LECTURE 1,2

ABOUT Software Engineering and Characteristics of Software Engineering

Software Engineering is a discipline that focuses on the systematic approach to the design, development, maintenance, and evolution of software systems. It involves applying engineering principles and methodologies to software development to ensure software products' reliability, efficiency, and maintainability. Software Engineering is crucial because it helps manage the complexity of software systems, ensures quality and reliability, meets user requirements, and delivers software projects within budget and on schedule. It provides structured approaches to software development that improve productivity and reduce risks associated with software failures.

Software engineering is a complex field that requires a deep understanding of various principles and characteristics.

These **characteristics are the fundamental attributes** that define the nature and functionality of software systems.

.

1. **Maintainability**

Maintainability is a critical characteristic of software engineering, as it accounts for about [75% of project costs](https://www.devstringx.com/software-maintainability).

It refers to the ease with which a software system can be modified

High maintainability is desirable as it reduces the time and effort required for modifications and updates. Several factors contribute to a software system’s maintainability, including , readability, and simplicity of design.

How easily and cost-effectively software can be modified, updated, or extended.

**2.Portability:**The ability of software to run on different platforms or environments without requiring significant modifications.

3. **Dependability**: Dependability is one of the crucial software engineering characteristics. It refers to the reliability, safety, and security of a software system.

A dependable software system is one that performs its intended functions correctly and consistently without failure.

4**. Efficiency**

Efficiency in software engineering refers to a software system’s ability to perform its functions quickly without wasting resources.

An efficient software system makes optimal use of system resources, such as memory and processor time, to deliver high performance.

[Software engineers](https://www.institutedata.com/blog/hairdressing-to-high-tech-avril-malehams-empowering-journey-into-software-engineering/) strive to create efficient software by optimising algorithms, reducing memory usage, and minimising the load on system resources.

The characteristics of software engineering are not just theoretical concepts; they have practical implications in the real world.

They guide the design and development of software systems, influencing their quality, performance, and reliability.

Understanding these characteristics is essential for software engineers, as it helps them create software that meets users’ needs and performs well in a variety of environments.

By focusing on these characteristics, software engineers can ensure that their software is maintainable, dependable, and efficient, thereby delivering value to users and businesses alike.

**Life cycle of a software system**

software creation is not an easy task. Hence, you would need to design a Software Development Life Cycle (SDLC) to circumvent obstacles in the implementation process.

Let’s explore the different phases to [start a successful software development project](https://blog.kms-solutions.asia/how-to-start-a-software-project).

Here’re ***key reasons*** why SDLC is essential for creating a software system:

1. It provides a foundation for project planning, scheduling and estimating.
2. Increases the visibility of project planning to all relevant development process stakeholders.
3. Helps accelerate and improve development
4. It is a system for managing and tracking the progress of projects.
5. Helps achieve the highest level of transparency and management control.
6. Decrease project risk and improve development speed.

An SDLC describes each stage of the software development cycle, breaking down the process into a variety of separate steps: goals, expectations, tasks, process instructions, documentation, deliverables, and go-to personnel.

The precise number and characteristics of steps depend on the company and its product objectives. Each phase in an SDLC results in an output that serves as the prerequisite for the following step.

In most cases, the software development life cycle consists of 7 phases.

 most cases, the software development life cycle consists of 7 phases.



1. Requirements Gathering & Analysis

**Purpose: This is where teams examine the requirement analysis with inputs from clients, the sales department, market research reports and subject matter experts. For instance, a banking mobile app would require [eKYC](https://blog.kms-solutions.asia/ekyc-in-digital-banking) for the digital onboarding process and**[**digital lending platforms**](https://blog.kms-solutions.asia/most-popular-digital-lending-business-model)**covering the entire loan lifecycle.**

**Some critical questions during this phase that you can consider include:**

**What is the objective?**

**What are the main results that companies hope to get from the product?**

**Is it developing a new software project or upgrading an existing one?**

**Do we have any deadlines that need to pay attention to?**

**Do we use the internal development team, or will we have to outsource some part(s) of the project?**

**The person mainly in charge: Business analysts (BA).**

**The requirements collected by BA will need to get approval from either the customer or the market analysts via a Software Requirement Specification (SRS), which contains both functional and non-functional requirements to be developed throughout the project lifecycle.**

**Output: The SRS documents that defined the project's specific objectives and needs will then be used as a point of reference for the next stage.**

**The SRS will need to include all the software, hardware, security, and specifications needed.**

**2. Planning**

**Purpose:**Once gathering requirements is completed, the next step is to plan. From there, the terms of the project will be determined:

* Cost estimation
* Required resources
* Tech stack
* Roles needed in the project
* Project scheduling

Leadership structure

**The person mainly in charge:**Team Leaders and Higher Management

Project managers will need to clearly outline the project’s scope and objective. This charts the team's trajectory to efficiently develop the software. Additionally, it establishes constraints to prevent the project from escalating or deviating from its initial objective.

**Output:**An expanded SRS document with product feasibility analysis and risk identification.

Based on this outcome, you can define the appropriate technical methods that should be implemented for the project to run smoothly with minimum risks.

3. Design

**Purpose:** The third phase goes into great detail about the required standards, features, and activities that fulfil the functional specifications of the proposed system.

Some essential factors that you need to take into account:

* Architecture: quality attributes, IT environment, human dynamics, and business strategy.

User Interface: how users interact with the software.

* Platforms - decide on which platform the software will launch (Windows, Android, iOS, and Macintosh).
* Programming involves techniques for resolving issues and executing tasks in the app alongside the programming language.
* Communications: specifies the channels through which the software can talk to other assets, e.g. a central server.
* Security: describes the procedures taken to safeguard the program, which might consist of password protection, SSL traffic encryption, and secure user credential storage.

During this SDLC step, a prototype is worth considering. Although it is time-consuming, prototyping will be much more affordable than making significant modifications after the software development phase.

**The person mainly in charge:**Architects and Senior Developers

A person who is responsible for this will gather this info in the Design Document Specification (DDS). Stakeholders then will examine the DDS based on the software’s risk assessment, design modularity and time constraints.

**Output:**A detailed DDS that lists all the information required to code the product.

4. Software Development

**Purpose:** This phase of SDLC involves writing code and transforming design documentation into actual software. Tools like Access Control or Source Code Management might be used to keep track to code changes.

**The person mainly in charge:**Developers

The developers’ team should make sure that their code adheres to the planned blueprint to meet the software requirement specifications. It’s also important that you have clear instructions about the code practices and style.

**Output:**Source code of a testable, fully functional software**.**

5. Testing

**Purpose:** This stage of SDLC helps ensure the software has no bugs or exploits and can run smoothly.

Businesses have various [testing strategies](https://blog.kms-solutions.asia/testing-strategies-for-mobile-app-development) to evaluate the new product, including

[Performance testing](https://kms-solutions.asia/performance-testing-services)

[Security testing](https://kms-solutions.asia/security-testing-services)

Unit testing (functional tests)

[Automation testing](https://kms-solutions.asia/automation-testing-for-bfsi)

[Integration testing.](https://kms-solutions.asia/api-testing-services)

When the product is stable, bug-free, and up to the quality criteria outlined in the earlier stages, the testing step is complete.

**The person mainly in charge:**Quality Assurance (QA) specialists.

QA engineers usually deploy a range of frameworks in addition to continuous integration to perform unit tests and automation compilation. Testers will need to have a combination of manual and[automated testing knowledge and skills](https://blog.kms-solutions.asia/what-are-essential-skills-for-a-top-automation-tester) to perform the tests effectively,

**Output:**A thoroughly tested version of the product ready for a production environment.

6. Deployment

**Purpose:** Depending on the client’s requirements, the product might undergo User Acceptance Testing (UAT) where it is tested in a real business environment with a limited segment. Most businesses deploy their new products to a small percentage of users (10-15%) and slowly release them to the rest of the customer base.

**The person mainly in charge:**Project Managers

**Output:** The release of a fully functional and tested product.

7. Maintenance

**Purpose:**The final SDLC phase entails maintenance and regular upgrades. This stage concentrates on keeping the software safe, increasing performance, and occasionally adding new features.

**The person mainly in charge:**Production support engineers, testers, and developers.

**Output:** A fully monitored software that’s constantly requiring improvements

What is the most important phase of SDLC?

Requirements gathering and analysis is the most crucial stage of the SDLC cycle. Without understanding the requirements, no project team can create a solution that is appreciated by customers. Most teams put effort into developing new solutions and neglecting requirement gathering which results in the failure of the software projects. This is one of the main reasons driving 66% of software projects to fail, as per the Standish Group’s 2022 CHAOS report. Hence, it’s crucial that each requirement is outlined in detail and understood by the development team.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** **Common methodologies include:**

* + **Waterfall: Sequential approach with distinct phases and rigid structure.**
  + **Agile: Iterative and incremental approach focusing on flexibility, collaboration, and continuous improvement.**
  + **Scrum: Agile framework emphasizing teamwork, accountability, and iterative progress through sprints.**
  + **DevOps: Integration of development and operations to streamline software delivery and improve collaboration.**

**lecture 3.Software Process and** **Software Development Lifecycle Model**

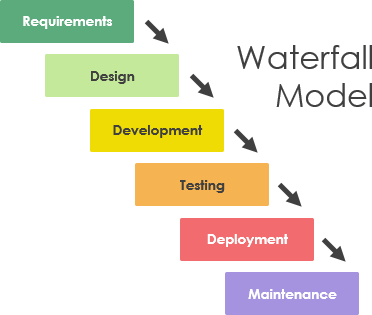
# One of the basic notions of the software development process is SDLC models which stands for Software Development Life Cycle models. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out. The most used, popular and important SDLC models are given below:

# Waterfall model V model Incremental model RAD model Agile model

# Iterative model Spiral model Prototype model .

**Waterfall Model**

The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialisation of tasks. The approach is typical for certain areas of engineering design.



**Key Phases of the Waterfall Model:**

1. **Requirements Gathering and Analysis:**

All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

1. **System Design:**

Based on the requirements, the system's architecture and design are created. This phase involves creating design documents that specify hardware and system requirements.

1. **Development(Coding):**

The actual source code is written based on the design specifications. This phase is typically divided into units or modules.

1. **Integration and Testing:**

Once the code is written, the different modules are integrated and tested as a complete system to check for any faults and bugs.

1. **Deployment of the System:**

The system is deployed in the production environment where it will be used by the end-users.

1. **Maintenance:**

The system is maintained and updated as necessary to handle any post-deployment issues or new requirements that emerge over time. With an application, I will explain this topic

**Example: Developing a Library Management System**

1. **Requirements Gathering and Analysis:**

Identify all the features needed in the library management system. For example:

* + - User management (librarian and members)
    - Book management (adding, removing, and updating book records)
    - Issuing and returning books
    - Search functionality for books
    - Fine calculation for overdue books

These requirements are documented in detail.

1. **System Design:**

Design the overall system architecture.

Decide on the database schema (e.g., tables for users, books, transactions).

Create design documents specifying how different components will interact.

Design the user interface layout and interactions.

1. **(Coding):**

Developers write the code for each module based on the design documents.

Example modules might include:

* + - User authentication and authorization
    - Book catalog management
    - Transaction handling (issuing and returning books)
    - Search functionality
    - Fine calculation logic

1. **Integration and Testing:**

Integrate the modules to create a complete system.

Conduct various types of testing:

* + - Unit testing to check individual modules
    - Integration testing to ensure modules work together
    - System testing to validate the entire system against the requirements
    - User acceptance testing to ensure the system meets users' needs

1. **Deployment of the System:**

Deploy the completed library management system on the library's server.

Train librarians and staff to use the system.

1. **Maintenance:**

Monitor the system for any issues that arise post-deployment.

Provide updates and patches as needed.

Implement new features or modifications based on user feedback or changing requirements.

**Advantages of the Waterfall Model:**

* **Simplicity and Ease of Use:** Easy to understand and manage due to its linear nature.
* **Structured Approach:** Clear documentation and distinct stages ensure thoroughness.
* **Early Detection of Defects:** Defects can be identified early in the lifecycle during the requirements and design phases.

**Disadvantages of the Waterfall Model:**

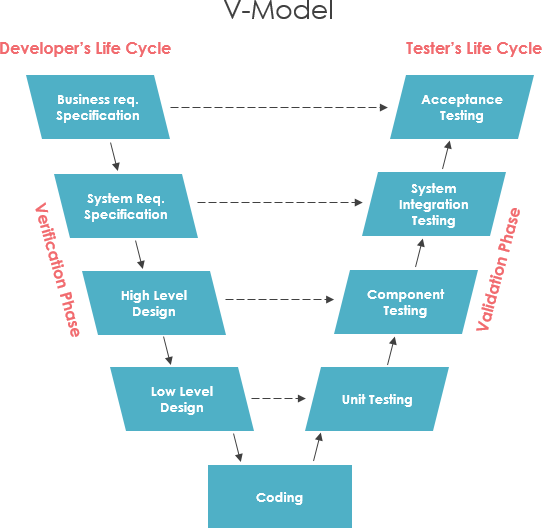
**Inflexibility:** Once a phase is completed, going back to make changes is difficult and costly.

**Poor Handling of Changes:** It is not well-suited for projects where requirements are likely to evolve over time.

**Late Testing:** Testing is only done after development is complete, making it harder to identify and fix issues early.

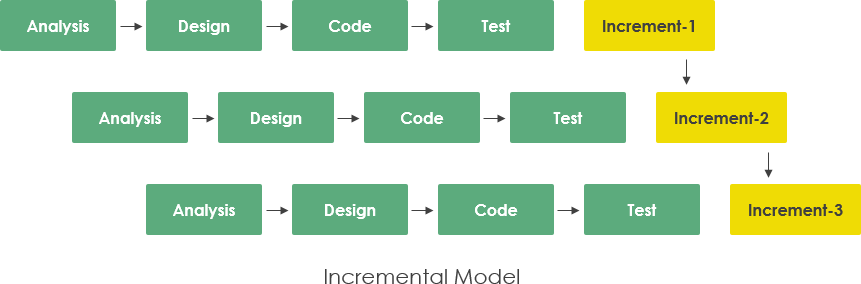
**V Model**

**The V-model represents a development process that may be considered an extension of the waterfall model and is an example of the more general V-model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.**



**Incremental model**

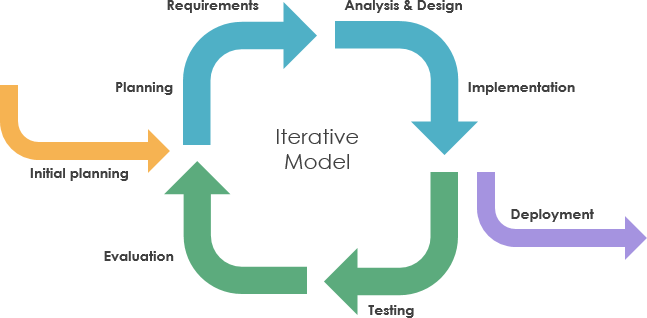
**The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements. Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionally has been implemented. This model combines the elements of the waterfall model with the iterative philosophy of prototyping.**



**Iterative Model**

An iterative life cycle model does not attempt to start with a full specification of requirements by first focusing on an initial, simplified set user features, which then progressively gains more complexity and a broader set of features until the targeted system is complete. When adopting the iterative approach, the philosophy of incremental development will also often be used liberally and interchangeably.

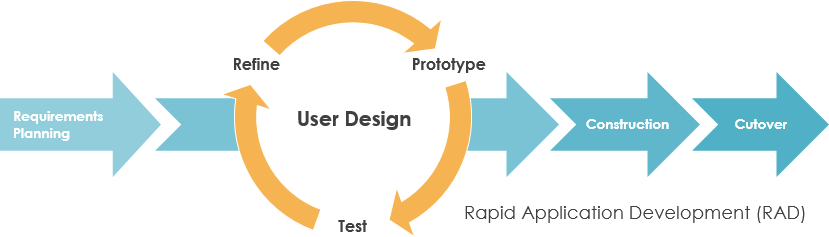
In other words, the iterative approach begins by specifying and implementing just part of the software, which can then be reviewed and prioritized in order to identify further requirements. This iterative process is then repeated by delivering a new version of the software for each iteration. In a light-weight iterative project the code may represent the major source of documentation of the system; however, in a critical iterative project a formal software specification may also be required.



**RAD model**

Rapid application development was a response to plan-driven waterfall processes, developed in the 1970s and 1980s, such as the Structured Systems Analysis and Design Method (SSADM). Rapid application development (RAD) is often referred as the adaptive software development. RAD is an incremental prototyping approach to software development that end users can produce better feedback when examining a live system, as opposed to working strictly with documentation. It puts less emphasis on planning and more emphasis on an adaptive process.

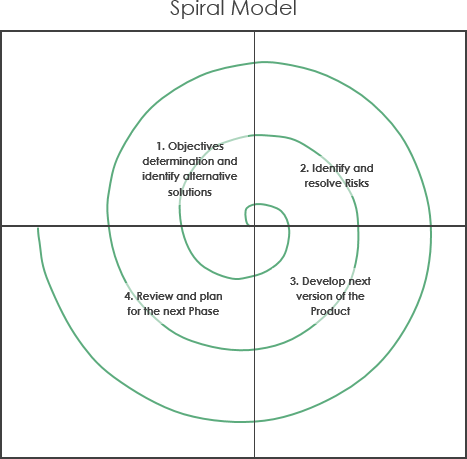
RAD may resulted in a lower level of rejection when the application is placed into production, but this success most often comes at the expense of a dramatic overruns in project costs and schedule. RAD approach is especially well suited for developing software that is driven by user interface requirements. Thus, some GUI builders are often called rapid application development tools.



**Spiral model**

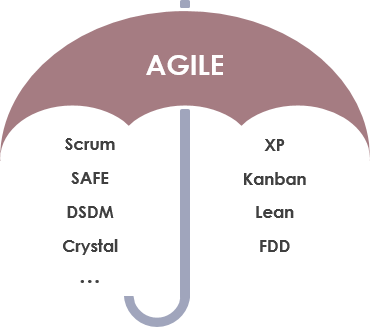
The spiral model, first described by Barry Boehm in 1986, is a risk-driven software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model. A spiral model looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. This model supports risk handling, and the project is delivered in loops. Each loop of the spiral is called a Phase of the software development process.

The initial phase of the spiral model in the early stages of Waterfall Life Cycle that is needed to develop a software product. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using a spiral model.



**Agile model**

Agile is an umbrella term for a set of methods and practices based on the values and principles expressed in the Agile Manifesto that is a way of thinking that enables teams and businesses to innovate, quickly respond to changing demand, while mitigating risk. Organizations can be agile using many of the available frameworks available such as Scrum, Kanban, Lean, Extreme Programming (XP) and etc.



The Agile model in software engineering is an iterative and incremental approach to project management and software development. It emphasizes flexibility, collaboration, customer feedback, and rapid delivery of functional software. Unlike traditional models like the Waterfall model, where the project progresses through a linear sequence of phases, the Agile model promotes continuous iteration of development and testing throughout the project lifecycle.

**Key Principles of Agile:**

1. **Customer Collaboration over Contract Negotiation:** Engage customers continuously to gather feedback and ensure the product meets their needs.
2. **Individuals and Interactions over Processes and Tools:** Focus on communication and teamwork.
3. **Working Software over Comprehensive Documentation:** Prioritize delivering functional software.
4. **Responding to Change over Following a Plan:** Be adaptable to changes even in late development stages.

**Agile Methodologies:**

Several methodologies are based on Agile principles, including Scrum, Kanban, Lean, Extreme Programming (XP), and others.

**Example of Agile in Practice (Using Scrum):**

1. **Product Backlog:** A prioritized list of features, enhancements, and bug fixes required for the product.
2. **Sprint Planning:** The team selects a set of items (user stories) from the product backlog to work on during the next sprint (a time-boxed period, usually 2-4 weeks).
3. **Sprint:** The team works on the selected items. Daily stand-up meetings are held to discuss progress, impediments, and plans for the day.
4. **Increment:** At the end of the sprint, a potentially shippable product increment is delivered.
5. **Sprint Review:** The team demonstrates the work done to stakeholders and gathers feedback.
6. **Sprint Retrospective:** The team reflects on the sprint to identify and implement improvements in the next sprint.

**Example Scenario:**

Imagine a team developing a new e-commerce website.

1. **Product Backlog:** The product owner creates a backlog with items like "User can create an account," "User can browse products," "User can add products to the cart," and "User can checkout."
2. **Sprint Planning:** The team decides to focus on "User can create an account" and "User can browse products" for the first sprint.
3. **Sprint:** Developers, testers, and designers work together on these features. Daily stand-ups ensure everyone is aligned and any blockers are addressed.
4. **Increment:** By the end of the sprint, the account creation and product browsing features are complete and functional.
5. **Sprint Review:** The team presents these features to the product owner and stakeholders, who provide feedback.
6. **Sprint Retrospective:** The team discusses what went well, what didn't, and how they can improve in the next sprint.

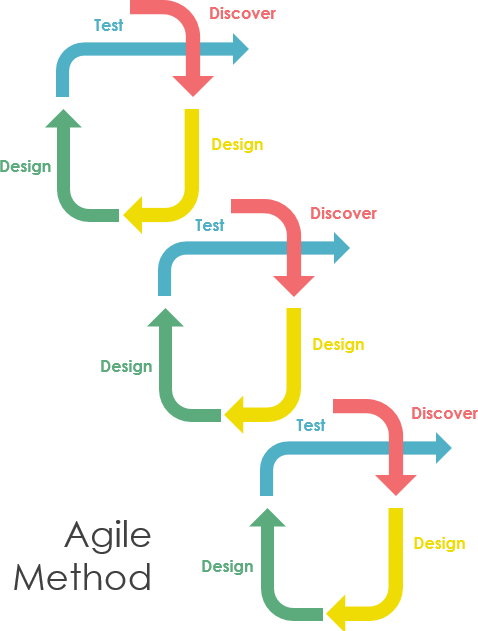
**Benefits of Agile:**

* **Flexibility:** Easy to adapt to changes in requirements.
* **Customer Satisfaction:** Continuous delivery of valuable software ensures the final product meets customer expectations.
* **Improved Quality:** Frequent testing and feedback loops enhance software quality.
* **Transparency:** Regular meetings and updates keep everyone informed and involved.

**Challenges of Agile:**

* **Requires Experience:** Teams need to be well-versed in Agile practices for it to be effective.
* **Scope Creep:** Continuous changes can lead to an ever-growing project scope.
* **Coordination:** Requires strong communication and coordination, especially in larger teams or distributed environments.

Overall, Agile is a highly effective model for managing software projects in dynamic and fast-paced environments, where requirements frequently change, and customer involvement is crucial.



terative and Agile are both approaches used in software development to manage and improve the development process. While they share similarities, such as breaking down the project into smaller segments and delivering software in increments, there are key differences between them in terms of principles, flexibility, and methodologies.

**Iterative Model**

**Definition:** The iterative model is a process where software development is carried out in repeated cycles (iterations). Each iteration aims to improve and expand the software based on feedback and new requirements.

**Key Characteristics:**

1. **Sequential Cycles:** Development is divided into a series of cycles, each building on the previous one.
2. **Feedback-Driven:** Each iteration is reviewed, and feedback is used to refine the product.
3. **Less Flexibility:** Changes can be incorporated in subsequent iterations, but the overall process is more rigid compared to Agile.

**Example:** Consider developing a word processing application.

* **Iteration 1:** Basic text editor functionality (open, edit, save text files).
* **Iteration 2:** Add formatting features (bold, italic, underline).
* **Iteration 3:** Incorporate advanced features (spell check, grammar check).

At the end of each iteration, the software is reviewed, and feedback is incorporated into the next iteration. However, significant changes might be deferred to later iterations rather than being immediately addressed.

**Agile Model**

**Definition:** The Agile model is a flexible, iterative approach emphasizing customer collaboration, continuous delivery, and the ability to respond to change quickly.

**Key Characteristics:**

1. **Short Iterations (Sprints):** Work is divided into small, manageable units (sprints), typically lasting 2-4 weeks.
2. **Customer Involvement:** Regular feedback from customers and stakeholders is integral.
3. **High Flexibility:** Changes can be made at any point based on customer feedback and evolving requirements.
4. **Agile Methodologies:** Includes Scrum, Kanban, Extreme Programming (XP), etc.

**Example Using Scrum (E-Commerce Website):**

* **Sprint 1:** Develop user registration and login functionality.
* **Sprint Review:** Present to stakeholders, receive feedback.
* **Sprint 2:** Based on feedback, improve the registration process and add product browsing.
* **Sprint Review:** Present new features, gather further feedback.
* **Sprint 3:** Address feedback, add shopping cart functionality, and continue iterating.

In Agile, each sprint ends with a potentially shippable product increment, and teams regularly reassess priorities and make adjustments as needed.

**Key Differences**

| **Aspect** | **Iterative Model** | **Agile Model** |
| --- | --- | --- |
| **Approach** | Sequential iterations with feedback for each cycle | Iterative and incremental with continuous feedback loops |
| **Flexibility** | Less flexible, changes incorporated in future iterations | Highly flexible, changes can be made anytime |
| **Customer Involvement** | Limited to feedback at the end of iterations | High, continuous collaboration throughout the project |
| **Feedback** | At the end of each iteration | Continuous, throughout each sprint |
| **Cycle Length** | Longer iterations | Short, time-boxed sprints (2-4 weeks) |
| **Focus** | Improving product incrementally with each iteration | Rapid delivery, customer satisfaction, and adaptability |

**Practical Examples:**

**Iterative Model Example:** Developing a new mobile operating system:

1. **Iteration 1:** Implement basic OS functionalities (booting, basic UI).
2. **Iteration 2:** Add essential apps (phone, messaging).
3. **Iteration 3:** Introduce advanced features (multi-tasking, app store).

**Agile Model Example:** Creating a new project management tool:

1. **Sprint 1:** Basic task creation and management.
2. **Sprint Review:** Gather feedback from users.
3. **Sprint 2:** Based on feedback, enhance task features and add user collaboration.
4. **Sprint Review:** Present to users, gather further feedback.
5. **Sprint 3:** Continuously improve and add features based on ongoing user feedback.

In summary, while both Iterative and Agile models aim to improve software development processes, Agile offers a more flexible, customer-centric approach with frequent iterations and feedback loops, making it well-suited for dynamic environments where requirements frequently change.