Software Development Life Cycle (SDLC) Models

They define a systematic approach to break the software development process into manageable phases, ensuring quality, efficiency, and alignment with user needs.

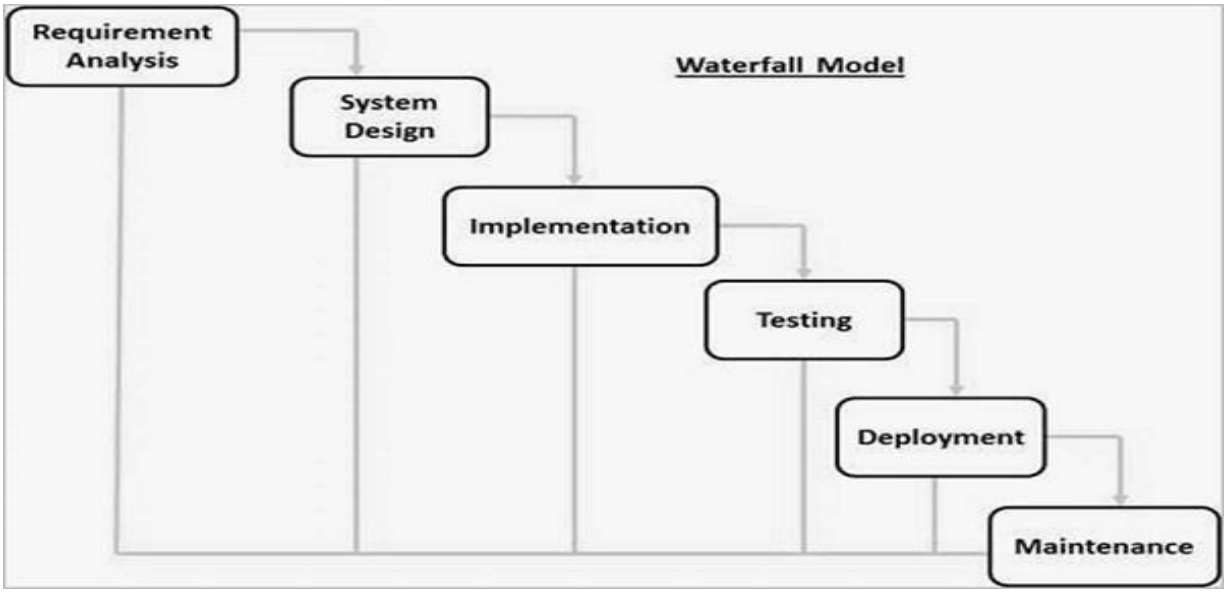
Each SDLC model describes the order of activities and focuses on key objectives such as risk management, user involvement, or rapid delivery. The choice of model depends on project requirements, complexity, and stakeholder expectations.

---

1. Waterfall Model

Overview:

The Waterfall Model is a sequential design process in which progress flows steadily through phases like Requirement Analysis, Design, Implementation, Testing, Deployment, and Maintenance.



Phases of the Waterfall Model

1. **Requirement Analysis**:
    - Gather all project requirements from stakeholders.
    - Document the requirements for future reference.
    - Outcome: A detailed **Requirements Specification Document**.
2. **System Design**:
    - Create system architecture based on the requirements.
    - Break down the system into modules and define data flows.
    - Outcome: **Design Documents** like DFDs, ERDs, or UML diagrams.
3. **Implementation (Coding)**:
    - Developers write the code based on the design.
    - Focus on building each module independently and integrating them.
    - Outcome: **Executable Code**.
4. **Testing**:
    - Test the system for bugs and ensure it meets the requirements.
    - Includes unit testing, integration testing, system testing, and acceptance testing.
    - Outcome: A **stable, bug-free product**.
5. **Deployment**:
    - Deliver the product to the client for real-world use.
    - May involve installation and setup in the client's environment.
    - Outcome: A **deployed software system**.
6. **Maintenance**:
    - Perform updates, bug fixes, and improvements based on user feedback.
    - Ensure the system remains operational and relevant over time.
    - Outcome: **Long-term system upkeep**.

Advantages:

1. **Simple and Easy to Understand**: Each phase is well-defined and does not overlap.
2. **Ideal for Small Projects**: Works well for projects with clear and stable requirements.

Disadvantages:

1. **Rigid and Inflexible**: Difficult to accommodate changes once a phase is completed.
2. **Late Testing**: Errors are discovered late in the development cycle, potentially increasing costs.

When to Use:

- Projects with **well-defined, stable, and unchanging requirements**.
- Projects where technology is **well-understood**.

---

2. Prototype Model

Overview:

The Prototype Model involves building a working prototype of the system to understand requirements better before full-scale development.

The **Prototype Model** is a software development approach that focuses on creating a **working prototype** early in the development process to better understand and refine the system's requirements. This prototype is an initial version or representation of the final product that demonstrates its core functionalities.

The key idea is **iterative refinement**—feedback from users or stakeholders is gathered, and the prototype is updated until the final requirements are understood and approved.

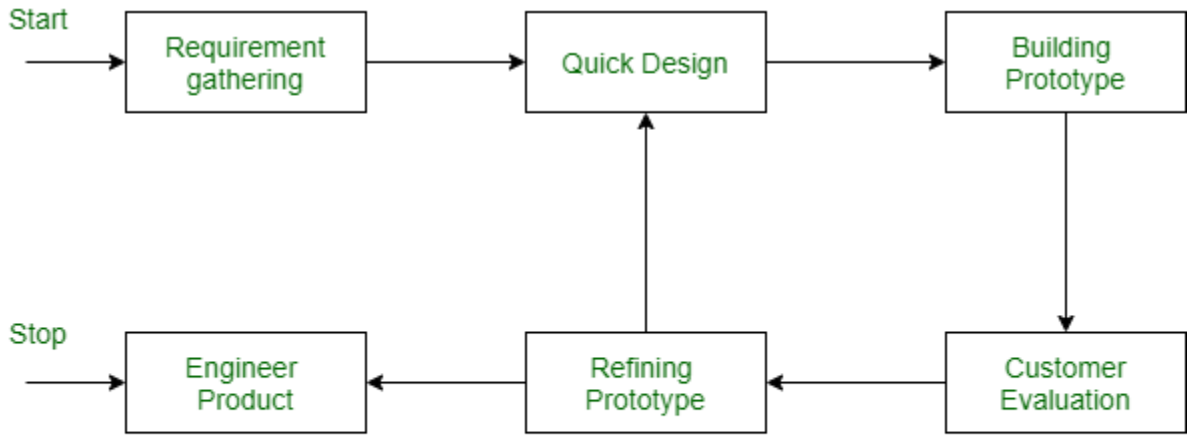


Figure - Prototype Model

Phases of the Prototype Model

1. **Requirement Gathering**:

    - Initial requirements are collected, focusing on features that are unclear or need user input.
    - High-level functional requirements are identified.
2. **Quick Design**:
    - A basic design of the system is created, focusing on the critical areas.
    - Includes a user interface mock-up or a simple model to demonstrate the main functionalities.
3. **Prototype Development**:
    - A working prototype is built based on the quick design.
    - The prototype includes key features but is not a complete or final version of the product.
4. **User Evaluation**:
    - Stakeholders or end-users evaluate the prototype.
    - Feedback on the functionality, interface, and usability is collected.
5. **Prototype Refinement**:
    - Based on the feedback, the prototype is refined and re-evaluated in subsequent iterations.
    - This process continues until the requirements are clearly defined and approved.
6. **Final Development**:
    - Once the prototype is accepted, the actual development begins using the final, refined requirements.
    - The prototype may or may not be discarded.

Advantages:

1. **Better Requirement Understanding**: Helps in clarifying unclear or evolving requirements.
2. **Enhanced User Feedback**: Early user involvement leads to higher satisfaction.

Disadvantages:

1. **Time-Consuming**: Iterative prototyping may delay actual development.
2. **Increased Costs**: Developing multiple prototypes can be expensive.

When to Use:

- Projects with **unclear or evolving requirements**.
- Systems where **user interaction is critical** (e.g., UI-heavy applications).

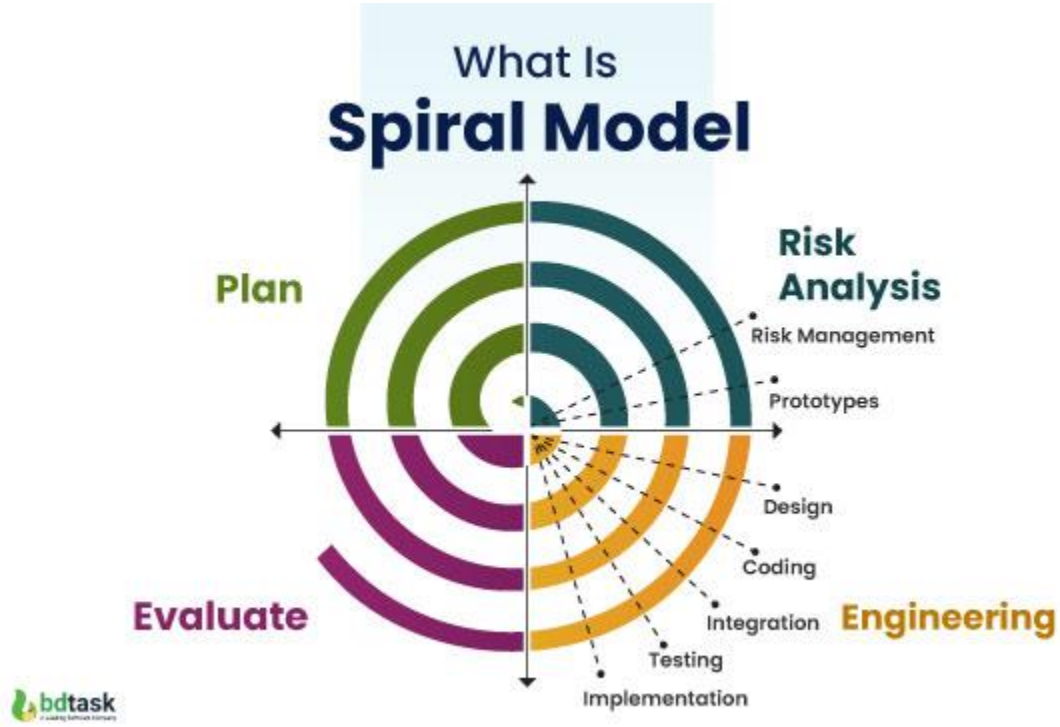
---

3. Spiral Model

Overview:

The **Spiral Model** is a risk-driven software development model that combines elements of the Waterfall Model and iterative development. It emphasizes continuous refinement of the product through repeated cycles (or iterations) called "spirals." Each spiral consists of four phases: planning, risk analysis, engineering, and evaluation.

It is particularly suited for large, complex, or high-risk projects, where understanding and addressing risks early is critical.



Phases of the Spiral Model

Each spiral (iteration) involves the following key phases:

1. **Planning Phase**:
    - Identify objectives for the current iteration.
    - Gather initial requirements or refine existing ones.

    - Plan activities and resources for this spiral.
    - Outcome: A detailed plan for the iteration.
2. **Risk Analysis Phase**:
    - Identify risks associated with the project or iteration (e.g., technical, cost, schedule risks).
    - Evaluate and analyze risks to prioritize mitigation strategies.
    - Develop prototypes or models to mitigate identified risks.
    - Outcome: Risk assessment document or prototype.
3. **Engineering Phase**:
    - Develop and test the product for this iteration.
    - Includes design, coding, and testing activities.
    - Outcome: A working version of the software.
4. **Evaluation Phase**:
    - Deliver the product increment to stakeholders or end-users for evaluation.
    - Gather feedback to refine requirements for the next iteration.
    - Outcome: Stakeholder feedback and updated requirements.

Advantages:

1. **Risk Reduction**: Early identification and mitigation of risks.
2. **Flexibility**: Accommodates changes and evolving requirements easily.

Disadvantages:

1. **Expensive**: Complex and costly due to risk management and iterative cycles.
2. **Requires Expertise**: Needs skilled professionals to manage risks and iterations.

When to Use:

- Large and complex projects with **significant risks**.
- Projects where **requirements are expected to evolve** over time.

---

4. Agile Model

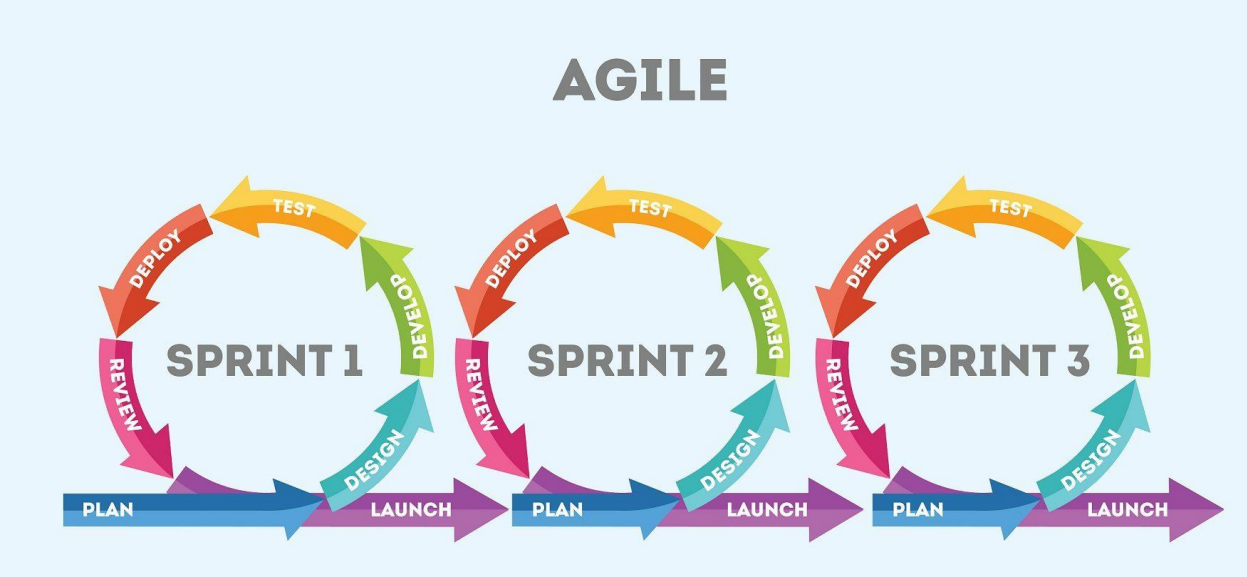
Overview:

The Agile Model emphasizes iterative development, collaboration, and flexibility in accommodating changes.

The agile model refers to the iterative approach to delivering a software product. This means that instead of delivering a large project only when all parts are complete, a team breaks down a large project into smaller parts, and delivers these completed smaller parts in regular cycles.

For example, using the agile model a developer delivers multiple releases of a product and each release could add a feature, such as drop-down menus, checkboxes, or multiple-choice buttons. The final release would contain all features.

The **Agile Model** is a flexible and iterative approach to software development, emphasizing **collaboration**, **customer feedback**, and **adaptive planning**. It focuses on delivering small, functional parts of the system (called increments) in short development cycles known as **iterations** or **sprints**. Agile allows teams to quickly respond to changing requirements and priorities.



Phases of the Agile Model

1. **Concept and Inception**:

    - Define the high-level scope and goals of the project.
    - Create a backlog of features (user stories) that need to be developed.
2. **Iteration Planning**:
    - Select a subset of features or user stories to implement in the upcoming iteration.
    - Prioritize features based on business value and user needs.
3. **Design and Development**:
    - Design and code the features selected for the sprint.
    - Developers work collaboratively, often using techniques like pair programming.
4. **Testing and Integration**:
    - Continuous testing ensures the software meets quality standards.
    - Features are integrated with the existing system after testing.
5. **Review and Feedback**:
    - Deliver the working software increment to stakeholders.
    - Gather feedback and update the product backlog for future iterations.
6. **Release**:
    - Deliver the final, fully functional product to the customer after multiple iterations.

Advantages:

1. **Highly Flexible**: Easily adapts to changing requirements.
2. **User-Centric**: Continuous delivery ensures user satisfaction.

Disadvantages:

1. **Less Predictable**: Difficult to estimate costs and timelines accurately.
2. **Documentation Issues**: Focus on working software may lead to less comprehensive documentation.

When to Use:

- Projects with **rapidly changing requirements**.
- Systems needing **frequent updates** or **early market release**.