UNIT – II: PROCESS MODELS

Software Process Model

A software process model is a **structured representation of the activities of the software development process**. During the development of software, **various steps that are important for the successful development of the project** are taken and if we **structured them according to the proper order in a model** then it is called a software process model. The software process model includes various activities such as steps like planning, designing, implementation, defining tasks, setting up milestones, roles, and responsibilities, etc.

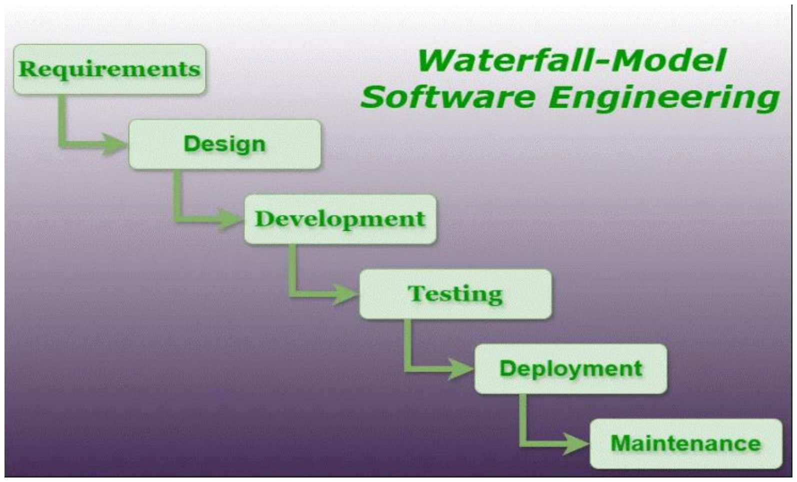
Waterfall Model

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.

Phases of Waterfall Model

Classical Waterfall Model divides the life cycle into a **set of phases.** The development process can be considered as a sequential flow in the waterfall. The different sequential phases of the classical waterfall model are follow:



Requirements Analysis and Specification

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.

1.1 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).

1.2. 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.

2. Design

The goal of this **Software Design 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).

- **High-Level Design (HLD)**: This phase focuses on **outlining the broad structure of the system**. It **highlights the key components** and **how they interact with each other**, giving a **clear overview of the system's architecture**.
- **Low-Level Design (LLD)**: Once the high-level design is in place, **this phase zooms into the details. It breaks down each component into smaller parts** and provides specifics about **how each part will function, guiding the actual coding process**.

3. Development

In the **Development 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.**

- In this phase, **developers begin writing the actual source code based on the designs created earlier.**
- The goal is **to transform the design into working code** using the most suitable programming languages.
- **Unit tests** are often performed during this phase **to make sure that each component functions correctly on its own.**

4. Testing and Deployment

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** is carried out on this. 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. Deployment:

Once the software has been thoroughly tested, it's time **to deploy it to the customer or end-users.** This means making the software **ready and available for use**, often by **moving it to a live or staging environment.**

During this phase, we also **focus on helping users get comfortable with the software by providing training, setting up necessary environments, and ensuring everything is running smoothly.** The goal is to make sure the system works as **expected in real-world conditions and that users can start using it without any hitches.**

7. Maintenance

In **Maintenance Phase** 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.

Features of 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 **depended 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 **careful planning process**, where the **project scope, timelines, and deliverables are carefully defined and monitored** throughout the project lifecycle.

Importance of 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 phases each have **unique inputs and outputs, guaranteeing a planned development with clear checkpoints.**
3. **Documentation:** A focus on thorough documentation helps with **software understanding**, **maintenance, and future growth.**

4. **Stability in Requirements:** Suitable for projects when the requirements are **clear and stable, 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.**

Incremental Process Model

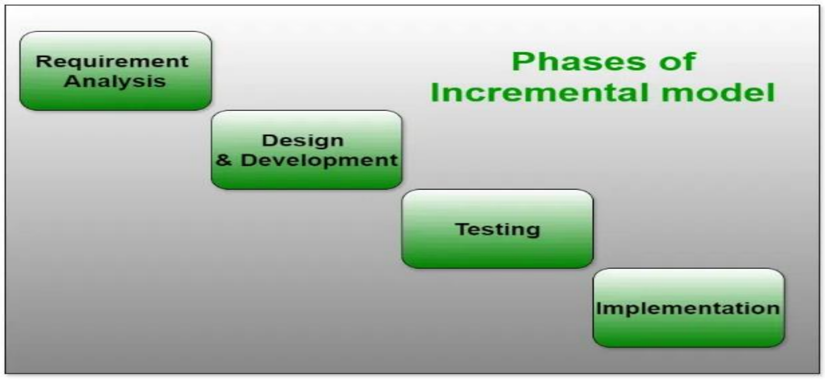
The Incremental Process Model is a method of software development where the **system is built step by step**. Instead of **delivering the whole system at once**, it is **developed** and **delivered in small parts** called **increments**. **Each increment builds upon** the **previous one by adding new functionality**, until the **complete system is finished**.

Key Characteristics of Incremental Process Model

- **Partial System Delivery:** The system is **developed and delivered in small**, manageable pieces. Each part **adds new features to the previous version**.
- **Early Functionality: Basic functionality** is available **early in the project**. This allows **users to start** using and **testing the system quickly**.
- **Customer Feedback Loop**: **Feedback is collected** after **each part is delivered**. This helps **improve** the **next version of the system**.
- **Flexible to Changes: Changes** or **new features** can be **added between increments**. This makes the model flexible to **evolving needs**.
- **Combination of Linear and Iterative Approaches: Combines the structured approach of Waterfall** with flexibility. Supports both **planning** and **ongoing improvements.**

Phases of the Incremental Model

The phases of Incremental model is divided into the four parts which is **Requirement**, **Design**, **Testing** and **Implementation** phases. In those phase, **the process continues until we got the expected output at the end.**



Phases of incremental model

Requirement Analysis

The first step in the Incremental Model is **understanding what the software needs** to do. The **team gathers the requirements** from **the product experts and clearly** defines the **system's functional needs.** This phase is important because it **sets the foundation for everything** else in the development process.

Design & Development

Next, the team focuses on **designing how the software will function** and **starts developing it**. They work on **adding new features** and **making sure the system works as expected**. The design and development steps go **hand-in-hand to build the functionality of the software.**

Testing

Once a feature is developed, it goes through testing. The testing phase **checks how the software performs, including both new and existing features.** The team **uses different testing methods** to make sure **everything is working correctly**.

Implementation

This phase involves **writing the final code based on the design and development steps.** After testing the functionality, the team **verify that everything is working as planned.** By the end of this phase, the product is **regularly improved and updated** until **it becomes the final working version**.

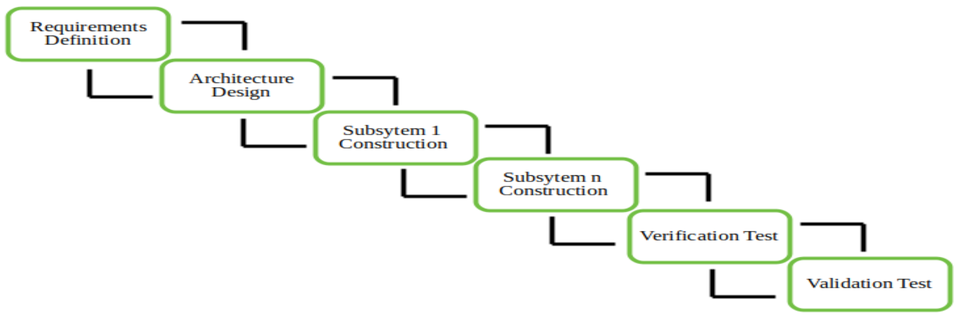
Types of Incremental Model

The Incremental Model has two main types, each offers different approaches to how software is developed in parts. Here are the two types:

Staged Delivery Model

The Staged Delivery Model **develops software in a sequence of planned stages**, where **each stage delivers a functional part of the system**. **Each release** brings the **product closer to completion,** allowing it to evolve regularly. Working **versions are delivered** at **regular intervals**, making **progress visible and manageable** throughout the **development process.**

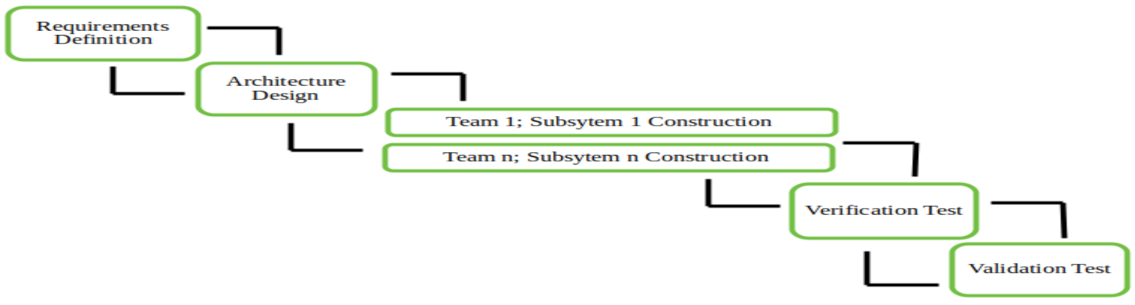
The diagram below shows this model :



Staged Delivery Model

Parallel Development Model

The Parallel Development Model **divides the system into multiple modules** that are **developed simultaneously at the same time by different teams**. By working on **separate components in parallel**, the **development process becomes faster** and more **efficient.** This approach **reduces overall project time** and allows **teams to focus on specific functionalities concurrently.** Given below is the diagram showing the model:



Parallel Development Model

Use Cases of Incremental Process Model

- **When the requirements are well-defined and clear:**
  Because increments can be planned and developed step-by-step with minimal requirement changes.
- **If the project has a long development timeline:**
  Incremental delivery helps manage complexity over time by breaking the project into smaller, manageable parts.
- **If the customer needs a quick product release:**
  You can deliver the most critical features early in the first increment, allowing the customer to start using the software sooner.
- **When you want to develop the most important features first:**
  This allows early feedback on key functionalities and better prioritization for subsequent increments.

Advantages of Incremental Process Model

The Incremental Model of software development builds a system in **small, manageable sections** (increments), making it a **good choice for many projects**. Below are the key advantages:

- **Faster Software Delivery**
  - **Initial working versions of the software** can be **delivered quickly**.
  - Early delivery **increases customer satisfaction** and feedback opportunities.
- **Clear Understanding for Clients**
  - Clients get **to see parts of the system at each stage.**
  - This visibility ensures that the **final product meets their expectations.**
- **Easy to Implement Changes**
  - Requirements can evolve, and **changes can be integrated in subsequent** increments.
  - It supports **flexibility without heavily disrupting earlier stages.**
- **Effective Risk Management**
  - **Risks can be identified and handled early** due to the staged approach.
  - **Each increment allows for testing and validation, reducing** the impact of unpredicted issues.

- **Flexible Criteria and Scope**
    - **Requirements** can be **adjusted without a major cost increase**.
    - Better scope management helps keep the project aligned with business goals.
- **Cost-Effective**
    - **Compared to models** like the **Waterfall**, the **incremental model** is generally more **cost-efficient**.
    - **Budget** is **spread across stages**, making **it easier to manage finances.**
- **Simpler Error Identification**
    - Since **development is done in parts**, it's **easier to pinpoint and fix errors within a specific increment**.
    - Testing each module separately enhances **quality and reliability.**

Disadvantages of Incremental Process Model

Incremental Model comes with some limitations and challenges. Below are the key disadvantages:

- **Requires a Skilled Team and Proper Planning**
    - Successful implementation demands an **experienced team**.
    - **Poor planning or coordination** can **lead** to **confusion between increments.**
- **Cost Can Increase Over Time**
    - **Due to repeated testing, redesign, and integration in every cycle,** the overall **project cost** may **rise.**
    - **Continuous iteration** involves **added overhead**
- **Incomplete Requirement Gathering Can Cause Design Issues**
    - **If all requirements** are **not identified early**, the **system architecture** may **not support future needs.**
    - Insufficient upfront design can **lead to rework** and architectural mismatches.
- **Lack of Smooth Flow Between Increments**
    - **Each iteration** may **function independently**, which can create inconsistencies.
    - There might be integration challenges when **combining all the increments** into a integrated product.
- **High Effort to Fix Repeated Issues**
    - **A defect in one increment** may **exist in others**.

- Fixing the **same issue across multiple units** can be **time-consuming** and **resource-intensive**

Evolutionary Process Model

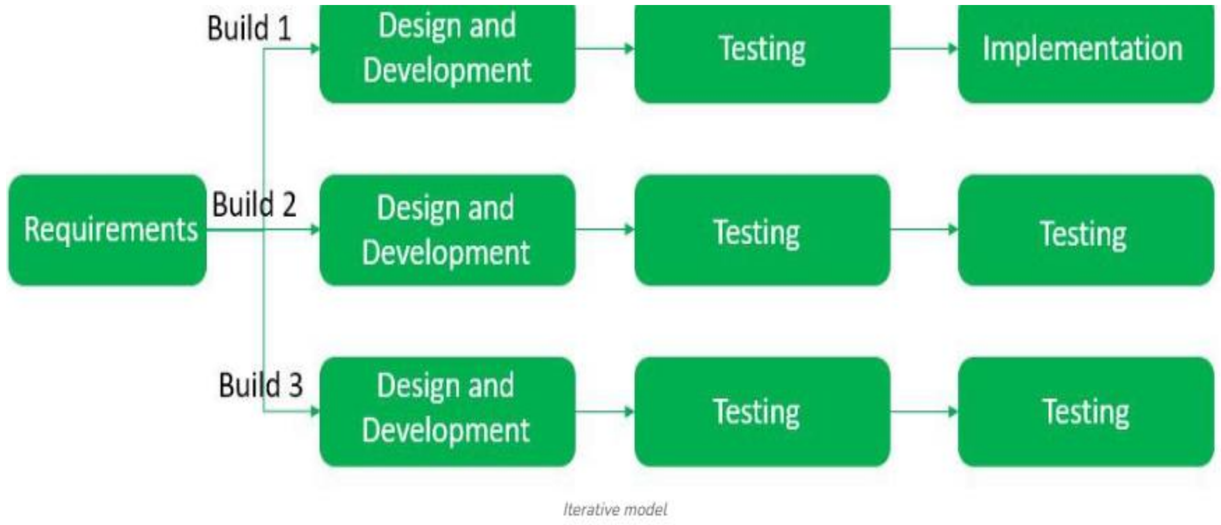
The evolutionary model is based on the concept of making an initial product and then evolving the software product over time with iterative and incremental approaches with proper feedback. In this type of model, the product will go through several iterations and come up when the final product is built through multiple iterations. The development is carried out simultaneously with the feedback during the development. This model has a number of advantages such as customer involvement, taking feedback from the customer during development, and building the exact product that the user wants. Because of the multiple iterations, the chances of errors get reduced and the reliability and efficiency will increase.

Types of Evolutionary Process Models

1. Iterative Model
2. Incremental Model
3. Spiral Model

1. Iterative Model

In the iterative model first, we take the initial requirements then we enhance the product over multiple iterations until the final product gets ready. In every iteration, some design modifications were made and some changes in functional requirements is added. The main idea behind this approach is **to build the final product through multiple iterations** that result in the **final product** being almost the same as the **user wants with fewer errors and the performance, and quality would be high.**



Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." This process may be described as an "evolutionary acquisition" or "incremental build" approach."

In this incremental model, the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.

The key to a successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests must be repeated and extended to verify each version of the software.

Advantages of the Iterative Model:

- **Early risk mitigation:** Potential problems can be identified and addressed early in the development process.
- **Adaptability to change:** The model allows for easier integration of changing requirements or feedback.
- **Improved quality:** Continuous feedback and testing lead to a more robust and refined product.
- **Faster time to market:** Working versions of the software are available in each iteration.
- **Increased stakeholder involvement:** Stakeholders can provide feedback throughout the development process.

Disadvantages of the Iterative Model:

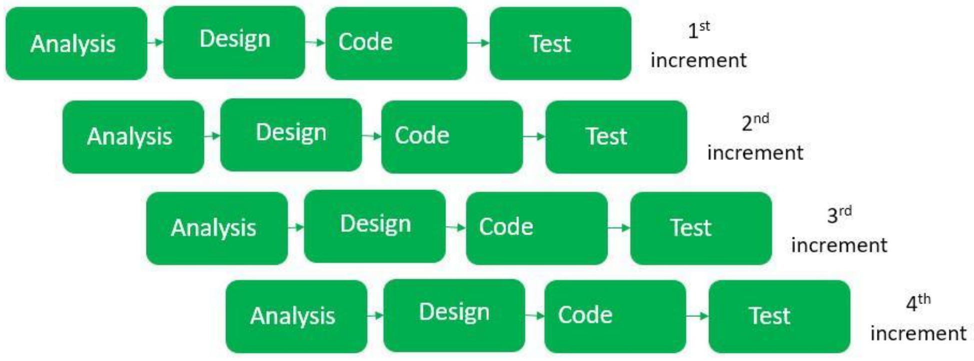
- **Increased complexity:** Managing iterations and ensuring consistency can be challenging.
- **Potential for scope creep:** Uncontrolled changes during iterations can lead to scope creep.
- **Higher initial costs:** Iterative development can be more resource-intensive.

Examples of Iterative Model:

- **Prototyping:** Creating and refining prototypes in iterative cycles.
- **Agile development methodologies:** Frameworks like Scrum and Kanban utilize iterative and incremental approaches.
- **Rational Unified Process (RUP):** RUP incorporates iterative and incremental development as a core principle.
- **Rapid Application Development (RAD):** RAD emphasizes rapid prototyping and iterative development.

2. Incremental Model

In the incremental model, we first build the project with basic features and then evolve the project in every iteration, it is mainly used for large projects. The first step is to gather the requirements and then perform analysis, design, code, and test and this process goes the same over and over again until our final project is ready.



Incremental Model

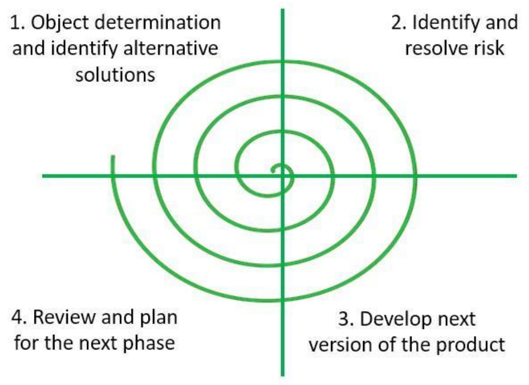
3. Spiral Model

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 **Stages 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.

Phases of the Spiral Model


1. Object determination and identify alternative solutions
2. Identify and resolve risk
3. Develop next version of the product
4. Review and plan for the next phase

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. Each phase of the Spiral Model is divided into four Quadrants:

1. Objectives Defined

In first phase of the spiral model we clarify what the project aims to achieve, including functional and non-functional requirements.

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. Risk Analysis and Resolving

In the risk analysis phase, the risks associated with the project are identified and evaluated.

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.

In the evaluation phase, the software is evaluated to determine if it meets the customer's requirements and if it is of high quality.

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.

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.

Advantages of the Evolutionary Process Model

1. During the development phase, the customer gives feedback regularly because the customer's requirement gets clearly specified.
2. After every iteration risk gets analyzed.
3. Suitable for big complex projects.
4. The first build gets delivered quickly as it used an iterative and incremental approach.
5. **Enhanced Flexibility:** The iterative nature of the model allows for continuous changes and refinements to be made, accommodating changing requirements effectively.
6. **Risk Reduction:** The model's emphasis on risk analysis during each iteration helps in identifying and mitigating potential issues early in the development process.
7. **Adaptable to Changes:** Since changes can be incorporated at the beginning of each iteration, it is well-suited for projects with evolving or uncertain requirements.
8. **Customer Collaboration:** Regular customer feedback throughout the development process ensures that the end product aligns more closely with the customer's needs and expectations.

Disadvantages of the Evolutionary Process Model

1. It is not suitable for small projects.
2. The complexity of the spiral model can be more than the other sequential models.
3. The cost of developing a product through a spiral model is high.
4. **Project Management Complexity:** The iterative nature of the model can make project management and tracking more complex compared to linear models.
5. **Resource Intensive:** The need for continuous iteration and customer feedback demands a higher level of resources, including time, personnel, and tools.

6. **Documentation Challenges:** Frequent changes and iterations can lead to challenges in maintaining accurate and up-to-date documentation.
7. **Potential Scope Creep:** The flexibility to accommodate changes can sometimes lead to an uncontrolled expansion of project scope, resulting in scope creep.
8. **Initial Planning Overhead:** The model's complexity requires a well-defined initial plan, and any deviations or adjustments can be time-consuming and costly.

Specialized process models

Specialized process models in software engineering are adaptations of traditional software development models that address particular project needs or challenges encountered with conventional methods. They are chosen when the project requires a specific approach due to factors such as size, complexity, technical hurdles, or features like security, reusability, or automation.

Here are some examples of these models with illustrative scenarios:

- **Component-Based Development (CBD) Model:**

  - Description: This model leverages pre-existing software components (like Commercial off-the-shelf software or COTS) to construct software, aiming for faster development and lower costs. It encourages modularity and reuse through a cycle of component identification, integration, customization, testing, deployment, and maintenance.

  - Example: A company developing a new Enterprise Resource Planning (ERP) system might opt for CBD, using pre-built components for common functions like accounting, human resources, or inventory management. This allows them to focus on developing unique business logic, reducing development time and effort. The reuse of these components also enhances the system's maintainability and scalability.

- **Formal Methods Model**:

  - **Description:** This model employs mathematical techniques and formal logic to specify, develop, and verify software systems, ensuring high reliability and correctness, particularly in critical applications. Formal methods use rigorous proofs and mathematical reasoning to ensure the system behaves as intended under all conditions, addressing issues like ambiguity, incompleteness, and inconsistency.

  - **Example**: Developing software for a nuclear reactor's control system, where errors can have catastrophic consequences, would benefit from formal methods. Engineers would use mathematical specifications to model the system's behavior and verify that it will respond correctly to any possible temperature input, including edge cases that might be missed by traditional testing methods.

- **Aspect-Oriented Software Development (AOSD) Model:**

  - Description: This model enhances modularity by separating crosscutting concerns, such as logging, security, or error handling, into independent modules called aspects. This separation improves code organization, reduces redundancy, and simplifies maintenance and updates.

  - Example: In a large banking application, a crosscutting concern like security might involve authentication, authorization, and data encryption. Using AOSD, a security aspect could encapsulate all security-related logic, which is then automatically applied across various modules of the application, such as user login, transaction processing, and data reporting. This makes the code base more organized and easier to manage.

Agile Process

Agile is a widely adopted methodology in software engineering that emphasizes an iterative, collaborative, and flexible approach to software development. It focuses on delivering working software frequently, incorporating customer feedback, and adapting to changing requirements. This differs from traditional, sequential approaches like the Waterfall model.

Here's a closer look at the agile process:

1. Core principles and values

Agile is guided by the Agile Manifesto, a document created in 2001 by a group of software developers. Its four core values are:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

These values are further supported by 12 principles that provide more concrete guidance.

2. Phases of the agile development lifecycle

The agile software development lifecycle (SDLC) typically includes six phases:

1. **Concept:** Defining the project vision, scope, and objectives.
2. **Inception**: Building the development team, designing the system architecture, and gathering detailed requirements.

3. **Iteration (Construction)**: Developers work in short cycles (sprints) to build and refine the software, delivering working increments.

4. **Release**: Testing and quality assurance, user training, and deployment of the software.

5. **Maintenanc**e: Ongoing support, bug fixes, and feature enhancements based on feedback.

6. **Retirement**: Replacing the software with a new system or phasing it out when it becomes outdated.

4. Agile methodologies

Several frameworks and methods are used to implement Agile principles, such as:

- **Scrum**: A popular framework that utilizes time-boxed iterations called "sprints," defined roles (Product Owner, Scrum Master, Development Team), and regular meetings (daily stand-ups, sprint reviews, retrospectives).

- **Kanban:** A visual project management system that focuses on workflow visualization, limiting work in progress (WIP), and maximizing efficiency.

- **Extreme Programming** (XP): A disciplined approach emphasizing practices like pair programming, test-driven development, and continuous integration to enhance software quality.

- **Lean Software Development**: A methodology focusing on maximizing customer value and eliminating waste in the development process.

- **Scaled Agile Framework (SAFe**): A framework that integrates Lean-Agile principles for scaling Agile practices across larger organizations and projects.

5. Key practices in agile software development

- **Iterative development**: Breaking down large projects into smaller, manageable chunks and delivering working software in increments.

- **Frequent communication and collaboration**: Encouraging regular interaction among team members, stakeholders, and customers.

- **Continuous feedback and improvement**: Actively seeking and incorporating feedback from users and stakeholders throughout the development cycle.

- **Self-organizing teams**: Empowering development teams to make decisions and manage their work.

- **Transparency**: Keeping the development process and progress visible to all stakeholders.

6. Benefits of the agile approach

- **Increased flexibility and adaptability**: Easily responding to changes in requirements, technology, and market conditions.

- **Faster delivery and quicker time-to-market**: Delivering functional software in shorter cycles.

- **Enhanced quality and customer satisfaction**: Continuous testing and feedback loops lead to better products that meet customer expectations.
- **Improved team collaboration and communication**: Fostering a more engaging and productive work environment.
- **Reduced risk**: Identifying and addressing potential issues early in the process.

7. Challenges in adopting agile

While Agile offers numerous benefits, implementing it successfully can pose challenges, including:

- **Resistance to change**: Overcoming ingrained habits and mindsets from traditional methodologies.
- **Lack of Agile experience**: Teams may need training and coaching to adapt to Agile principles and practices.
- **Inadequate communication**: Maintaining open and transparent communication channels, especially in distributed teams.
- **Insufficient stakeholder involvement**: Ensuring continuous engagement from customers and stakeholders.
- **Cultural misalignment**: Adapting organizational culture to embrace Agile values like collaboration and empowerment.

Agile Process Model

The Agile Process Model was primarily designed to help a project adapt quickly to change requests. So, the main aim of the Agile model is to facilitate quick project completion. To accomplish this task, it's important that agility is required. Agility is achieved by fitting the process to the project and removing activities that may not be essential for a specific project.

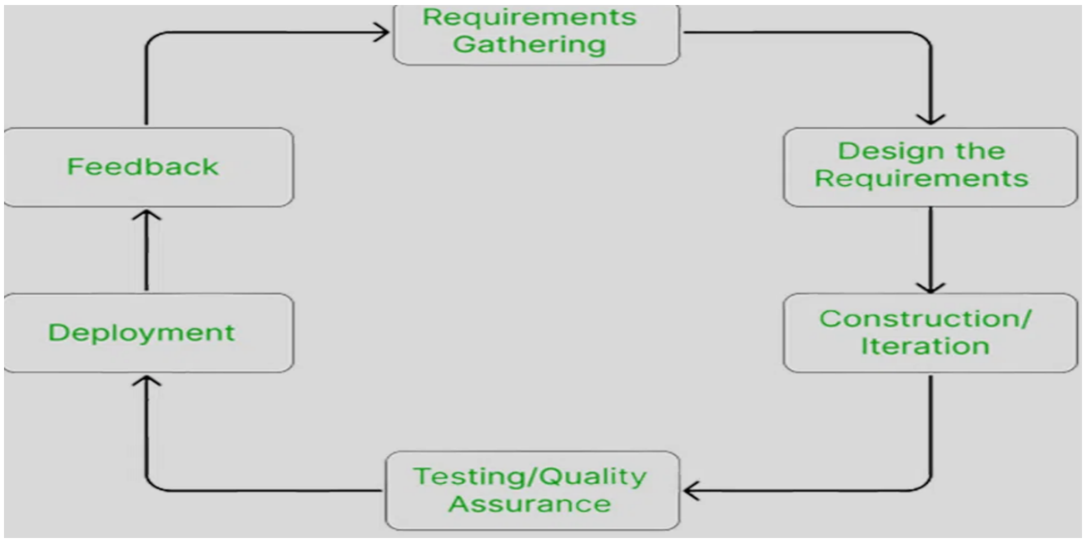
Also, anything that is a waste of time and effort is avoided. The Agile Model refers to a group of development processes. These processes share some basic characteristics but do have certain subtle differences among themselves.

Steps in the Agile Model

The Agile Model is a combination of iterative and incremental process models. The phases involve in **Agile (SDLC) Model** are:

1. Requirement Gathering
2. Design the Requirements
3. Construction / Iteration

4. Testing / Quality Assurance
5. Deployment
6. Feedback



1.Requirement Gathering

In this step, the development team must gather the requirements, by interaction with the customer. **development team** should **plan** the **time and effort needed to build the project**. Based on this information you can evaluate technical and economical feasibility.

- Meet with the customer to really understand their needs and what they expect from the software.
- Identify the key requirements and business goals to make sure everyone is on the same page.
- Estimate how much time and effort it will take to develop the software.
- Assess if the project is technically possible and whether it's worth the investment from both a technical and economic standpoint.

2.Design the Requirements

In this step, the development team will use user-flow-diagram or high-level **UML Diagrams** to show the working of the new features and show how they will apply to the existing software. Wireframing and designing user interfaces are done in this phase.

- **Designing the system**: Once the requirements are gathered, the next step is to design the system's overall architecture based on those needs. This helps to verify the software is structured in a way that meets the user's expectations.
- **Creating wireframes**: Next, wireframes for the user interface (UI) are created. These are simple blueprints that show how the software will look and how users will interact with it, ensuring it's user-friendly and easy to navigate.
- **High-level design with UML diagrams**: At this stage, high-level designs using UML (Unified Modeling Language) diagrams are created to visually represent the software's structure and how different parts will work together.
- **Prototyping for feedback**: Prototypes are made to give stakeholders an early look at the software. This helps gather feedback early in the process and allows for adjustments before the full development begins.

3. Construction / Iteration

In this step, development team members start working on their project, which aims to deploy a working product. Each cycle typically consist between **1-4 weeks**, and at the end, the team delivers a working version of the software.

- **Development of Features**: The team works on the features identified during the requirement and design phases.
- **Coding and Implementation**: New functionalities are coded and integrated into the software based on the goals for that specific iteration.
- **Delivering a Working Product**: After each iteration, a usable version of the software is ready.
- **Incremental Improvement**: With every cycle, the software is enhanced, adding more features and refining existing ones.

4. Testing / Quality Assurance

Testing involves Unit Testing, Integration Testing, and System Testing, Which help in the agile development models:

- **Unit Testing:** Unit testing is the process of checking small pieces of code to ensure that the individual parts of a program work properly on their own. Unit testing is used to test individual blocks (units) of code.
- **Integration Testing:** Integration testing is used to identify and resolve any issues that may arise when different units of the software are combined.

- **System Testing:** Goal is to ensure that the software meets the requirements of the users and that it works correctly in all possible scenarios.

5. Deployment

In this step, the development team will deploy the working project to end users.

Once an iteration is finished and fully tested, the software is ready to be released to the end users. In Agile, deployment isn't a one-time event—it's an ongoing process. Updates and improvements are rolled out regularly, making sure the software keeps evolving and getting better with each release.

- Deploy the software to a test or live environment so that it can be used by customers or end-users.
- Make the software accessible to users, verifying they can start using it as expected.
- Verify the deployment goes smoothly and fix any issues that come up quickly

6. Feedback

This is the last step of the **Agile Model.** In this, the team receives feedback about the product and works on correcting bugs based on feedback provided by the customer.

- **Take feedback** from customers, users, and stakeholders after each iteration.
- Understand how well the product meets user needs and identify areas for improvement.
- **Check for bugs** or issues that need fixing.
- **Make adjustments** to the development plan based on feedback to improve the product further.

Benefits of Agile development methodology

The **Advantages of the Agile Model** are as follows:

- **Flexibility and Adaptability**: Agile can quickly adapt to changes, allowing teams to respond to new customer needs and market conditions.
- **Improved Collaboration**: Agile encourages constant communication between developers and stakeholders, ensuring the product meets user expectations.
- **Faster Delivery**: Agile ensures quicker releases, keeping customers engaged and their feedback incorporated early.
- **Enhanced Quality and Customer Satisfaction**: Agile focuses on customer feedback, ensuring the product meets their needs and delivering high-quality results.
- **Iterative Development**: Work is done in small, manageable steps, allowing for regular improvements and quick adjustments.

- **Transparency**: Agile keeps stakeholders informed at every stage, ensuring clarity and alignment.
- **Quality Assurance**: Agile prioritizes quality, ensuring the product meets users' expectations through continuous improvements.
- **Continuous Improvement**: Regular feedback ensures the product keeps improving, preventing last-minute issues and maintaining high quality.

Extreme Programming (XP)

Extreme Programming (XP) is an Agile software development methodology that focuses on delivering high-quality software through **frequent and continuous feedback, collaboration**, and **adaptation.** XP emphasizes a close working relationship between the **development team, the customer, and stakeholders, with an emphasis on rapid, iterative development and deployment.**

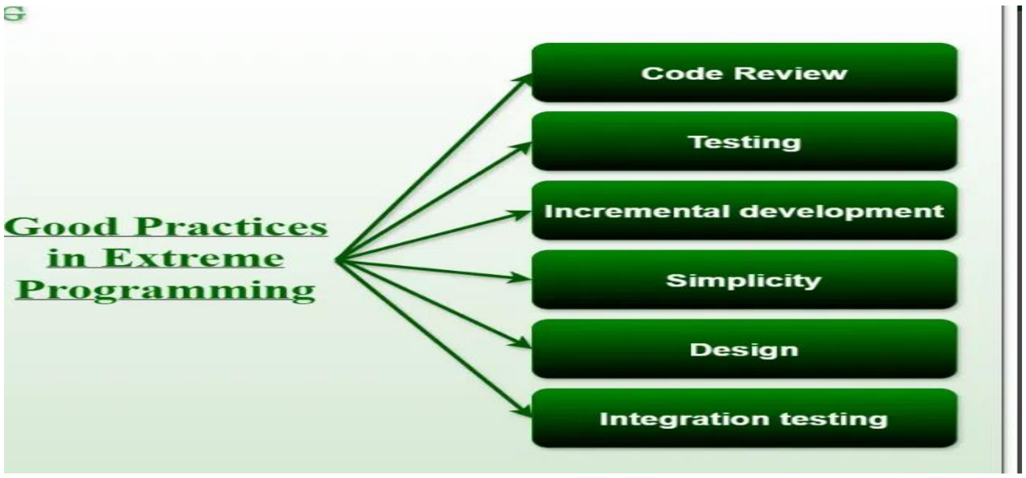
Agile approaches are based on some **common principles**, some of which are:

1. Working software is the key measure of progress in a project.
2. For progress in a project, therefore software should be developed and delivered rapidly in small increments.
3. Even late changes in the requirements should be entertained.
4. Face-to-face communication is preferred over documentation.
5. Continuous feedback and involvement of customers are necessary for developing good-quality software.
6. A simple design that involves and improves with time is a better approach than doing an elaborate design up front for handling all possible scenarios.
7. The delivery dates are decided by empowered teams of talented individuals.

Extreme programming is one of the most popular and well-known approaches in the family of agile methods. an XP project starts with user stories which are short descriptions of what scenarios the customers and users would like the system to support. Each story is written on a separate card, so they can be flexibly grouped.

Good Practices in Extreme Programming

Some of the good practices that have been recognized in the extreme programming model and suggested to maximize their use are given below:



- **Code Review:** Code review detects and corrects errors efficiently. It suggests pair programming as coding and reviewing of written code carried out by a pair of programmers who switch their work between them every hour.
- **Testing:** Testing code helps to remove errors and improves its reliability. XP suggests test-driven development (TDD) to continually write and execute test cases. In the TDD approach, test cases are written even before any code is written.
- **Incremental development:** Incremental development is very good because customer feedback is gained and based on this development team comes up with new increments every few days after each iteration.
- **Simplicity:** Simplicity makes it easier to develop good-quality code as well as to test and debug it.
- **Design:** Good quality design is important to develop good quality software. So, everybody should design daily.
- **Integration testing:** Integration Testing helps to identify bugs at the interfaces of different functionalities. Extreme programming suggests that the developers should achieve continuous integration by building and performing integration testing several times a day.

Basic Principles of Extreme programming

XP is based on the frequent iteration through which the developers implement User Stories. User stories are simple and informal statements of the customer about the functionalities needed. A User Story is a conventional description by the user of a feature of the required system. It does not mention finer details such as the different scenarios that can occur. Based on User stories, the project team proposes Metaphors. Metaphors are a common vision of how the system would work. The development team may decide to build a Spike for some features. A Spike is a very simple program that is constructed to explore the suitability of a solution being proposed. It can be considered similar to a prototype. Some of the **basic activities that are followed during software development by using the XP model** are given below:

- **Coding:** The concept of coding which is used in the XP model is slightly different from traditional coding. Here, the coding activity includes drawing diagrams (modeling) that will be transformed into code, scripting a web-based system, and choosing among several alternative solutions.
- **Testing:** The XP model gives high importance to testing and considers it to be the primary factor in developing fault-free software.
- **Listening:** The developers need to carefully listen to the customers if they have to develop good quality software. Sometimes programmers may not have the depth knowledge of the system to be developed. So, the programmers should understand properly the functionality of the system and they have to listen to the customers.
- **Designing:** Without a proper design, a system implementation becomes too complex, and very difficult to understand the solution, thus making maintenance expensive. A good design results elimination of complex dependencies within a system. So, effective use of suitable design is emphasized.
- **Feedback:** One of the most important aspects of the XP model is to gain feedback to understand the exact customer needs. Frequent contact with the customer makes the development effective.
- **Simplicity:** The main principle of the XP model is to develop a simple system that will work efficiently in the present time, rather than trying to build something that would take time and may never be used. It focuses on some specific features that are immediately needed, rather than engaging time and effort on speculations of future requirements.
- **Pair Programming:** XP encourages pair programming where two developers work together at the same workstation. This approach helps in knowledge sharing, reduces errors, and improves code quality.
- **Continuous Integration:** In XP, developers integrate their code into a shared repository several times a day. This helps to detect and resolve integration issues early on in the development process.
- **Refactoring:** XP encourages refactoring, which is the process of restructuring existing code to make it more efficient and maintainable. Refactoring helps to keep the codebase clean, organized, and easy to understand.

- **Collective Code Ownership:** In XP, there is no individual ownership of code. Instead, the entire team is responsible for the codebase. This approach ensures that all team members have a sense of ownership and responsibility towards the code.
- **Planning Game:** XP follows a planning game, where the customer and the development team collaborate to prioritize and plan development tasks. This approach helps to ensure that the team is working on the most important features and delivers value to the customer.
- **On-site Customer:** XP requires an on-site customer who works closely with the development team throughout the project. This approach helps to ensure that the customer's needs are understood and met, and also facilitates communication and feedback.