UNIT-1

Introduction to Software Engineering: The evolving role of software, changing nature of software, software myths. A Generic view of process: Software engineering- a layered technology, a process framework, the capability maturity model integration (CMMI). Process models: The waterfall model, Spiral model and Agile methodology.

- **Software** is a program or set of programs containing instructions that provide the desired functionality. Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.
Engineering: Engineering is the application of scientific and practical knowledge to invent, design, build, maintain, and improve frameworks, processes, etc.

What is Software Engineering?
Software Engineering is the process of designing, developing, testing, and maintaining software. It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software.

1. Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.
2. It is a rapidly evolving field, and new tools and technologies are constantly being developed to improve the software development process.
3. By following the principles of software engineering and using the appropriate tools and methodologies, software developers can create high-quality, reliable, and maintainable software that meets the needs of its users.
4. Software Engineering is mainly used for large projects based on software systems rather than single programs or applications.
5. The main goal of Software Engineering is to develop software applications for improving quality, budget, and time efficiency.
6. Software Engineering ensures that the software that has to be built should be consistent, correct, also on budget, on time, and within the required requirements.

Key Principles of Software Engineering

1. **Modularity**: Breaking the software into smaller, reusable components that can be developed and tested independently.
2. **Abstraction**: Hiding the implementation details of a component and exposing only the necessary functionality to other parts of the software.

3. **Encapsulation**: Wrapping up the data and functions of an object into a single unit, and protecting the internal state of an object from external modifications.
4. **Reusability**: Creating components that can be used in multiple projects, which can save time and resources.
5. **Maintenance**: Regularly updating and improving the software to fix bugs, add new features, and address security vulnerabilities.
6. **Testing**: Verifying that the software meets its requirements and is free of bugs.
7. **Design Patterns**: Solving recurring problems in software design by providing templates for solving them.
8. **Agile methodologies:** Using iterative and incremental development processes that focus on customer satisfaction, rapid delivery, and flexibility.
9. **Continuous Integration & Deployment:** Continuously integrating the code changes and deploying them into the production environment.

Characteristics of Software Engineering:

Software Engineering is a systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance of a software system. There are four main Attributes of Software Engineering.

1. **Efficiency:** It provides a measure of the resource requirement of a software product efficiently.
2. **Reliability:** It assures that the product will deliver the same results when used in similar working environment.
3. **Reusability:** This attribute makes sure that the module can be used in multiple applications.
4. **Maintainability:** It is the ability of the software to be modified, repaired, or enhanced easily with changing requirements.

Evolving role (or) Dual Role of Software:

There is a dual role of software in the industry. The first one is as a product and the other one is as a vehicle for delivering the product. We will discuss both of them.

1. As a Product

- It delivers computing potential across networks of Hardware.
- It enables the Hardware to deliver the expected functionality.
- It acts as an information transformer because it produces, manages, acquires, modifies, displays, or transmits information.

2. As a Vehicle for Delivering a Product

- It provides system functionality (e.g., payroll system).
- It controls other software (e.g., an operating system).
- It helps build other software (e.g., software tools).

Objectives of Software Engineering

1. **Maintainability:** It should be feasible for the software to evolve to meet changing requirements.
2. **Efficiency:** The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
3. **Correctness:** A software product is correct if the different requirements specified in the SRS Document have been correctly implemented.
4. **Reusability:** A software product has good reusability if the different modules of the product can easily be reused to develop new products.
5. **Testability:** Here software facilitates both the establishment of test criteria and the evaluation of the software concerning those criteria.

6. **Reliability:** It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
7. **Portability:** In this case, the software can be transferred from one computer system or environment to another.
8. **Adaptability:** In this case, the software allows differing system constraints and the user needs to be satisfied by making changes to the software.
9. **Interoperability:** Capability of 2 or more functional units to process data cooperatively.

Program vs Software Product

Parameters	Program	Software Product
Definition	A program is a set of instructions that are given to a computer in order to achieve a specific task.	Software is when a program is made available for commercial business and is properly documented along with its licensing. Software Product = Program + Documentation + Licensing.
Stages Involved	Program is one of the stages involved in the development of the software.	Software Development usually follows a life cycle, which involves the feasibility study of the project, requirement gathering, development of a prototype, system design, coding, and testing.

Advantages of Software Engineering

There are several advantages to using a systematic and disciplined approach to software development, such as:

1. **Improved Quality:** By following established software engineering principles and techniques, the software can be developed with fewer bugs and higher reliability.
2. **Increased Productivity:** Using modern tools and methodologies can streamline the development process, allowing developers to be more productive and complete projects faster.
3. **Better Maintainability:** Software that is designed and developed using sound software engineering practices is easier to maintain and update over time.
4. **Reduced Costs:** By identifying and addressing potential problems early in the development process, software engineering can help to reduce the cost of fixing bugs and adding new features later on.
5. **Increased Customer Satisfaction:** By involving customers in the development process and developing software that meets their needs, software engineering can help to increase customer satisfaction.
6. **Better Team Collaboration:** By using Agile methodologies and continuous integration, software engineering allows for better collaboration among development teams.
7. **Better Scalability**: By designing software with scalability in mind, software engineering can help to ensure that software can handle an increasing number of users and transactions.

8. **Better Security:** By following the Software Development Life Cycle (SDLC) and performing security testing, software engineering can help to prevent security breaches and protect sensitive data.

In summary, software engineering offers a structured and efficient approach to software development, which can lead to higher-quality software that is easier to maintain and adapt to changing requirements. This can help to improve customer satisfaction and reduce costs, while also promoting better collaboration among development teams.

Disadvantages of Software Engineering

While Software Engineering offers many advantages, there are also some potential disadvantages to consider:

1. **High upfront costs:** Implementing a systematic and disciplined approach to software development can be resource-intensive and require a significant investment in tools and training.
2. **Limited flexibility:** Following established software engineering principles and methodologies can be rigid and may limit the ability to quickly adapt to changing requirements.
3. **Bureaucratic**: Software Engineering can create an environment that is bureaucratic, with a lot of processes and paperwork, which may slow down the development process.
4. **Complexity**: With the increase in the number of tools and methodologies, software engineering can be complex and difficult to navigate.
5. **Limited creativity:** The focus on structure and process can stifle creativity and innovation among developers.
6. **High learning curve:** The development process can be complex, and it requires a lot of learning and training, which can be challenging for new developers.
7. **High dependence on tools:** Software engineering heavily depends on the tools, and if the tools are not properly configured or are not compatible with the software, it can cause issues.
8. **High maintenance**: The software engineering process requires regular maintenance to ensure that the software is running efficiently, which can be costly and time-consuming.

In summary, software engineering can be expensive and time-consuming, and it may limit flexibility and creativity. However, the benefits of improved quality, increased productivity, and better maintainability can outweigh the costs and complexity. It's important to weigh the pros and cons of using software engineering and determine if it is the right approach for a particular software project.

Changing Nature of Software:

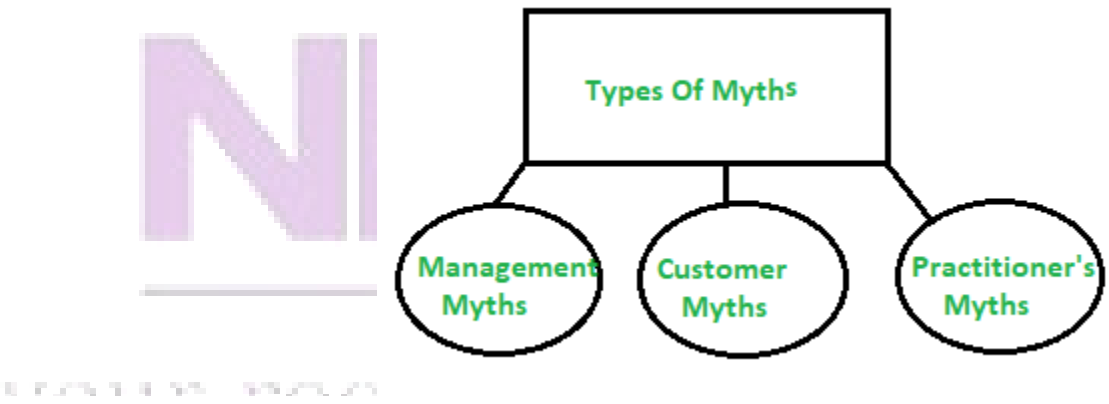
Nowadays, seven broad categories of computer software present continuing challenges for software engineers. Which is given below:

1. **System Software:** System software is a collection of programs that are written to service other programs. Some system software processes complex but determinate, information structures. Other system application processes largely indeterminate data. Sometimes when, the system software area is characterized by the heavy interaction with computer hardware that requires scheduling, resource sharing, and sophisticated process management.
2. **Application Software:** Application software is defined as programs that solve a specific business need. Application in this area processes business or technical data in a way that facilitates business operation or management technical decision-making. In addition to conventional data processing applications, application software is used to control business functions in real-time.

3. **Engineering and Scientific Software:** This software is used to facilitate the engineering function and task. however modern applications within the engineering and scientific area are moving away from conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take a real-time and even system software characteristic.
4. **Embedded Software:** Embedded software resides within the system or product and is used to implement and control features and functions for the end-user and for the system itself. Embedded software can perform limited and esoteric functions or provide significant function and control capability.
5. **Product-line Software:** Designed to provide a specific capability for use by many customers, product-line software can focus on the limited and esoteric marketplace or address the mass consumer market.
6. **Web Application:** It is a client-server computer program that the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as e-commerce and B2B applications grow in importance. Web apps are evolving into a sophisticated computing environment that not only provides a standalone feature, computing function, and content to the end user.
7. **Artificial Intelligence Software:** Artificial intelligence software makes use of a non numerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis. Applications within this area include robotics, expert systems, pattern recognition, artificial neural networks, theorem proving, and game playing.

Software Myths

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



`Types of Software Myths

(i) Management Myths:

Myth 1:

We have all the standards and procedures available for software development.

Fact:

- Software experts do not know all the requirements for the software development.

- And all existing processes are incomplete as new software development is based on new and different problem.

Myth 2:

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

Fact:

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

Myth 3:

- With the addition of more people and program planners to Software development can help meet project deadlines (If lagging behind).

Fact:

- If software is late, adding more people will merely make the problem worse. This is because the people already working on the project now need to spend time educating the newcomers, and are thus taken away from their work. The newcomers are also far less productive than the existing software engineers, and so the work put into training them to work on the software does not immediately meet with an appropriate reduction in work.

(ii) Customer Myths:

The customer can be the direct users of the software, the technical team, marketing / sales department, or other company. Customer has myths leading to false expectations (customer) & that's why you create dissatisfaction with the developer.

Myth 1:

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

Fact:

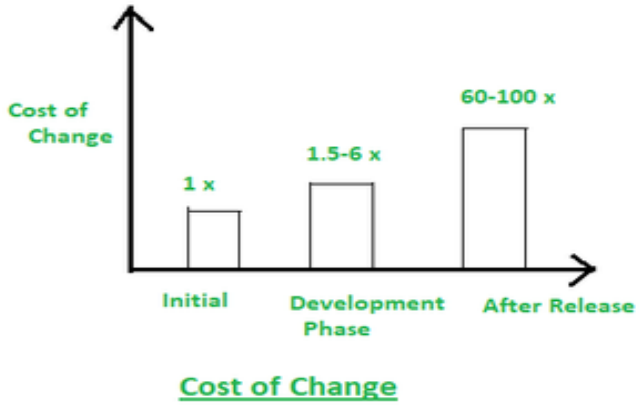
- Official and detailed description of the database function, ethical performance, communication, structural issues and the verification process are important.
- Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

Myth 2:

Software requirements continually change, but change can be easily accommodated because software is flexible

Fact:

- It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small. However, as time passes, the cost impact grows rapidly—resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modification.



Different Stages of Myths

(iii)Practitioner's Myths:

Myths 1:

They believe that their work has been completed with the writing of the plan.

Fact:

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

Myths 2:

There is no other way to achieve system quality, until it is "running".

Fact:

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

Myth 3:

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

Fact:

- A working system is not enough, the right document brochures and booklets are also required to provide guidance & software support.

Myth 4:

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

Fact:

- Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

A Generic view of process:

Layered Technology in Software Engineering

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

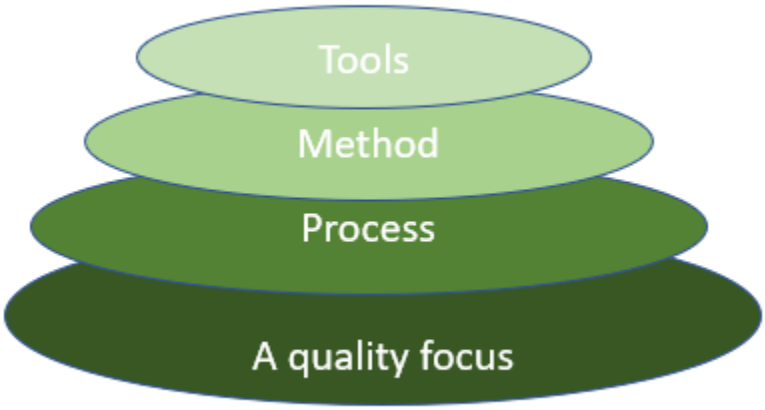
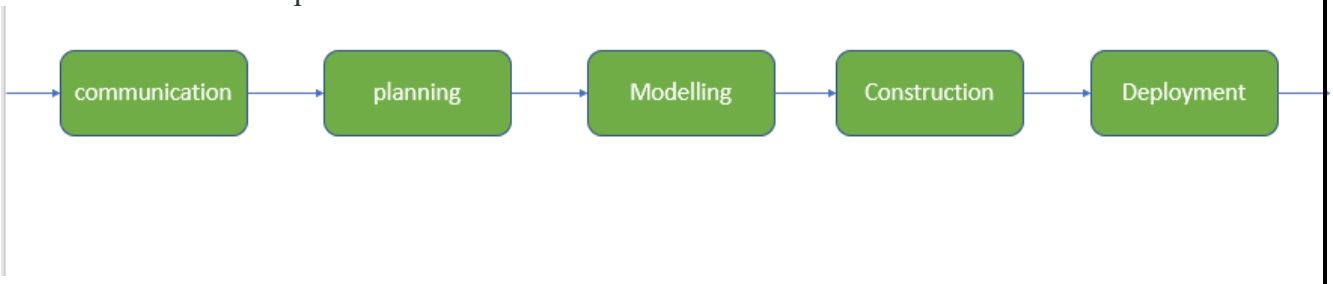


Fig: The diagram shows the layers of software development

Layered technology is divided into four parts:

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

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



Process activities are listed below:-

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

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

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

Software Process Framework – Software Engineering

A **Software Process Framework** is a structured approach that defines the steps, tasks, and activities involved in software development. This framework serves as a **foundation for software engineering**, guiding the development team through various stages to ensure a **systematic and efficient process**. A Software Process Framework helps in project planning, risk management, and quality assurance by detailing the chronological order of actions.

What is a Software Process Framework?

Software Process Framework details the steps and chronological order of a process. Since it serves as a foundation for them, it is utilized in most applications. Task sets, umbrella activities, and process framework activities all define the characteristics of the software development process. Software Process includes:

1. **Tasks:** They focus on a small, specific objective.
2. **Action:** It is a set of tasks that produce a major work product.
3. **Activities:** Activities are groups of related tasks and actions for a major objective.

What Is a Software Development Framework?

A software development framework is a structured set of tools, libraries, best practices, and guidelines that help developers build software applications. Think of it as a template or foundation that provides the basic structure and components needed for a software project.

Key Points

1. **Foundation**: It gives a basic structure or template for developing software, so developers don't have to start from scratch.
2. **Components and Tools**: It includes pre-built components and tools that make development faster and easier.
3. **Best Practices and Guidelines**: It offers best practices and guidelines to ensure the software is built in an organized and efficient way.
4. **Customization**: Developers can modify and add new functions to customize the framework to their specific needs.

Advantages of Software Development Framework

A **Software Development Framework** offers numerous benefits that streamline the **software development process** and enhance the quality and efficiency of the final product. Here are some key advantages:

1. **Increased Productivity**: Frameworks provide pre-built components and tools, allowing developers to focus on specific application logic rather than reinventing the wheel.
2. **Consistent Quality**: By following best practices and standardized processes, frameworks help ensure consistent code quality and structure across the project.
3. **Reduced Development Time**: With ready-to-use templates and libraries, developers can significantly cut down on the time needed to build applications from scratch.
4. **Better Maintainability**: A structured framework makes the codebase more organized and easier to understand, which simplifies maintenance and updates.
5. **Enhanced Security**: Frameworks often include built-in security features and follow industry best practices, reducing the risk of vulnerabilities.

6. **Scalability**: Frameworks are designed to handle growth, making it easier to scale applications as user demand increases.

Disadvantages of Software Development Framework

While **Software Development Frameworks** offer several advantages, they also come with certain drawbacks that developers and organizations should consider:

1. **Learning Curve**: Frameworks often have a steep learning curve, requiring developers to invest time and effort in understanding the framework's architecture, conventions, and best practices.
2. **Restrictions**: Some frameworks impose constraints and limitations on how developers can design and implement certain features, potentially limiting flexibility and creativity.
3. **Complexity Overhead**: In some cases, frameworks introduce unnecessary complexity, especially for smaller or simpler projects, which can lead to over-engineering.
4. **Performance Overhead**: Using a framework may introduce additional layers of abstraction and overhead, which can impact the performance of the application, particularly in resource-intensive environments.
5. **Vendor Lock-in**: Depending heavily on a specific framework can lead to vendor lock-in, making it challenging to switch to alternative technologies or frameworks in the future.

How to Choose a Suitable Development Framework

Choosing a suitable development framework is crucial for the success of a software project. Here are key steps to help you make an informed decision. Here is a simple and effective strategy to help you select the most suitable framework for your project

1. Consider the Framework's Language

- **Popular Languages**: Start with frameworks in popular programming languages like Java, Python, or Ruby if you have no preference. These languages often have robust frameworks with strong community support.

2. Open-Source vs. Paid Frameworks

- **Open-Source**: Generally have a large user base, frequent updates, and community contributions.
- **Paid**: Often more reliable with better support but may lack customization and timely updates.

3. Community and Support

- **Community Size**: A large, active community means better support, more tutorials, and a more mature framework. Look for frameworks with extensive community resources and engagement.

4. Review Case Studies and Example Applications

- **Practical Insights**: Check the framework's website or repositories for case studies or example applications. These can provide insights into development processes and methods that work well with the framework.

5. Test the Framework Yourself

- **Hands-On Experience**: Try out the framework in your own project to see how it fits your needs. Testing helps you understand the framework's functionality and whether it suits your development scenario.

Software Process Framework Activities

The Software process framework is required for representing common process activities. Five framework activities are described in a process framework for software engineering. Communication, planning, modeling, construction, and deployment are all examples of framework

activities. Each engineering action defined by a framework activity comprises a list of needed work outputs, project milestones, and software quality assurance (SQA) points.

Capability Maturity Model Integration (CMMI)

- The Capability Maturity Model Integration (CMMI) is an advanced framework designed to improve and integrate processes across various disciplines such as software engineering, systems engineering, and people management. It builds on the principles of the original CMM, enabling organizations to enhance their processes systematically. CMMI helps organizations fulfill customer needs, create value for investors, and improve product quality and market growth. It offers two representations, staged and continuous, to guide organizations in their process improvement efforts.

What is Capability Maturity Model Integration (CMMI)?

Capability Maturity Model Integration (CMMI) is a successor of CMM and is a more evolved model that incorporates best components of individual disciplines of CMM like Software CMM, Systems Engineering CMM, People CMM, etc. Since CMM is a reference model of matured practices in a specific discipline, so it becomes difficult to integrate these disciplines as per the requirements. This is why CMMI is used as it allows the integration of multiple disciplines as and when needed.

Objectives of CMMI

1. Fulfilling customer needs and expectations.
2. Value creation for investors/stockholders.
3. Market growth is increased.
4. Improved quality of products and services.
5. Enhanced reputation in Industry.

CMMI Representation – Staged and Continuous

A representation allows an organization to pursue a different set of improvement objectives. There are two representations for CMMI :

- **Staged Representation :**
  - uses a pre-defined set of process areas to define improvement path.
  - provides a sequence of improvements, where each part in the sequence serves as a foundation for the next.
  - an improved path is defined by maturity level.
  - maturity level describes the maturity of processes in organization.
  - Staged CMMI representation allows comparison between different organizations for multiple maturity levels.
- **Continuous Representation :**
  - allows selection of specific process areas.
  - uses capability levels that measures improvement of an individual process area.
  - Continuous CMMI representation allows comparison between different organizations on a process-area-by-process-area basis.
  - allows organizations to select processes which require more improvement.
  - In this representation, order of improvement of various processes can be selected which allows the organizations to meet their objectives and eliminate risks.

CMMI Model – Maturity Levels

In CMMI with staged representation, there are five maturity levels described as follows :

1. **Maturity level 1 : Initial**
   - processes are poorly managed or controlled.
   - unpredictable outcomes of processes involved.
   - ad hoc and chaotic approach used.
   - No KPAs (Key Process Areas) defined.
   - Lowest quality and highest risk.
2. **Maturity level 2 : Managed**
   - requirements are managed.
   - processes are planned and controlled.
   - projects are managed and implemented according to their documented plans.
   - This risk involved is lower than Initial level, but still exists.
   - Quality is better than Initial level.
3. **Maturity level 3 : Defined**
   - processes are well characterized and described using standards, proper procedures, and methods, tools, etc.
   - Medium quality and medium risk involved.
   - Focus is process standardization.
4. **Maturity level 4 : Quantitatively managed**
   - quantitative objectives for process performance and quality are set.
   - quantitative objectives are based on customer requirements, organization needs, etc.
   - process performance measures are analyzed quantitatively.
   - higher quality of processes is achieved.
   - lower risk
5. **Maturity level 5 : Optimizing**
   - continuous improvement in processes and their performance.
   - improvement has to be both incremental and innovative.
   - highest quality of processes.
   - lowest risk in processes and their performance.

CMMI Model – Capability Levels

A capability level includes relevant specific and generic practices for a specific process area that can improve the organization's processes associated with that process area. For CMMI models with continuous representation, there are six capability levels as described below :

1. **Capability level 0 : Incomplete**
   - incomplete process – partially or not performed.
   - one or more specific goals of process area are not met.
   - No generic goals are specified for this level.
   - this capability level is same as maturity level 1.
2. **Capability level 1 : Performed**
   - process performance may not be stable.
   - objectives of quality, cost and schedule may not be met.
   - a capability level 1 process is expected to perform all specific and generic practices for this level.
   - only a start-step for process improvement.
3. **Capability level 2 : Managed**
   - process is planned, monitored and controlled.

- managing the process by ensuring that objectives are achieved.
- objectives are both model and other including cost, quality, schedule.
- actively managing processing with the help of metrics.

4. **Capability level 3 : Defined**
   - a defined process is managed and meets the organization's set of guidelines and standards.
   - focus is process standardization.
5. **Capability level 4 : Quantitatively Managed**
   - process is controlled using statistical and quantitative techniques.
   - process performance and quality is understood in statistical terms and metrics.
   - quantitative objectives for process quality and performance are established.
6. **Capability level 5 : Optimizing**
   - focuses on continually improving process performance.
   - performance is improved in both ways – incremental and innovation.
   - emphasizes on studying the performance results across the organization to ensure that common causes or issues are identified and fixed.

Process Models:
Waterfall Model – Software Engineering

The waterfall model is the basic software development life cycle model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. However, it is very important because all the other software development life cycle models are based on the classical waterfall model.

What is the SDLC Waterfall Model ?

The waterfall model is a software development model used in the context of large, complex projects, typically in the field of information technology. It is characterized by a structured, sequential approach to project management and software development.

The waterfall model is useful in situations where the project requirements are well-defined and the project goals are clear. It is often used for large-scale projects with long timelines, where there is little room for error and the project stakeholders need to have a high level of confidence in the outcome.

Features of the SDLC Waterfall Model

1. **Sequential Approach**: The waterfall model involves a sequential approach to software development, where each phase of the project is completed before moving on to the next one.
2. **Document-Driven:** The waterfall model relies heavily on documentation to ensure that the project is well-defined and the project team is working towards a clear set of goals.
3. **Quality Control:** The waterfall model places a high emphasis on quality control and testing at each phase of the project, to ensure that the final product meets the requirements and expectations of the stakeholders.
4. **Rigorous Planning**: The waterfall model involves a rigorous planning process, where the project scope, timelines, and deliverables are carefully defined and monitored throughout the project lifecycle.

Overall, the waterfall model is used in situations where there is a need for a highly structured and systematic approach to software development. It can be effective in ensuring that large, complex projects are completed on time and within budget, with a high level of quality and customer satisfaction.

Importance of SDLC Waterfall Model

1. **Clarity and Simplicity:** The linear form of the Waterfall Model offers a simple and unambiguous foundation for project development.
2. **Clearly Defined Phases:** The Waterfall Model's phases each have unique inputs and outputs, guaranteeing a planned development with obvious checkpoints.
3. **Documentation:** A focus on thorough documentation helps with software comprehension, upkeep, and future growth.
4. **Stability in Requirements:** Suitable for projects when the requirements are clear and steady, reducing modifications as the project progresses.
5. **Resource Optimization:** It encourages effective task-focused work without continuously changing contexts by allocating resources according to project phases.
6. **Relevance for Small Projects:** Economical for modest projects with simple specifications and minimal complexity.

Phases of SDLC Waterfall Model – Design

The Waterfall Model is a classical software development methodology that was first introduced by Winston W. Royce in 1970. It is a linear and sequential approach to software development that consists of several phases that must be completed in a specific order.

The Waterfall Model has six phases which are:

1. Requirements: The first phase involves gathering requirements from stakeholders and analyzing them to understand the scope and objectives of the project.

2. Design: Once the requirements are understood, the design phase begins. This involves creating a detailed design document that outlines the software architecture, user interface, and system components.

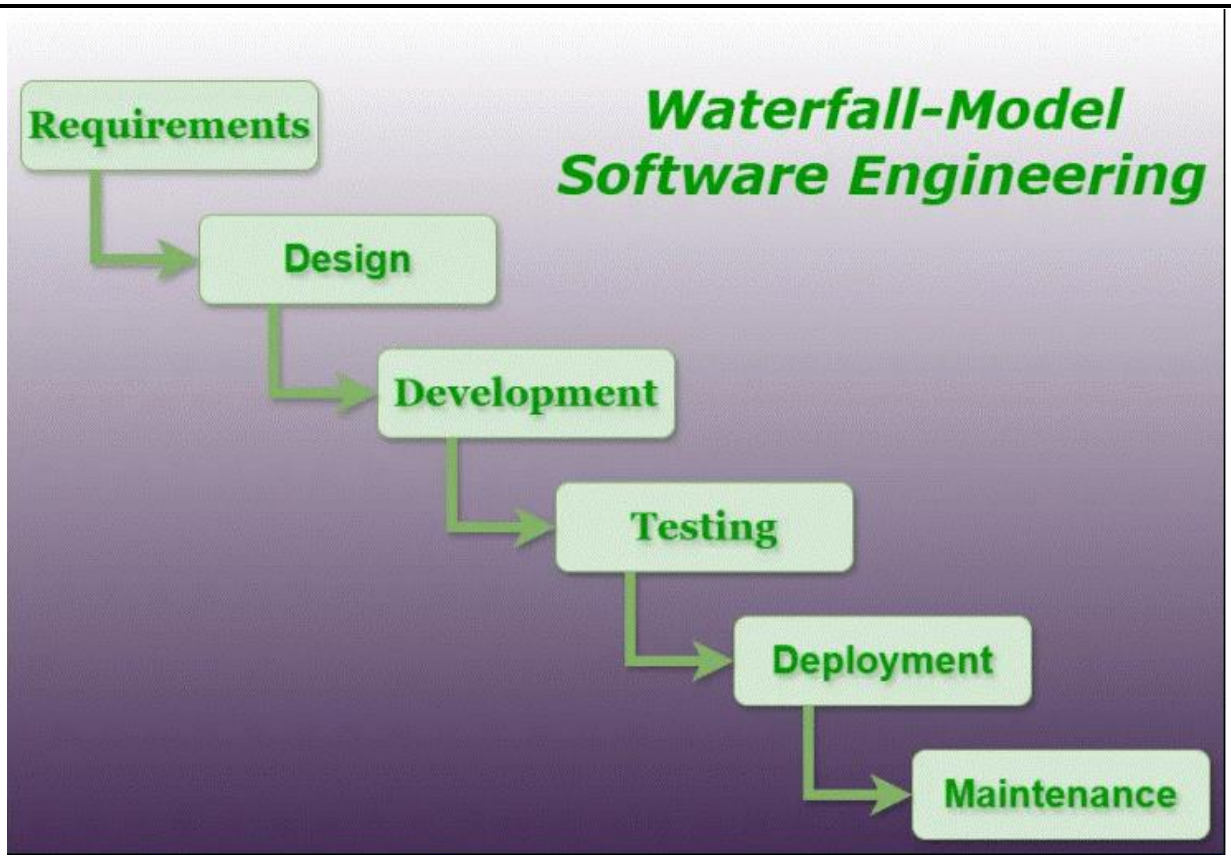
3. Development: The Development phase include implementation involves coding the software based on the design specifications. This phase also includes unit testing to ensure that each component of the software is working as expected.

4. Testing: In the testing phase, the software is tested as a whole to ensure that it meets the requirements and is free from defects.

5. Deployment: Once the software has been tested and approved, it is deployed to the production environment.

6. Maintenance: The final phase of the Waterfall Model is maintenance, which involves fixing any issues that arise after the software has been deployed and ensuring that it continues to meet the requirements over time.

The classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after the completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure.



Waterfall Model-Software Engineering

Let us now learn about each of these phases in detail which include further phases.

1. Feasibility Study:

The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software.

The feasibility study involves understanding the problem and then determining the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.

2. Requirements Analysis **and Specification:**

The requirement analysis and specification phase aims to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

- **Requirement gathering and analysis:** Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (an inconsistent requirement is one in which some part of the requirement contradicts some other part).
- **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

3. Design:

The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture. A Software Design Document is used to document all of this effort (SDD).

4. Coding and Unit Testing:

In the coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The unit testing phase aims to check whether each module is working properly or not.

5. Integration and System testing:

Integration of different modules is undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over several steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing. System testing consists of three different kinds of testing activities as described below.

- **Alpha testing:** Alpha testing is the system testing performed by the development team.
- **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
- **Acceptance testing:** After the software has been delivered, the customer performs acceptance testing to determine whether to accept the delivered software or reject it.

6. Maintenance:

Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software. There are three types of maintenance.

- **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
- **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.
- **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as working on a new computer platform or with a new operating system.

Advantages of the SDLC Waterfall Model

The classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model.

- **Easy to Understand:** The Classical Waterfall Model is very simple and easy to understand.
- **Individual Processing:** Phases in the Classical Waterfall model are processed one at a time.
- **Properly Defined:** In the classical waterfall model, each stage in the model is clearly defined.
- **Clear Milestones:** The classical Waterfall model has very clear and well-understood milestones.
- **Properly Documented:** Processes, actions, and results are very well documented.
- **Reinforces Good Habits:** The Classical Waterfall Model reinforces good habits like define-before-design and design-before-code.
- **Working:** Classical Waterfall Model works well for smaller projects and projects where requirements are well understood.

Disadvantages of the SDLC Waterfall Model

The Waterfall Model suffers from various shortcomings we can't use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model.

- **No Feedback Path:** In the classical waterfall model evolution of software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phase. Therefore, it does not incorporate any mechanism for error correction.
- **Difficult to accommodate Change Requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but the customer's requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
- **No Overlapping of Phases:** This model recommends that a new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase efficiency and reduce cost, phases may overlap.
- **Limited Flexibility:** The Waterfall Model is a rigid and linear approach to software development, which means that it is not well-suited for projects with changing or uncertain requirements. Once a phase has been completed, it is difficult to make changes or go back to a previous phase.
- **Limited Stakeholder Involvement:** The Waterfall Model is a structured and sequential approach, which means that stakeholders are typically involved in the early phases of the project (requirements gathering and analysis) but may not be involved in the later phases (implementation, testing, and deployment).
- **Late Defect Detection:** In the Waterfall Model, testing is typically done toward the end of the development process. This means that defects may not be discovered until late in the development process, which can be expensive and time-consuming to fix.
- **Lengthy Development Cycle:** The Waterfall Model can result in a lengthy development cycle, as each phase must be completed before moving on to the next. This can result in delays and increased costs if requirements change or new issues arise.

Spiral Model in Software Engineering

-

The Spiral Model is one of the most important Software Development Life Cycle models. The Spiral Model is a combination of the waterfall model and the iterative model. It provides support for **Risk Handling**. The Spiral Model was first proposed by **Barry Boehm**. This article focuses on discussing the Spiral Model in detail.

What is the Spiral Model?

The Spiral Model is a **Software Development Life Cycle (SDLC)** model that provides a systematic and iterative approach to software development. In its diagrammatic representation, looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a **phase** of the software development process.

Some Key Points regarding the phase of a Spiral Model:

1. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks.

2. As the project manager dynamically determines the number of phases, the project manager has an important role in developing a product using the spiral model.
3. It is based on the idea of a spiral, with each iteration of the spiral representing a complete software development cycle, from requirements gathering and analysis to design, implementation, testing, and maintenance.

What Are the Phases of the Spiral Model?

The Spiral Model is a risk-driven model, meaning that the focus is on managing risk through multiple iterations of the software development process. It consists of the following phases:

1. **Objectives Defined: I**n first phase of the spiral model we clarify what the project aims to achieve, including functional and non-functional requirements.
2. **Risk Analysis:** In the risk analysis phase, the risks associated with the project are identified and evaluated.
3. **Engineering:** In the engineering phase, the software is developed based on the requirements gathered in the previous iteration.
4. **Evaluation:** In the evaluation phase, the software is evaluated to determine if it meets the customer's requirements and if it is of high quality.
5. **Planning:** The next iteration of the spiral begins with a new planning phase, based on the results of the evaluation.

The Spiral Model is often used for complex and large software development projects, as it allows for a more flexible and adaptable approach to software development. It is also well-suited to projects with significant uncertainty or high levels of risk.

The Radius of the spiral at any point represents the expenses (cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.

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

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop the next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so-far developed version of the software. In the end, planning for the next phase is started.

Risk Handling in Spiral Model

A risk is any adverse situation that might affect the successful completion of a software project. The most important feature of the spiral model is handling these unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype.

1. The spiral model supports coping with risks by providing the scope to build a prototype at every phase of software development.

2. The **Prototyping Model** also supports risk handling, but the risks must be identified completely before the start of the development work of the project.
3. But in real life, project risk may occur after the development work starts, in that case, we cannot use the Prototyping Model.
4. In each phase of the Spiral Model, the features of the product dated and analyzed, and the risks at that point in time are identified and are resolved through prototyping.
5. Thus, this model is much more flexible compared to other SDLC models.

Why Spiral Model is called Meta Model?

The Spiral model is called a Meta-Model because it subsumes all the other SDLC models. For example, a single loop spiral actually represents the Iterative Waterfall Model.

1. The spiral model incorporates the stepwise approach of the Classical Waterfall Model.
2. The spiral model uses the approach of the Prototyping Model by building a prototype at the start of each phase as a risk-handling technique.
3. Also, the spiral model can be considered as supporting the Evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

Advantages of the Spiral Model

Below are some advantages of the Spiral Model.

1. **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
2. **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
3. **Flexibility in Requirements:** Change requests in the Requirements at a later phase can be incorporated accurately by using this model.
4. **Customer Satisfaction:** Customers can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.
5. **Iterative and Incremental Approach:** The Spiral Model provides an iterative and incremental approach to software development, allowing for flexibility and adaptability in response to changing requirements or unexpected events.
6. **Emphasis on Risk Management:** The Spiral Model places a strong emphasis on risk management, which helps to minimize the impact of uncertainty and risk on the software development process.
7. **Improved Communication:** The Spiral Model provides for regular evaluations and reviews, which can improve communication between the customer and the development team.
8. **Improved Quality:** The Spiral Model allows for multiple iterations of the software development process, which can result in improved software quality and reliability.

Disadvantages of the Spiral Model

Below are some main disadvantages of the spiral model.

1. **Complex:** The Spiral Model is much more complex than other SDLC models.
2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.

3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, time estimation is very difficult.
5. **Complexity:** The Spiral Model can be complex, as it involves multiple iterations of the software development process.
6. **Time-Consuming:** The Spiral Model can be time-consuming, as it requires multiple evaluations and reviews.
7. **Resource Intensive:** The Spiral Model can be resource-intensive, as it requires a significant investment in planning, risk analysis, and evaluations.

The most serious issue we face in the cascade model is that taking a long length to finish the item, and the product became obsolete. To tackle this issue, we have another methodology, which is known as the Winding model or spiral model. The winding model is otherwise called the cyclic model.

Agile Methodology in Software Engineering

- Agile Software Development is a software development methodology that values flexibility, collaboration, and customer satisfaction. It is based on the Agile Manifesto, a set of principles for software development that prioritize individuals and interactions, working software, customer collaboration, and responding to change.

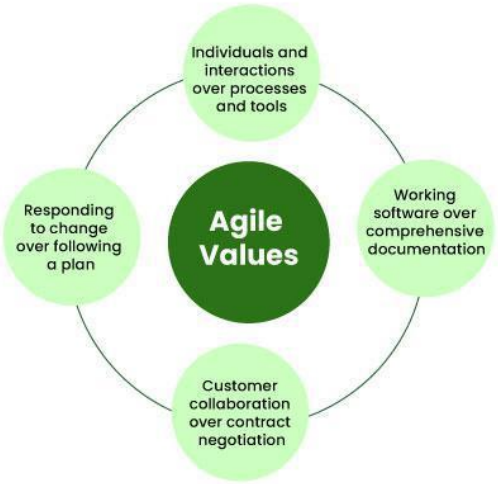
Agile Software Development is an iterative and incremental approach to software development that emphasizes the importance of delivering a working product quickly and frequently. It involves close collaboration between the development team and the customer to ensure that the product meets their needs and expectations.

Why Agile is Used?

1. **Creating Tangible Value:** Agile places a high priority on creating tangible value as soon as possible in a project. Customers can benefit from early delivery of promised advantages and opportunity for prompt feedback and modifications.
2. **Concentrate on Value-Added Work**: Agile methodology promotes teams to concentrate on producing functional and value-added product increments, hence reducing the amount of time and energy allocated to non-essential tasks.
3. **Agile as a Mindset:** Agile represents a shift in culture that values adaptability, collaboration, and client happiness. It gives team members more authority and promotes a cooperative and upbeat work atmosphere.
4. **Quick Response to Change:** Agile fosters a culture that allows teams to respond swiftly to constantly shifting priorities and requirements. This adaptability is particularly useful in sectors of the economy or technology that experience fast changes.
5. **Regular Demonstrations:** Agile techniques place a strong emphasis on regular demonstrations of project progress. Stakeholders may clearly see the project's status, upcoming problems, and upcoming new features due to this transparency.
6. **Cross-Functional Teams:** Agile fosters self-organizing, cross-functional teams that share information effectively, communicate more effectively and feel more like a unit.

4 Core Values of Agile Software Development

The Agile Software Development Methodology Manifesto describe four core values of Agile in software development.



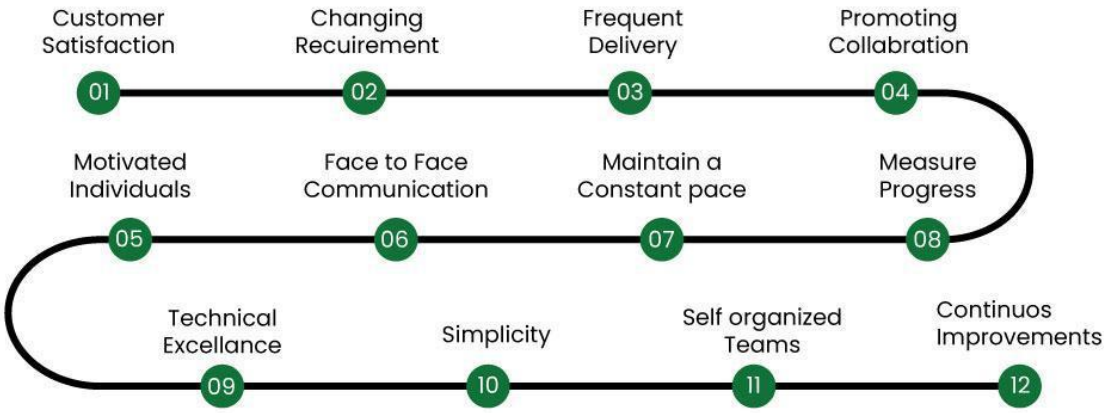
4 Values of Agile

4 Values of Agile

1. Individuals and Interactions over Processes and Tools
2. Working Software over Comprehensive Documentation
3. Customer Collaboration over Contract Negotiation
4. Responding to Change over Following a Plan

Principles of Agile Software Development

The Agile Manifesto is based on four values and twelve principles that form the basis, for methodologies.



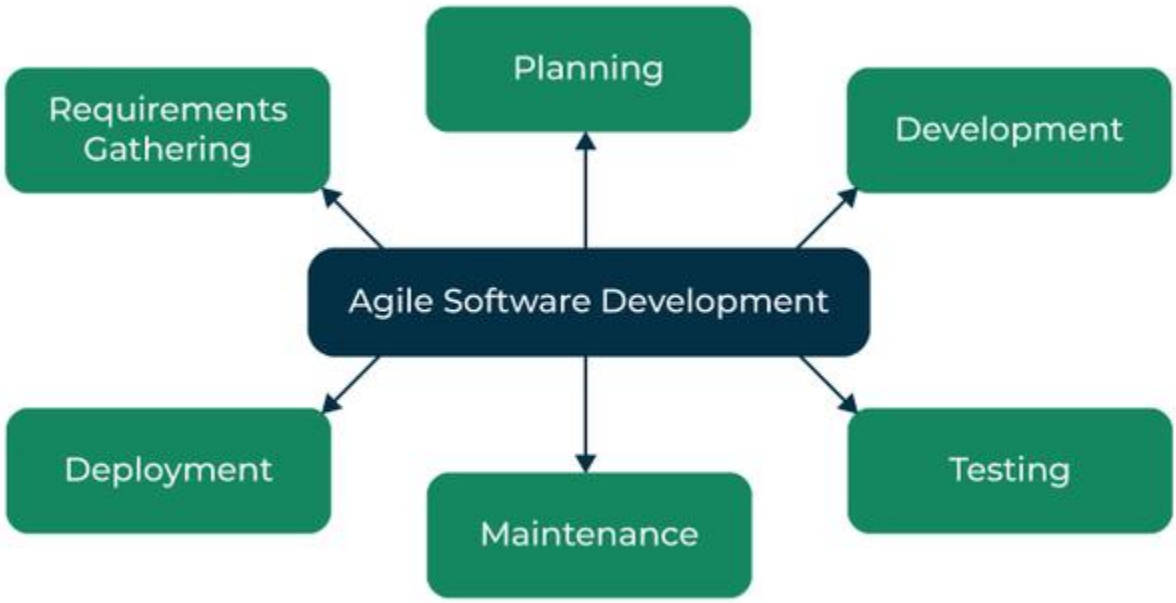
12 Principles of Agile Methodology

12 Principles of Agile Methodology

These principles include:

1. Ensuring customer satisfaction through the early delivery of software.
2. Being open to changing requirements in the stages of the development.
3. Frequently delivering working software with a main focus on preference for timeframes.
4. Promoting collaboration between business stakeholders and developers as an element.
5. Structuring the projects around individuals. Providing them with the necessary environment and support.
6. Prioritizing face to face communication whenever needed.
7. Considering working software as the measure of the progress.
8. Fostering development by allowing teams to maintain a pace indefinitely.
9. Placing attention on excellence and good design practices.
10. Recognizing the simplicity as crucial factor aiming to maximize productivity by minimizing the work.
11. Encouraging self organizing teams as the approach to design and build systems.
12. Regularly reflecting on how to enhance effectiveness and to make adjustments accordingly.

The Agile Software Development Process



Agile Software Development

1. **Requirements Gathering:** The customer's requirements for the software are gathered and prioritized.
2. **Planning:** The development team creates a plan for delivering the software, including the features that will be delivered in each iteration.

3. **Development:** The development team works to build the software, using frequent and rapid iterations.
4. **Testing:** The software is thoroughly tested to ensure that it meets the customer's requirements and is of high quality.
5. **Deployment:** The software is deployed and put into use.
6. **Maintenance:** The software is maintained to ensure that it continues to meet the customer's needs and expectations.

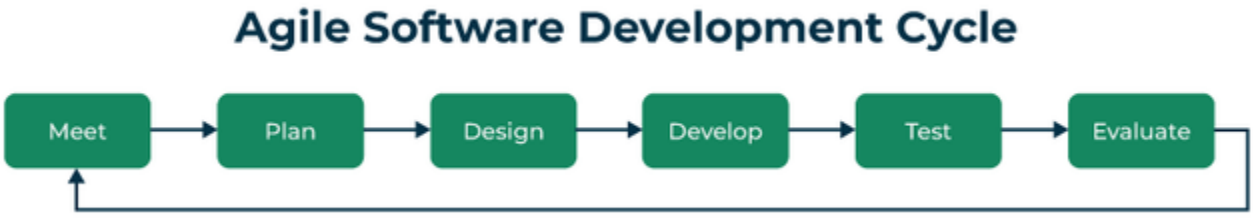
Agile Software Development is widely used by software development teams and is considered to be a flexible and adaptable approach to software development that is well-suited to changing requirements and the fast pace of software development.

Agile is a time-bound, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver all at once.

Agile Software development cycle

Let's see a brief overview of how development occurs in Agile philosophy.

1. concept
2. inception
3. iteration/construction
4. release
5. production
6. retirement



Agile software development cycle

- **Step** 1: In the first step, concept, and business opportunities in each possible project are identified and the amount of time and work needed to complete the project is estimated. Based on their technical and financial viability, projects can then be prioritized and determined which ones are worthwhile pursuing.
- **Step** 2: In the second phase, known as inception, the customer is consulted regarding the initial requirements, team members are selected, and funding is secured. Additionally, a schedule outlining each team's responsibilities and the precise time at which each sprint's work is expected to be finished should be developed.
- **Step** 3: Teams begin building functional software in the third step, iteration/construction, based on requirements and ongoing feedback. Iterations, also known as single development cycles, are the foundation of the Agile software development cycle.

Design Process of Agile software Development

- In Agile development, Design and Implementation are considered to be the central activities in the software process.
- The design and Implementation phase also incorporates other activities such as requirements elicitation and testing.

- In an agile approach, iteration occurs across activities. Therefore, the requirements and the design are developed together, rather than separately.
- The allocation of requirements and the design planning and development as executed in a series of increments. In contrast with the conventional model, where requirements gathering needs to be completed to proceed to the design and development phase, it gives Agile development an extra level of flexibility.
- An agile process focuses more on code development rather than documentation.

10 months. The company's head assigned two teams

Advantages Agile Software Development

- Deployment of software is quicker and thus helps in increasing the trust of the customer.
- Can better adapt to rapidly changing requirements and respond faster.
- Helps in getting immediate feedback which can be used to improve the software in the next increment.
- People – Not Process. People and interactions are given a higher priority than processes and tools.
- Continuous attention to technical excellence and good design.
- **Increased collaboration and communication:** Agile Software Development Methodology emphasize collaboration and communication among team members, stakeholders, and customers. This leads to improved understanding, better alignment, and increased buy-in from everyone involved.
- **Flexibility and adaptability:** Agile methodologies are designed to be flexible and adaptable, making it easier to respond to changes in requirements, priorities, or market conditions. This allows teams to quickly adjust their approach and stay focused on delivering value.
- **Improved quality and reliability:** Agile methodologies place a strong emphasis on testing, quality assurance, and continuous improvement. This helps to ensure that software is delivered with high quality and reliability, reducing the risk of defects or issues that can impact the user experience.
- **Enhanced customer satisfaction:** Agile methodologies prioritize customer satisfaction and focus on delivering value to the customer. By involving customers throughout the development process, teams can ensure that the software meets their needs and expectations.
- **Increased team morale and motivation:** Agile methodologies promote a collaborative, supportive, and positive work environment. This can lead to increased team morale, motivation, and engagement, which can in turn lead to better productivity, higher quality work, and improved outcomes.

Disadvantages Agile Software Development

- In the case of large software projects, it is difficult to assess the effort required at the initial stages of the software development life cycle.
- Agile Development is more code-focused and produces less documentation.
- Agile development is heavily dependent on the inputs of the customer. If the customer has ambiguity in his vision of the outcome, it is highly likely that the project to get off track.
- Face-to-face communication is harder in large-scale organizations.
- Only senior programmers are capable of making the kind of decisions required during the development process. Hence, it's a difficult situation for new programmers to adapt to the environment.

- **Lack of predictability:** Agile Development relies heavily on customer feedback and continuous iteration, which can make it difficult to predict project outcomes, timelines, and budgets.
- Limited scope control: Agile Development is designed to be flexible and adaptable, which means that scope changes can be easily accommodated. However, this can also lead to scope creep and a lack of control over the project scope.
- **Lack of emphasis on testing:** Agile Development places a greater emphasis on delivering working code quickly, which can lead to a lack of focus on testing and quality assurance. This can result in bugs and other issues that may go undetected until later stages of the project.
- **Risk of team burnout:** Agile Development can be intense and fast-paced, with frequent sprints and deadlines. This can put a lot of pressure on team members and lead to burnout, especially if the team is not given adequate time for rest and recovery.
- **Lack of structure and governance:** Agile Development is often less formal and structured than other development methodologies, which can lead to a lack of governance and oversight. This can result in inconsistent processes and practices, which can impact project quality and outcomes.

Agile is a framework that defines how software development needs to be carried on. Agile is not a single method, it represents the various collection of methods and practices that follow the value statements provided in the manifesto. Agile methods and practices do not promise to solve every problem present in the software industry (No Software model ever can). But they sure help to establish a culture and environment where solutions emerge.
Agile software development is an iterative and incremental approach to software development. It emphasizes collaboration between the development team and the customer, flexibility, and adaptability in the face of changing requirements, and the delivery of working software in short iterations.
The Agile Manifesto, which outlines the principles of agile development, values individuals and interactions, working software, customer collaboration, and response to change.