UNIT-I

INTRODUCTION TO SOFTWARE ENGINEERING

Software:

Software is Instructions (computer programs) that provide desired features, function, and performance,when executed Data structures that enable the programs to adequately manipulate information,Documents that describe the operation and use of the programs.

Characteristics of Software:

Software is developed or engineered; it is not manufactured in the classical sense.Software does not "wear out" Although the industry is moving toward component-based construction, most software continuesto be custom built.

Software Engineering:

The systematic, disciplined quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.The study of approaches as in (1)

EVOLVING ROLE OF SOFTWARE:

Software takes dual role. It is both a **product** and a **vehicle** for delivering a product.

As a **product**: It delivers the computing potential embodied by computer Hardware or by a network of computers.

As a **vehicle**: Itis information transformer-producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as single bit or as complex as a multimedia presentation. Software delivers the most important product of our time-information.

. It transforms personal data

. It manages business information to enhance competitiveness **3.** It provides a gateway to worldwide information networks

. It provides the means for acquiring information.

. Dramatic Improvements in hardware performance

. Vast increases in memory and storage capacity **7.** A wide variety of exotic input and output options

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
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 applications range from astronomy to volcanology from automotive stress analysis to space shuttle orbital dynamics from molecular biology to automated manufacturing computer aided design, system simulation and other interactive applications.

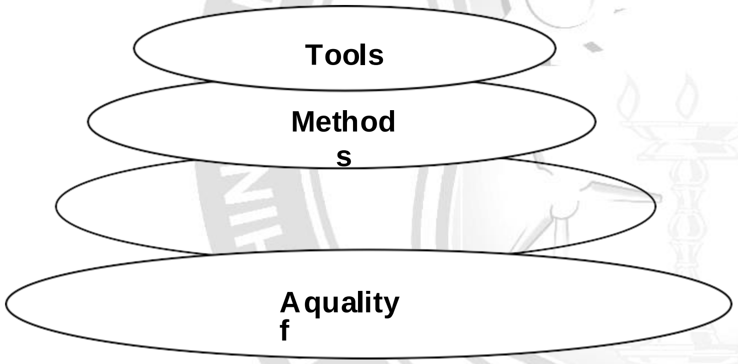
Embedded software:

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

A GENERIC VIEW OF PROCESS

SOFTWARE ENGINEERING - A LAYERED TECHNOLOGY:



Software Engineering Layers

Software engineering is a layered technology. Any engineering approach must rest on an organizational commitment to quality. The bedrock that supports software engineering is a quality focus.

The foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers. Process defines a framework that must be established for effective delivery of software engineering technology.

The software forms the basis for management control of software projects and establishes the context in which

1. technical methods are applied, 2.work products are produced,

3. milestones are established,

4 quality is ensured, And change is properly managed.

A PROCESS FRAMEWORK:

- Software process must be established for effective delivery of software engineering technology.
- **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.
- The process framework encompasses a set of umbrella activities that are applicable across the entire software process.
- Each framework activity is populated by a set of software engineering actions
- Each software engineering action is represented by a number of different task sets- each a collection of software engineering work tasks, related work products, quality assurance points, and project milestones.

"A **process** defines who is doing what, when, and how to reach a certain goal.

THE CAPABILITY MATURITY MODEL INTEGRATION (CMMI):

The CMMI represents a process meta-model in two different ways:

- As a continuous model
- As a staged model.

Each process area is formally assessed against specific goals and practices and is rated according to the following capability levels.

Level 0: Incomplete. The process area is either not performed or does not achieve all goals and objectives defined by CMMI for level 1 capability.

Level 1: Performed. All of the specific goals of the process area have been satisfied. Work tasks required to produce defined work products are being conducted.

Level 2: Managed. All level 1 criteria have been satisfied. In addition, all work associated with the process area conforms to an organizationally defined policy; all people doing the work have access to adequate resources to get the job done; stakeholders are actively involved in the process area as required; all work tasks and work products are "monitored, controlled, and reviewed;

Level 3: Defined. All level 2 criteria have been achieved. In addition, the process is "tailored from the organizations set of standard processes according to the organizations tailoring guidelines, and contributes and work products, measures and other process-improvement information to the organizational process assets".

Level 4: Quantitatively managed. All level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment."Quantitative objectives for quality and process performance are established and used as criteria in managing the process"

Level 5: Optimized. All level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative means to meet changing customer needs and to continually improve the efficacy of the process area under consideration"

The CMMI defines each process area in terms of "specific goals" and the "specific practices" required to achieve these goals. Specific practices refine a goal into a set of process-related activities.

The specific goals (SG) and the associated specific practices(SP) defined for project planning are

SG 1 Establish estimates

SP 1.1 Estimate the scope of the project

SP 1.2 Establish estimates of work product and task attributes SP 1.3 Define project life cycle SP 1.4 Determine estimates of effort and cost

SG 2 Develop a Project Plan

SP 2.1 Establish the budget and schedule SP 2.2 Identify project risks

SP 2.3 Plan for data management

SP 2.4 Plan for needed knowledge and skills SP 2.5 Plan stakeholder involvement SP 2.6 Establish the project plan

SG 3 Obtain commitment to the plan

SP 3.1 Review plans that affect the project SP 3.2 Reconcile work and resource levels SP 3.3 Obtain plan commitment

PERSONAL AND TEAM PROCESS MODELS:

The best software process is one that is close to the people who will be doing the work.Each software engineer would create a process that best fits his or her needs, and at the same time meets the broader needs of the team and the organization. Alternatively, the team itself would create its own process, and at the same time meet the narrower needs of individuals and the broader needs of the organization.

Personal software process (PSP)

The personal software process (PSP) emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product.The PSP process model defines five framework activities: planning, high-level design, high level design review, development, and postmortem.

Planning: This activity isolates requirements and, base on these develops both size and resource estimates. In addition, a defect estimate is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.

High level design: External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.

High level design review: Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.

Development: The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important task and work results.

Postmortem: Using the measures and metrics collected the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness. PSP stresses the need for each software engineer to identify errors early and, as important, to understand the types of errors that he is likely to make.

PSP represents a disciplined, metrics-based approach to software engineering.

Team software process (TSP): The goal of TSP is to build a "self-directed project team that organizes itself to produce high-quality software. The following are the objectives for TSP:

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

Show managers how to coach and motivate their teams and how to help them sustain peak performance.

Accelerate software process improvement by making CMM level 5 behavior normal andexpected.

Provide improvement guidance to high-maturity organizations.

Facilitate university teaching of industrial-grade teamskills. A self-directed team defines

roles and responsibilities for each team member

tracks quantitative project data

identifies a team process that is appropriate for the project

a strategy for implementing the process

defines local standards that are applicable to the teams software engineering work; - continually assesses risk and reacts to it - Tracks, manages, and reports project status.

-

TSP defines the following framework activities: launch, high-level design, implementation, integration and test, and postmortem.

TSP makes use of a wide variety of scripts, forms, and standards that serve to guide team members in their work.

Scripts define specific process activities and other more detailed work functions that are part of the team process.

Each project is "launched" using a sequence of tasks.

The following launch script is recommended

Review project objectives with management and agree on and document team goals

Establish team roles

Define the teams development process

Make a quality plan and set quality targets

Plan for the needed support facilities

PROCESS MODELS

Prescriptive process models define a set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software. These process models are not perfect, but they do provide a useful roadmap for software engineering work.

A prescriptive process model populates a process framework with explicit task sets for software engineering actions.

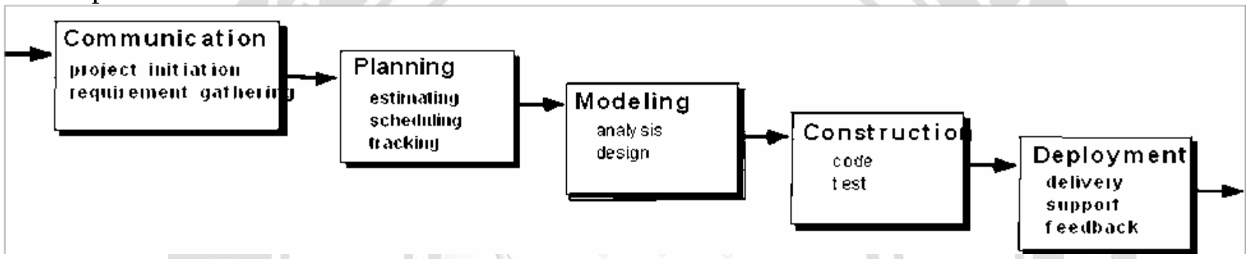
THE WATERFALL MODEL:

The waterfall model, sometimes called the *classic life cycle*, suggests a systematic sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment.

Context: Used when requirements are reasonably well understood.

Advantage:

It can serve as a useful process model in situations where requirements are fixed and work is to proceed to complete in a linear manner.



The **problems** that are sometimes encountered when the waterfall model is applied are:

1. Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
2. It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exist at the beginning of many projects.
3. The customer must have patience. A working version of the programs will not be available until late in the project time-span. If a major blunder is undetected then it can be disastrous until the program is reviewed.

Software Development Life Cycle

What is SDLC?

SDLC 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 Life**
Cycle and is also referred to as the Application Development life-cycle

In this Software Development Life Cycle tutorial, you will learn

- What is SDLC (Software Development Life Cycle)?
- Why SDLC?
- SDLC Phases
- 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
- Popular SDLC models

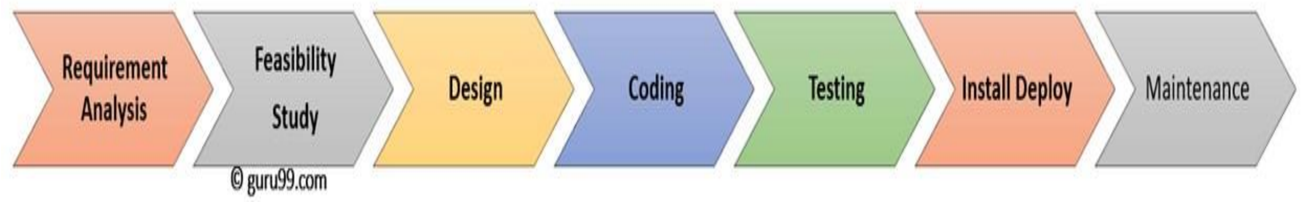
Why SDLC?

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 SDLC steps:



SDLC Phases

- 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 Software Development Life Cycle 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 sdlc 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.
Popular SDLC models
Here, are some of the most important models of Software Development Life Cycle (SDLC):
Waterfall model in SDLC
The waterfall is a widely accepted SDLC model. In this approach, the whole process of the software development is divided into various phases of SDLC. 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.

Incremental Model in SDLC

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 life cycle 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.

V-Model in SDLC

In this type of SDLC model testing and the development, the phase is planned in parallel. So, there are verification phases of SDLC on the side and the validation phase on the other side. V-Model joins by Coding phase.

Agile Model in SDLC

Agile methodology is a practice which promotes continue interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks.

Spiral Model

The spiral model is a risk-driven process model. This SDLC testing 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.

Big bang model

Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come.

This model works best for small projects with smaller size development team which are working together. It is also useful for academic software development projects. It is an ideal model where requirements is either unknown or final release date is not given.

Summary

- The Software Development Life Cycle (SDLC) is a systematic process for building software that ensures the quality and correctness of the software built
- The full form SDLC is Software Development Life Cycle or Systems Development Life Cycle.
- SDLC in software engineering provides a framework for a standard set of activities and deliverables
- Seven different SDLC stages are 1) Requirement collection and analysis 2) Feasibility study: 3) Design 4) Coding 5) Testing: 6) Installation/Deployment and 7) Maintenance
- The senior team members conduct the requirement analysis phase
- Feasibility Study stage includes everything which should be designed and developed during the project life cycle
- In the Design phase, the system and software design documents are prepared as per the requirement specification document
- In the coding phase, developers start build the entire system by writing code using the chosen programming language
- Testing is the next phase which is conducted to verify that the entire application works according to the customer requirement.
- Installation and deployment face begins when the software testing phase is over, and no bugs or errors left in the system
- Bug fixing, upgrade, and engagement actions covered in the maintenance face
- Waterfall, Incremental, Agile, V model, Spiral, Big Bang are some of the popular SDLC models in software engineering

- SDLC in software testing consists of a detailed plan which explains how to plan, build, and maintain specific software

Traditional and Agile Software Development

Difference between Traditional and Agile Software Development

Traditional Software Development:

Traditional software development is the software development process used to design and develop the simple software. It is basically used when the security and many other factors of the software are not much important. It is used by freshers in order to develop the software. It consists of five phases:

1. Requirements analysis
2. Design
3. Implementation
4. Coding and Testing
5. Maintenance

Agile Software Development**:**

Agile software development is the software development process used to design complicated software. It is basically used when the software is quite sensitive and complicated. It is used when security is much important. It is used by professionals in order to develop the software. It consists of three phases:

1. Project initiation
2. Sprint planning
3. Demos

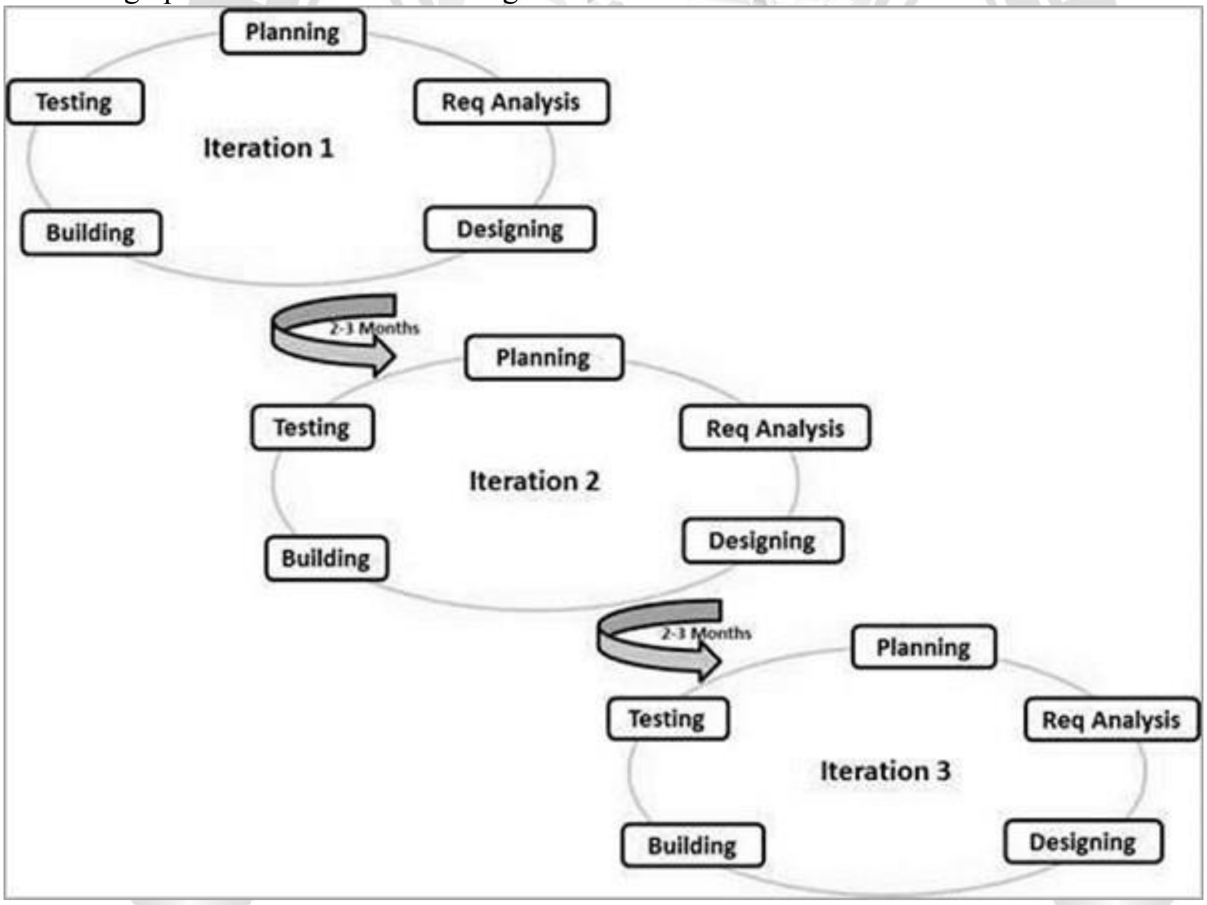
Difference between Traditional and Agile Software Development:

Traditional Software Development	Agile Software Development
It is used to develop the simple software.	It is used to develop the complicated software.
In this methodology, testing is done once the development phase is totally completed.	In this methodology, testing and development processes are performed concurrently.
It provides less security.	It provides high security.
It provides less functionality in the software.	It provides all the functionality needed by the users.
It is basically used by freshers.	It is used by professionals.
Development cost is less using this methodology.	Development cost is high using this methodology.
It majorly consists of five phases.	It consists only three phases.
It is less used by software development firms.	It is normally used by software development firms.

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like −

- Planning
- Requirements Analysis
- Design
- Coding
- Unit Testing and
- Acceptance Testing.

At the end of the iteration, a working product is displayed to the customer and important stakeholders.
What is Agile?
Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.
Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.
Here is a graphical illustration of the Agile Model −



The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.
The most popular Agile methods include Rational Unified Process (1994), Scrum (1995),
Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as **Agile Methodologies**, after the Agile Manifesto was published in 2001.
Following are the Agile Manifesto principles −

- **Individuals and interactions** − In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.

- **Working software** − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
- **Customer collaboration** − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- **Responding to change** − Agile Development is focused on quick responses to change and continuous development.

  Agile Vs Traditional SDLC Models

  Agile is based on the **adaptive software development methods**, whereas the traditional SDLC models like the waterfall model is based on a predictive approach. Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.

  Predictive methods entirely depend on the **requirement analysis and planning** done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

  Agile uses an **adaptive approach** where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

  Customer Interaction is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

  Agile Model - Pros and Cons

  Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the Agile model.

The advantages of the Agile Model are as follows − Is a very realistic approach to software development.

- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum. Suitable for fixed or changing requirements Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

The disadvantages of the Agile Model are as follows − Not suitable for handling complex dependencies.

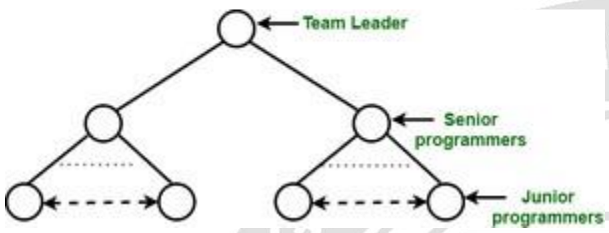
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

  Software Project Team Organization

  There are many ways to organize the project team. Some important ways are as follows :

1. Hierarchical team organization

2. Chief-programmer team organization
3. Matrix team, organization
4. Egoless team organization
5. Democratic team organization **Hierarchical team organization :**
   In this, the people of organization at different levels following a tree structure. People at bottom level generally possess most detailed knowledge about the system. People at higher levels have broader appreciation of the whole project.



Benefits of hierarchical team organization :

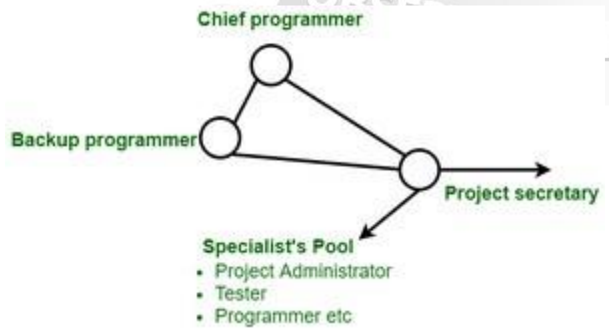
- It limits the number of communication paths and stills allows for the needed communication.
- It can be expanded over multiple levels.
- It is well suited for the development of the hierarchical software products.
- Large software projects may have several levels.

Limitations of hierarchical team organization :

- As information has to be travel up the levels, it may get distorted.
- Levels in the hierarchy often judges people socially and financially.
- Most technical competent programmers tend to be promoted to the management positions which may result in loss of good programmer and also bad manager.

Chief-programmer team organization :
This team organization is composed of a small team consisting the following team members :

- **The Chief programmer :** It is the person who is actively involved in the planning, specification and design process and ideally in the implementation process as well.
- **The project assistant :** It is the closest technical co-worker of the chief programmer.
- **The project secretary :** It relieves the chief programmer and all other programmers of administration tools.
- **Specialists :** These people select the implementation language, implement individual system components and employ software tools and carry out tasks.



Advantages of Chief-programmer team organization :

- Centralized decision-making
- Reduced communication paths

- Small teams are more productive than large teams
- The chief programmer is directly involved in system development and can exercise the better control function.

Disadvantages of Chief-programmer team organization :

- Project survival depends on one person only.
- Can cause the psychological problems as the "chief programmer" is like the "king" who takes all the credit and other members are resentful.
- Team organization is limited to only small team and small team cannot handle every project.
- Effectiveness of team is very sensitive to Chief programmer's technical and managerial activities.

Matrix Team Organization :

In matrix team organization, people are divided into specialist groups. Each group has a manager. Example of Metric team organization is as follows :

Egoless Team Organization :

Egoless programming is a state of mind in which programmer are supposed to separate themselves from their product. In this team organization goals are set and decisions are made by group consensus. Here group, 'leadership' rotates based on tasks to be performed and differing abilities of members. In this organization work products are discussed openly and all freely examined all team members. There is a major risk which such organization, if teams are composed of inexperienced or incompetent members.

Democratic Team Organization :

It is quite similar to the egoless team organization, but one member is the team leader with some responsibilities :

- Coordination
- Final decisions, when consensus cannot be reached.

Advantages of Democratic Team Organization :

- Each member can contribute to decisions.
- Members can learn from each other.
- Improved job satisfaction.

Disadvantages of Democratic Team Organization :

- Communication overhead increased.
- Need for compatibility of members.
- Less individual responsibility and authority