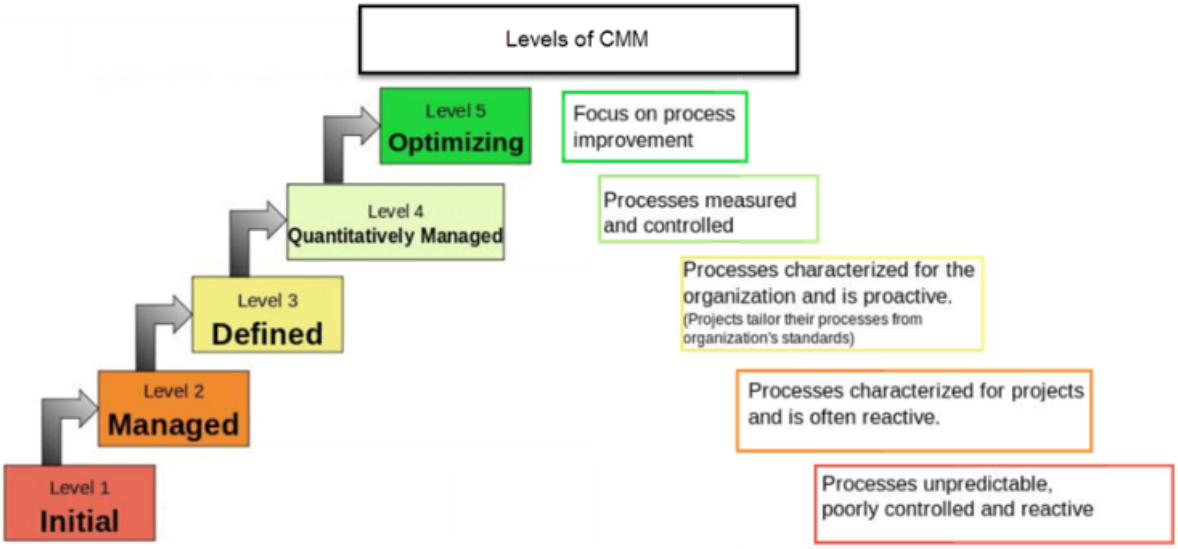
Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.

CMM was developed at the Software engineering institute in the late 80's. It was developed as a result of a study financed by the U.S Air Force as a way to evaluate the work of subcontractors. Later based on the CMM-SW model created in 1991 to assess the maturity of software development, multiple other models are integrated with CMM-I they are

**Capability Maturity Model (CMM) Levels:**

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

# UNIT 1




| Levels | Activities | Benefits |
|---|---|---|
| Level 1 Initial | • At level 1, the process is usually chaotic and ad hoc<br>• A capability is characterized on the basis of the individuals and not of the organization<br>• Progress not measured<br>• Products developed are often schedule and over budget<br>• Wide variations in the schedule, cost, functionality, and quality targets | None. A project is Total Chaos |

| | | |
|---|---|---|
| Level 2 Managed | • Requirement Management<br>• Estimate project parameters like cost, schedule, and functionality<br>• Measure actual progress<br>• Develop plans and process<br>• Software project standards are defined<br>• Identify and control products, problem reports changes, etc.<br>• Processes may differ between projects | • Processes become easier to comprehend<br>• Managers and team members spend less time in explaining how things are done and more time in executing it<br>• Projects are better estimated, better planned and more flexible<br>• Quality is integrated into projects<br>• Costing might be high initially but goes down overtime<br>• Ask more paperwork and documentation |
| Level-3 Defined | • Clarify customer requirements<br>• Solve design requirements, develop an implementation process<br>• Makes sure that product meets the requirements and intended use<br>• Analyze decisions systematically<br>• Rectify and control potential problems | • Process Improvement becomes the standard<br>• Solution progresses from being "coded" to being "engineered"<br>• Quality gates appear throughout the project effort with the entire team involved in the process<br>• Risks are mitigated and don't take the team by surprise |
| Level-4 Quantitatively Managed | • Manages the project's processes and sub-processes statistically<br>• Understand process performance, quantitatively manage the organization's project | • Optimizes Process Performance across the organization<br>• Fosters Quantitative Project Management in an organization. |
| Level-5 Optimizing | • Detect and remove the cause of defects early<br>• Identify and deploy new tools and process improvements to meet needs and business objectives | • Fosters Organizational Innovation and Deployment<br>• Gives impetus to Causal Analysis and Resolution |

## Limitations of CMM Models

- CMM determines what a process should address instead of how it should be implemented
- It does not explain every possibility of software process improvement
- It concentrates on software issues but does not consider strategic business planning, adopting technologies, establishing product line and managing human resources
- It does not tell on what kind of business an organization should be in
- CMM will not be useful in the project having a crisis right now

**Use of CMM:**

Today CMM act as a "seal of approval" in the software industry. It helps in various ways to improve the software quality.

- It guides towards repeatable standard process and hence reduce the learning time on how to get things done
- Practicing CMM means practicing standard protocol for development, which means it not only helps the team to save time but also gives a clear view of what to do and what to expect
- The quality activities gel well with the project rather than thought of as a separate event
- It acts as a commuter between the project and the team
- CMM efforts are always towards the improvement of the process