UNIT-I

INTRODUCTION TO SOFTWARE ENGINEERING

Introduction

Software is a program or set of programs containing instructions that provide desired functionality. And Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

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. Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.

Software engineering ethics

Like other engineering disciplines, software engineering is carried out within a social and legal framework that limits the freedom of people working in that area. As a software engineer, you must accept that your job involves wider responsibilities than simply the application of technical skills.

You must also behave in an ethical and morally responsible way if you are to be respected as a professional engineer. It goes without saying that you should uphold normal standards of honesty and integrity. You should not use your skills and abilities to behave in a dishonest way or in a way that will bring disrepute to the software engineering profession. However, there are areas where standards of acceptable behavior are not bound by laws but by the more tenuous notion of professional responsibility.

Some of these are:

1) Confidentiality You should normally respect the confidentiality of your employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
2) Competence You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3) Intellectual property rights You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.

4) Computer misuse You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses or other malware)

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following **Eight Principles:**

1) **Public** — Software engineers shall act consistently with the public interest.
2) **Client And Employer** — Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3) **Product** — Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4) **Judgment** — Software engineers shall maintain integrity and independence in their professional judgment.
5) **Management** — Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6) **Profession** — Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7) **Colleagues** — Software engineers shall be fair to and supportive of their colleagues.
8) **Self** — Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Software process models

A software process is a set of related activities that leads to the production of a software product. These activities may involve the development of software from scratch in a standard programming language like Java or C.

However, business applications are not necessarily developed in this way. New business software is now often developed by extending and modifying existing systems or by configuring and integrating off-the-shelf software or system components.

There are many different software processes but all must include four activities that are fundamental to software engineering:

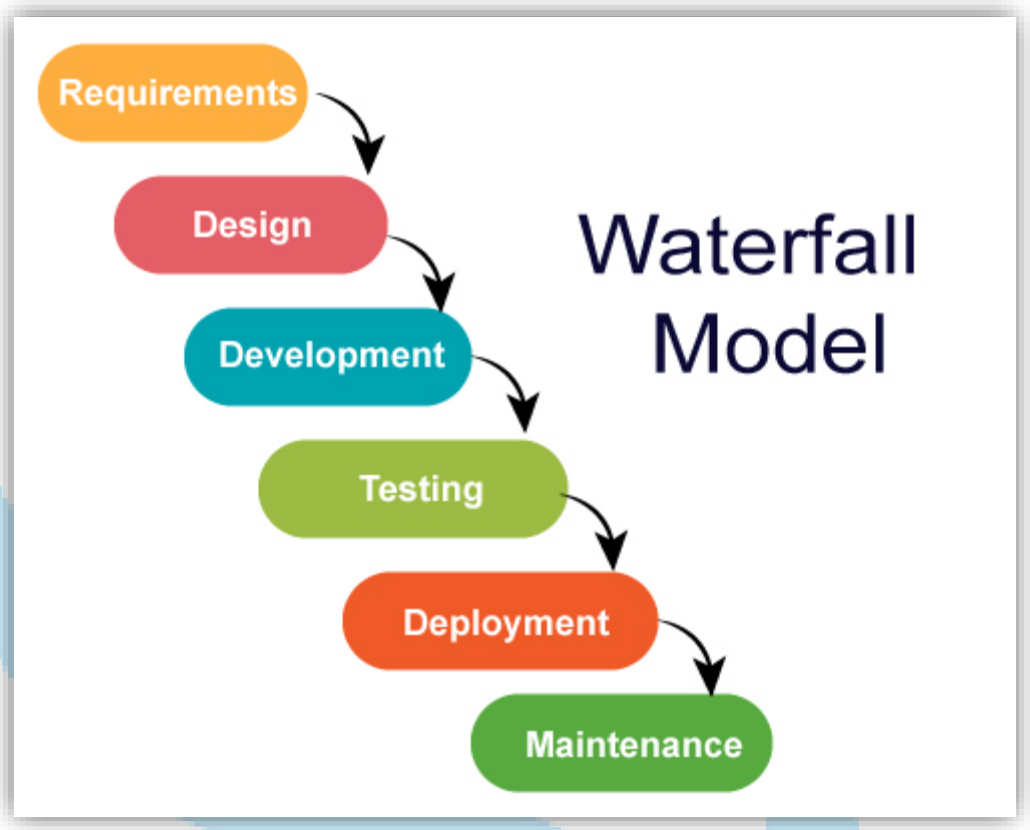
1) **Software specification** -The functionality of the software and constraints on its operation must be defined.
2) **Software design and implementation -** The software to meet the specification must be produced.
3) **Software validation -** The software must be validated to ensure that it does what the customer wants.
4) **Software evolution -** The software must evolve to meet changing customer needs.

When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc., and the ordering of these activities. However, as well as activities, process descriptions may also include:

1) **Products**, which are the outcomes of a process activity. For example, the outcome of the activity of architectural design may be a model of the software architecture.
2) **Roles**, which reflect the responsibilities of the people involved in the process. Examples of roles are project manager, configuration manager, programmer, etc.
3) **Pre and post-conditions**, which are statements that are true before and after a process activity has been enacted or a product produced. For example, before architectural design begins, a pre-condition may be that all requirements have been approved by the customer; after this activity is finished, a post-condition might be that the UML models describing the architecture have been reviewed.

Waterfall Model

The first published model of the software development process was derived from more general system engineering processes (Royce, 1970). This model is illustrated in Figure. Because of the cascade from one phase to another, this model is known as the 'waterfall model' or software life cycle. The waterfall model is an example of a plan-driven process—in principle, you must plan and schedule all of the process activities before starting work on them.



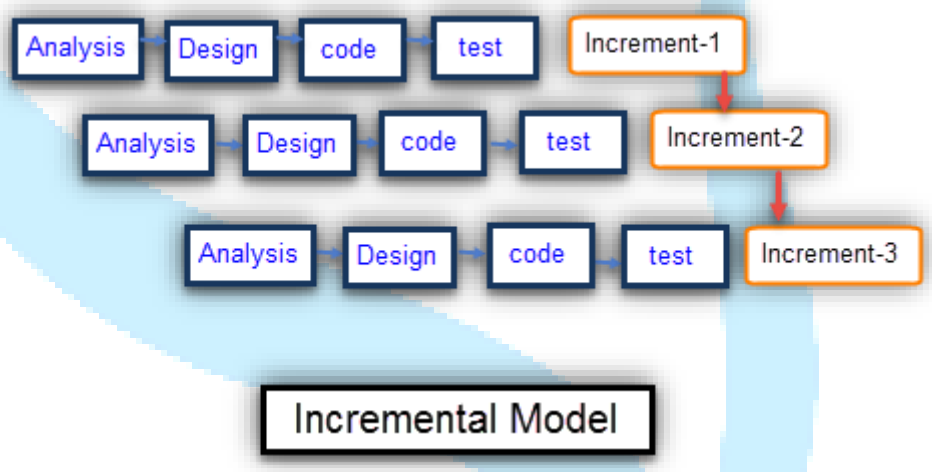
The principal stages of the waterfall model directly reflect the fundamental development activities:

1) **Requirements analysis and definition** - The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.
2) **System and software design** - The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.
3) **Implementation and unit testing** - During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.
4) **Integration and system testing** - The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
5) **Operation and maintenance Normally** (although not necessarily) - this is the longest life cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the

implementation of system units and enhancing the system's services as new requirements are discovered.

Incremental development

Incremental development is based on the idea of developing an initial implementation, exposing this to user comment and evolving it through several versions until an adequate system has been developed. In figure Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities.



Incremental Model

Incremental development has three important benefits, compared to the waterfall model:

1) The cost of accommodating changing customer requirements is reduced. The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
2) It is easier to get customer feedback on the development work that has been done. Customers can comment on demonstrations of the software and see how much has been implemented. Customers find it difficult to judge progress from software design documents.
3) More rapid delivery and deployment of useful software to the customer is possible, even if all of the functionality has not been included. Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development in some form is now the most common approach for the development of application systems. This approach can be either plan-driven, agile, or, more usually, a mixture of these approaches. In a plan-driven approach, the system increments are identified in advance; if an agile approach is adopted, the early increments are identified but the development of later increments depends on progress and customer priorities.

From a management perspective, the incremental approach has two problems:

1) The process is not visible. Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
2) System structure tends to degrade as new increments are added. Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

Process activities

Real software processes are interleaved sequences of technical, collaborative, and managerial activities with the overall goal of specifying, designing, implementing, and testing a software system. Generally, processes are now tool-supported. This means that software developers may use a range of software tools to help them, such as requirements management systems, design model editors, program editors, automated testing tools, and debuggers.

The four basic process activities of **specification, development, validation, and evolution** are organized differently in different development processes.

In the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved. How these activities are carried out depends on the type of software being developed, the experience and competence of the developers, and the type of the organization developing the software.

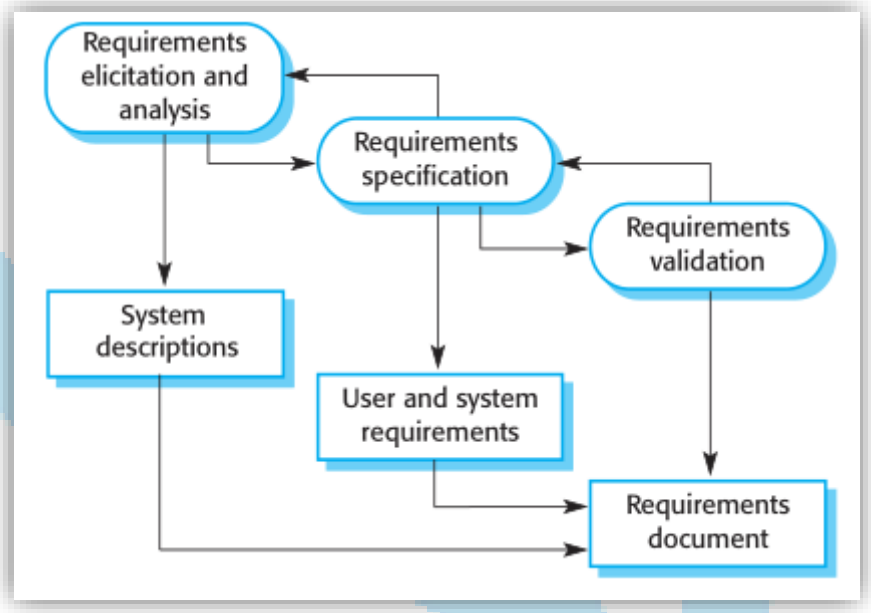
1)Software specification

Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.

- Requirements engineering is a particularly critical stage of the software process, as mistakes made at this stage inevitably lead to later problems in the system design and implementation.
- Before the requirements engineering process starts, a company may carry out a feasibility or marketing study to assess whether or not there is a need or a market for the software and whether or not it is technically and financially realistic to develop the software required.
- Feasibility studies are short-term, relatively cheap studies that inform the decision of whether or not to go ahead with a more detailed analysis.

The requirements engineering process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements. Requirements are usually presented at two levels of detail.

End-users and customers need a high-level statement of the requirements; system developers need a more detailed system specification.



There are three main activities in the requirements engineering process:

1) **Requirements elicitation and analysis -** This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.
2) **Requirements specification** - Requirements specification is the activity of translating the information gathered during requirements analysis into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; system requirements are a more detailed description of the functionality to be provided.
3) **Requirements validation** -This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

Requirements analysis continues during definition and specification, and new requirements come to light throughout the process. Therefore, the activities of analysis, definition, and specification are interleaved.

In agile methods, requirements specification is not a separate activity but is seen as part of system development. Requirements are informally specified for each increment of the system just before that increment is developed. Requirements are specified according to user priorities. The elicitation of requirements comes from users who are part of or work closely with the development team.

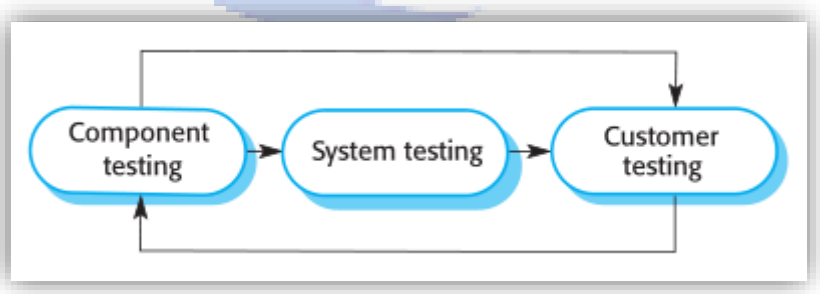
2)Software design and implementation

A software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used. Designers do not arrive at a finished design immediately but develop the design in stages. They add detail as they develop their design, with constant backtracking to modify earlier designs.

3)Software validation

Software validation or, more generally, verification and validation (V & V) is intended to show that a system both conforms to its specification and meets the expectations of the system customer. Program testing, where the system is executed using simulated test data, is the principal validation technique. Validation may also involve checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development.

However, most V & V time and effort is spent on program testing. Except for small programs, systems should not be tested as a single, monolithic unit. Figure shows a three-stage testing process in which system components are individually tested, then the integrated system is tested.



For custom software, customer testing involves testing the system with real customer data. For products that are sold as applications, customer testing is sometimes called **beta testing** where selected users try out and comment on the software.

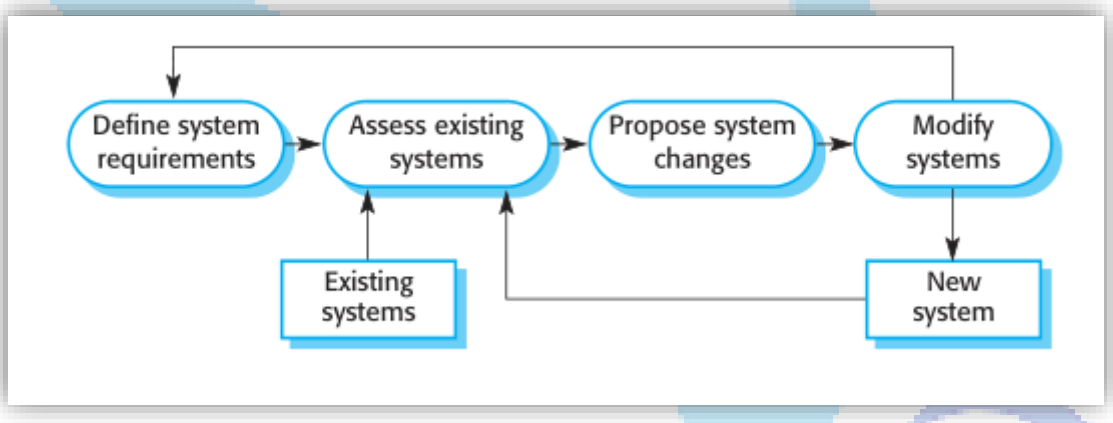
1) **Component testing -** The components making up the system are tested by the people developing the system. Each component is tested independently, without other system components. Components may be simple entities such as functions or object classes or may be coherent groupings of these entities. Test automation tools, such as JUnit for Java, that can rerun tests when new versions of the component are created, are commonly used.
2) **System testing -** System components are integrated to create a complete system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems.
    - It is also concerned with showing that the system meets its functional and non-functional requirements, and testing the emergent system properties.
    - For large systems, this may be a multistage process where components are integrated to form subsystems that are individually tested before these subsystems are integrated to form the final system.
3) **Customer testing** - This is the final stage in the testing process before the system is accepted for operational use. The system is tested by the system customer (or potential customer) rather than with simulated test data.
    - For custom-built software, customer testing may reveal errors and omissions in the system requirements definition, because the real data exercise the system in different ways from the test data.
    - Customer testing may also reveal requirements problems where the system's facilities do not really meet the users' needs or the system performance is unacceptable. For products, customer testing shows how well the software product meets the customer's needs.

4)Software evolution

The flexibility of software is one of the main reasons why more and more software is being incorporated into large, complex systems. Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design. However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware.

Historically, there has always been a split between the process of software development and the process of software evolution (software maintenance). People think of software development as a creative activity in which a software system is developed from an initial concept through to a working system. However, they sometimes think of software maintenance as dull and uninteresting. They think that software maintenance is less interesting and challenging than original software development.

This distinction between development and maintenance is increasingly irrelevant. Very few software systems are completely new systems, and it makes much more sense to see development and maintenance as a continuum. Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs.



Coping with Change

Change is inevitable in all large software projects. The system requirements change as businesses respond to external pressures, competition, and changed management priorities. As new technologies become available, new approaches to design and implementation become possible. Therefore, whatever software process model is used, it is essential that it can accommodate changes to the software being developed.

Change adds to the costs of software development because it usually means that work that has been completed has to be redone. This is called rework. For example, if the relationships between the requirements in a system have been analyzed and new requirements are then identified, some or all of the requirement's analysis has to be repeated. It may then be necessary to redesign the system to deliver the new requirements, change any programs that have been developed, and retest the system.

Two related approaches may be used to reduce the costs of rework:

1) **Change anticipation**, where the software process includes activities that can anticipate or predict possible changes before significant rework is required. For example, a prototype system may be developed to show some key features of the system to customers. They can experiment with the prototype and refine their requirements before committing to high software production costs.
2) **Change tolerance**, where the process and software are designed so that changes can be easily made to the system. This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have to be altered to incorporate the change.

In this section, we discuss two ways of coping with change and changing system requirements:

1) **System prototyping**, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This is a method of change anticipation as it allows users to experiment with the system before delivery and so refine their requirements. The number of requirements change proposals made after delivery is therefore likely to be reduced.
2) **Incremental delivery**, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance. It avoids the premature commitment to requirements for the whole system and allows changes to be incorporated into later increments at relatively low cost.

The notion of refactoring, namely, improving the structure and organization of a program, is also an important mechanism that supports change tolerance.

Prototyping

A prototype is an early version of a software system that is used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions. Rapid, iterative development of the prototype is essential so that costs are controlled and system stakeholders can experiment with the prototype early in the software process.

A software prototype can be used in a software development process to help anticipate changes that may be required:

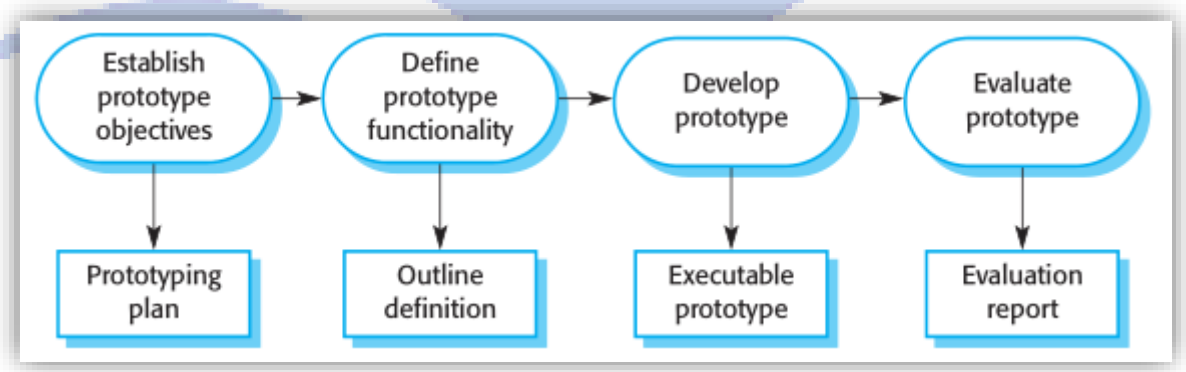
1) In the requirements engineering process, a prototype can help with the elicitation and validation of system requirements.
2) In the system design process, a prototype can be used to explore software solutions and in the development of a user interface for the system.

System prototypes allow potential users to see how well the system supports their work. They may get new ideas for requirements and find areas of strength and weakness in the software.

- They may then propose new system requirements. Furthermore, as the prototype is developed, it may reveal errors and omissions in the system requirements.
- A feature described in a specification may seem to be clear and useful. However, when that function is combined with other functions, users often find that their initial view was incorrect or incomplete. The system specification can then be modified to reflect the changed understanding of the requirements.

A **System prototype** may be used while the system is being designed to carry out design experiments to check the feasibility of a proposed design. For example, a database design may be prototyped and tested to check that it supports efficient data access for the most common user queries.

Rapid prototyping with end-user involvement is the only sensible way to develop user interfaces. Because of the dynamic nature of user interfaces, textual descriptions and diagrams are not good enough for expressing the user interface requirements and design.



A process model for prototype development is shown in Figure above.

The objectives of prototyping should be made explicit from the start of the process.

These may be to develop the user interface, to develop a system to validate functional system requirements, or to develop a system to demonstrate the application to managers. The same prototype usually cannot meet all objectives.

- If the objectives are left unstated, management or end-users may misunderstand the function of the prototype. Consequently, they may not get the benefits that they expected from the prototype development.

The **next stage** in the process is to decide what to put into and, perhaps more importantly, what to leave out of the prototype system. To reduce prototyping costs and accelerate the delivery schedule, you may leave some functionality out of the prototype.

- You may decide to relax non-functional requirements such as response time and memory utilization. Error handling and management may be ignored unless the objective of the prototype is to establish a user interface. Standards of reliability and program quality may be reduced.

The **final stage** of the process is prototype evaluation. Provision must be made during this stage for user training, and the prototype objectives should be used to derive a plan for evaluation. Potential users need time to become comfortable with a new system and to settle into a normal pattern of usage.

- Once they are using the system normally, they then discover requirements errors and omissions. A general problem with prototyping is that users may not use the prototype in the same way as they use the final system. Prototype testers may not be typical of system users.
- There may not be enough time to train users during prototype evaluation. If the prototype is slow, the evaluators may adjust their way of working and avoid those system features that have slow response times. When provided with better response in the final system, they may use it in a different way.

Agile software development

In the 1980s and early 1990s, there was a widespread view that the best way to achieve better software was through careful project planning, formalized quality assurance, the use of analysis and design methods supported by CASE tools, and controlled and rigorous software development processes. This view came from the software engineering community that was responsible for developing large, long-lived software systems such as aerospace and government .

This software was developed by large teams working for different companies. Teams were often geographically dispersed and worked on the software for long periods of time. An example of this type of software is the control systems for a modern aircraft, which might take up to 10 years from initial specification to deployment.

These plan driven approaches involve a significant overhead in planning, designing, and documenting the system. This overhead is justified when the work of multiple development teams has to be coordinated, when the system is a critical system, and when many different people will be involved in maintaining the software over its lifetime.

The meaning of Agile is swift or versatile. "**Agile process model**" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning.

- The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.
- Each iteration is considered as a short time **"frame"** in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements.
- Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

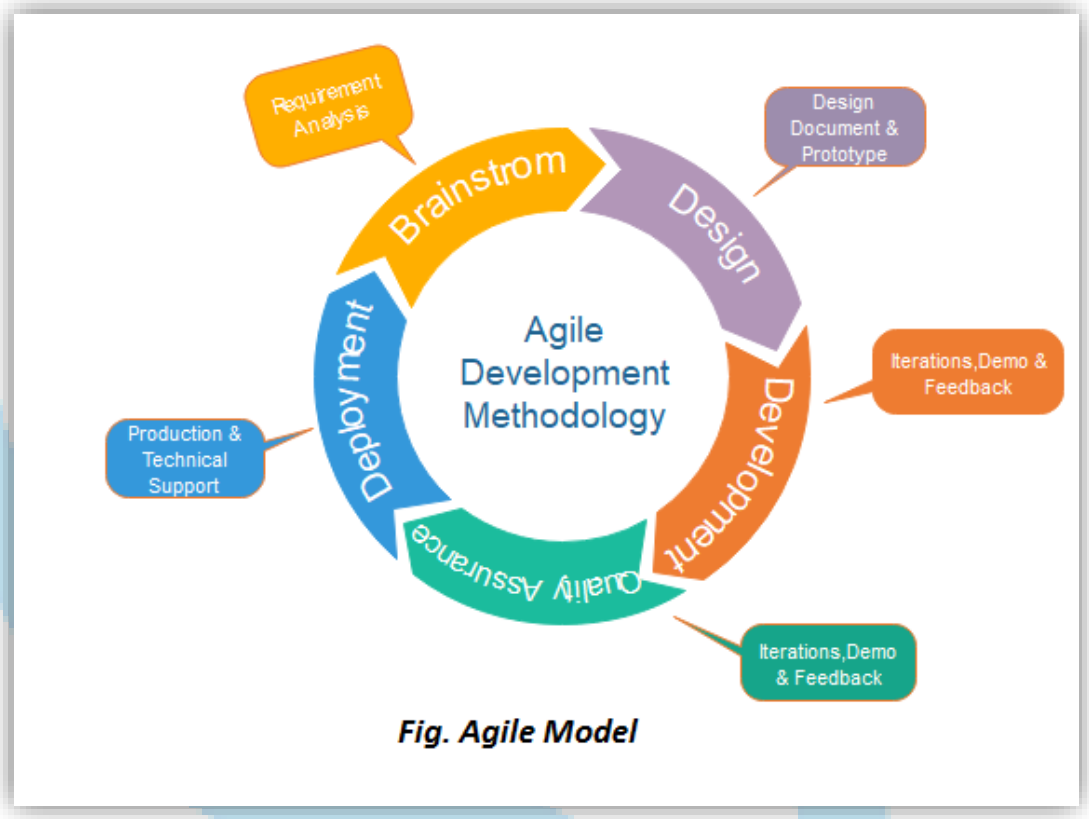


Fig. Agile Model

Phases of Agile Model:

Following are the phases in the Agile model are as follows:

- ✓ Requirements gathering
- ✓ Design the requirements
- ✓ Construction/ iteration
- ✓ Testing/ Quality assurance
- ✓ Deployment
- ✓ Feedback

1) **Requirements gathering:** In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.
2) **Design the requirements:** When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.
3) **Construction/ iteration:** When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4) **Testing:** In this phase, the Quality Assurance team examines the product's performance and looks for the bug.
5) **Deployment:** In this phase, the team issues a product for the user's work environment.
6) **Feedback:** After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Advantages of Agile Methodology

1) Customer satisfaction is rapid, continuous development and delivery of useful software.
2) Customer, Developer, and Product Owner interact regularly to emphasize rather than processes and tools.
3) Product is developed fast and frequently delivered (weeks rather than months.)
4) A face-to-face conversation is the best form of communication.
5) It continuously gave attention to technical excellence and good design.
6) Daily and close cooperation between business people and developers.
7) Regular adaptation to changing circumstances.
8) Even late changes in requirements are welcomed.

Agile methods

- Scrum
- Crystal
- Dynamic Software Development Method(DSDM)
- Feature Driven Development(FDD)
- Lean Software Development
- eXtreme Programming(XP)

Scrum

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

Scrum is a lightweight Agile framework that can be used by project managers to control all types of iterative and incremental projects. In Scrum, the product owner creates a product backlog that allows them to work with their team to identify and prioritize system functionality. The product backlog is a list of everything that needs to be accomplished to deliver a successful, working software system -- this includes bug fixes, features and non-functional requirements.

Once the product backlog is defined, no additional functionality can be added except by the corresponding team.

Once the team and the product owner have established the priorities, cross-functional teams step in and agree to deliver working increments of software during each sprint -- often within 30 days. After each sprint, the product backlog is reevaluated, analyzed and reprioritized in order to select a new set of deliverable functions for the next sprint. Scrum has gained popularity over the years since it is simple, has proven to be productive and can incorporate the various overarching practices promoted by the other Agile methods.

There are three roles in it, and their responsibilities are:

- **Scrum Master:** The scrum can set up the master team, arrange the meeting and remove obstacles for the process
- **Product owner:** The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.
- **Scrum Team:** The team manages its work and organizes the work to complete the sprint or cycle.

eXtreme Programming(XP)

This type of methodology is used when customers are constantly changing demands or requirements, or when they are not sure about the system's performance.

The XP method is based on the values of communication, feedback, simplicity and courage. Customers work closely with their development team to define and prioritize their requested user stories. However, it is up to the team to deliver the highest priority user stories in the form of working software that has been tested at each iteration. To maximize productivity, the XP method provides users with a supportive, lightweight framework that guides them and helps ensure the release of high-quality enterprise software.

Crystal

There are three concepts of this method-

1) Chartering: Multi activities are involved in this phase such as making a development team, performing feasibility analysis, developing plans, etc.

2) Cyclic delivery: under this, two more cycles consist, these are:
   - Team updates the release plan.
   - Integrated product delivers to the users.

3)Wrap up: According to the user environment, this phase performs deployment, post-deployment.

Plan- driven and agile development

Plan-driven and agile development are two contrasting approaches to software development. Each has its own set of principles, practices, and methodologies, and they are suited to different project contexts and goals. Here's an overview of both approaches:

Plan-Driven Development:

1) **Waterfall Model**: The Waterfall model is a classic example of plan-driven development. It follows a linear and sequential approach, where each phase must be completed before moving on to the next. The phases typically include requirements analysis, design, implementation, testing, and maintenance.
2) **Detailed Documentation**: Plan-driven development places a strong emphasis on comprehensive documentation, including detailed requirements specifications, design documents, and test plans. This documentation helps in minimizing misunderstandings and managing project scope.
3) **Predictability**: Plan-driven development aims to provide predictability in terms of project timelines, costs, and scope. It often relies on detailed project plans and Gantt charts.
4) **Change Control**: Changes to requirements and scope are typically discouraged in plan-driven development because they can disrupt the carefully planned project timeline. Changes are managed through a formal change control process.
5) **Well-suited for Stable Requirements**: Plan-driven approaches work best when the project's requirements are well-understood and unlikely to change significantly during development. They are commonly used in industries with strict regulatory requirements, such as aerospace and defense.

Agile Development:

1) **Iterative and Incremental:** Agile development is characterized by an iterative and incremental approach. It focuses on delivering working software in small, frequent increments, allowing for feedback and adaptation throughout the development process.
2) **Collaboration and Communication:** Agile places a strong emphasis on collaboration between cross-functional teams and close communication with stakeholders. This helps in responding to changing requirements and priorities.
3) **Responding to Change:** Agile embraces change and views it as an opportunity. It's well-suited for projects where requirements are not fully known upfront or are expected to change over time.
4) **Customer-Centric:** Agile development prioritizes delivering value to the customer early and often. Customer feedback is integral to the development process, and adjustments are made based on this feedback.
5) **Scrum, Kanban, and more:** Agile methodologies such as Scrum, Kanban, and Extreme Programming (XP) provide specific frameworks and practices for implementing agile principles.
6) **Well-suited for Dynamic Environments:** Agile is often used in software development contexts where requirements are subject to change, and where rapid delivery and adaptability are essential.

In practice, many software development projects use a combination of plan-driven and agile elements, selecting the approach that best fits the project's characteristics and constraints. This approach, sometimes called "hybrid" development, combines the structured planning and documentation of plan-driven methods with the flexibility and customer-centric focus of agile practices to find a balance that suits the project's needs.