SYSTEM DEVELOPMENT

A system, is a collection of elements and procedures that work together to accomplish a specific task.
All elements in a system are organized in a specific manner to achieve the system goal.

A real-world example of a computer system is a smartphone. It consists of hardware components like a processor, memory, display, and camera, as well as software, including an operating system (e.g., iOS or Android) and various applications.

Characteristics of a System

- *Interconnected Elements*: Systems consist of components or elements that are connected or interact with each other.
- *Common Purpose*: They have a shared goal or purpose that the components work together to achieve.
- *Boundaries*: Systems have defined boundaries that separate them from their environment.
- *Input and Output:* They receive input, process it, and produce output.
- *Organization:* Systems are organized and structured to fulfill their purpose efficiently.
- *Feedback:* They often incorporate feedback mechanisms to adapt and improve their performance.
- *Emergent Properties*: Systems can exhibit properties or behaviors that arise from the interaction of their components.
- *Environment Interaction*: Systems interact with their surrounding environment, influencing and being influenced by it.
- *Control:* They may have control mechanisms to manage and regulate their internal processes.
- *Hierarchy:* Systems can be part of larger systems and may consist of subsystems themselves.

Basic elements of a system

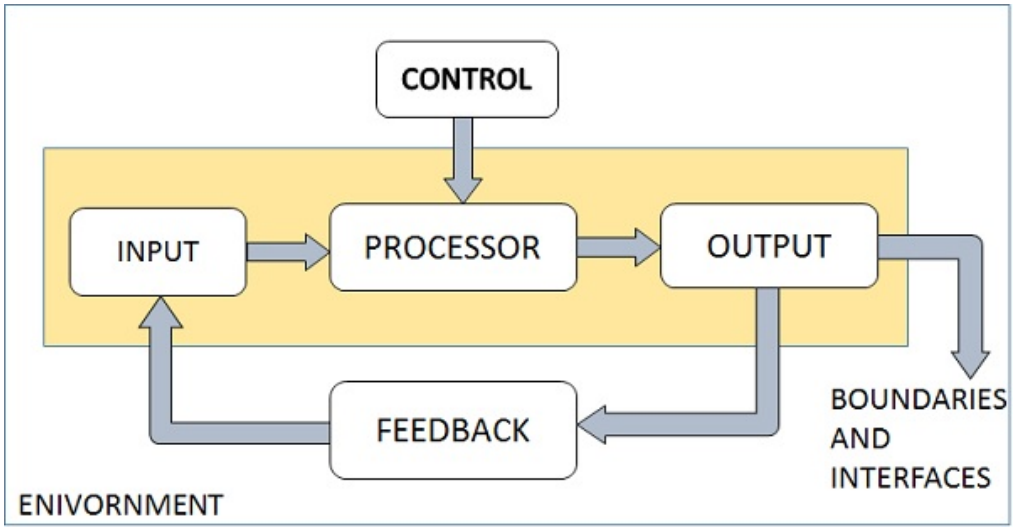
- Input: This is where the system receives information, data, or resources from its environment. It's the starting point for the system to operate.

- Processing: The processing element represents the core of the system, where the input is manipulated or transformed in some way to produce an output. This can involve computations, decisions, or actions.

- Output: The output is the result or outcome of the system's processing. It represents what the system produces or provides back to its environment.

- Feedback: Feedback is a mechanism that allows the system to monitor and adjust its performance. It provides information to the system about the impact of its output on the environment, enabling it to make necessary corrections or improvements.

- Control: Control is responsible for regulating the system's behavior. It ensures that the input, processing, and output components work in coordination to achieve the system's goals.



Classification of Systems

Open and Closed Systems:

Open Systems: Interact with their environment, exchanging matter and energy with it. For example, a living organism that takes in food and expels waste.

Closed Systems: Isolated from their environment and do not exchange matter with it. For example, a sealed thermos that keeps liquids hot or cold without letting heat escape.

Physical and Abstract Systems:

Differences between closed and open systems

Open System:

- Interacts with its environment.
- Exchanges matter and energy with the external environment.
- Allows inputs and outputs to flow between the system and its surroundings.
- Often found in living organisms, ecosystems, and many human-made systems.
- Can adapt and evolve based on interactions with the environment.

Closed System:

- Isolated from its environment.
- Does not exchange matter with the external environment, although it may exchange energy.
- Typically sealed or insulated from the surroundings.
- Common in physical and engineering systems like closed containers or controlled laboratory experiments.
- Behaves independently of external influences, as long as energy is not considered.

Physical or Abstract (Conceptual System)

Physical Systems: Have tangible, physical components. For example, a car with mechanical parts like engines and wheels.
Abstract Systems: Exist conceptually and may not have a physical presence. For example, a computer operating system that manages software and hardware but doesn't have a physical form.

Natural & Artificial Systems

Natural Systems:

Natural systems are those that exist in the natural world, without human intervention or design. Natural systems are governed by the laws of nature and have evolved over time. Examples of natural systems include:

- *Ecosystems:* Such as a forest, a coral reef, or a river system.
- *Climate Systems*: Like the global climate, weather patterns, and the water cycle.
- *Biological Systems*: Such as the human body, with its complex organs and processes.
- *Geological Systems*: Like volcanoes, earthquakes, and plate tectonics.

Artificial Systems:

Artificial systems, on the other hand, are created and designed by humans to serve specific purposes or functions. They are not naturally occurring and are typically the result of human engineering, technology, or organization.

Examples of artificial systems include:

- Transportation Systems: Such as highways, railways, airports, and traffic control systems.
- Communication Systems: Like the internet, telephone networks, and social media platforms.
- Manufacturing Systems: Such as assembly lines in factories and automated production systems.
- Economic Systems: Such as the stock market, banking systems, and economic models.
- Computer Systems: Including hardware, software, and networks.

Deterministic and Probabilistic Systems:

Deterministic Systems: Follow predictable cause-and-effect relationships. For example, the motion of planets in the solar system can be predicted based on physical laws.
Probabilistic Systems: Involve elements of chance and uncertainty. For example, weather forecasting uses probabilistic models due to the inherent uncertainty in weather patterns.

Open-Loop and Closed-Loop (Feedback) Systems:

Open-Loop Systems: Operate without feedback, meaning they do not adjust their actions based on results. For example, a basic timer that turns a light on and off at specific times.
Closed-Loop Systems: Use feedback to adjust their behavior based on input or performance data. For example, a thermostat in a heating system that regulates temperature based on room conditions.

Continuous and Discrete Systems:
Continuous Systems: Deal with continuous data or variables. For example, the flow of water in a pipe or analog signals in electrical circuits.
Discrete Systems: Work with discrete data or variables, often represented in a digital form. For example, the digital logic circuits in a computer that process binary data (0s and 1s).

Linear and Non-Linear Systems:
Linear Systems: Follow linear relationships between inputs and outputs. Changes in inputs have a proportional effect on outputs. For example, Ohm's law in electrical circuits.
Non-Linear Systems: Do not follow linear relationships, and small changes in inputs can lead to complex, non-proportional changes in outputs. For example, chaotic systems like weather patterns.

System Development

Is the process of defining, designing, creating, and implementing organized and integrated systems to address specific needs or solve problems.

Project Manager:

Responsibilities:

- Overall project leadership and management.
- Planning and defining project scope, goals, and deliverables.
- Resource allocation and task assignment.
- Risk assessment and management.
- Monitoring progress and ensuring project stays on schedule and within budget.
- Communication with stakeholders and team members.

Business Analyst:

Responsibilities:

- Gathering and analyzing business requirements.
- Defining project objectives and scope.
- Documenting user stories, use cases, and specifications.
- Bridging communication between technical and non-technical team members.

Project Sponsor:

Responsibilities:

- Providing project funding and resources.
- Championing the project's goals and objectives.
- Assisting in stakeholder management and issue resolution.
- Ensuring alignment of the project with organizational strategies.

Technical Lead/Architect:

Responsibilities:

- Designing the technical architecture of the project.
- Overseeing the development process.
- Making technical decisions and ensuring the project meets technical standards.
- Mentoring and guiding developers.

Developers/Programmers:

Responsibilities:

- Writing code and developing software or solutions.
- Collaborating with the team to implement features and functionality.
- Testing and debugging code.
- Adhering to coding and development standards.

Quality Assurance (QA) Tester:

Responsibilities:

- Developing and executing test plans.
- Identifying and reporting defects and issues.
- Ensuring the quality and reliability of the project.
- Collaborating with the development team to resolve issues.

Designers (UI/UX):

Responsibilities:

- Creating user interface (UI) and user experience (UX) designs.
- Ensuring the project is user-friendly and visually appealing.
- Collaborating with developers to implement design elements.

Data Analyst/Database Administrator (DBA):

Responsibilities:

- Managing project data and databases.
- Data modeling and optimization.
- Ensuring data integrity and security.
- Supporting data-related requirements of the project.

Subject Matter Experts (SMEs):

Responsibilities:

- Providing domain-specific knowledge and expertise.
- Offering insights into industry standards and best practices.
- Assisting with requirement gathering and validation.

Stakeholders:

Responsibilities:

- Defining project objectives and requirements.
- Providing feedback and approval at key project milestones.
- Ensuring alignment with business goals.
- Supporting the project throughout its lifecycle.

There are three main categories of System Development Life Cycle (SDLC) approaches typically include *predictive (or traditional), adaptive (or Agile),* and *object-oriented approaches*. These approaches encompass various methodologies and models used in software development. Here's a brief explanation of each category:

Predictive (Traditional) Approaches:

Is an approach whereby its methodologies follow a structured and sequential process for managing and executing projects. These approaches are often used in situations where the project's requirements are well-defined and unlikely to change significantly during development

Advantages:

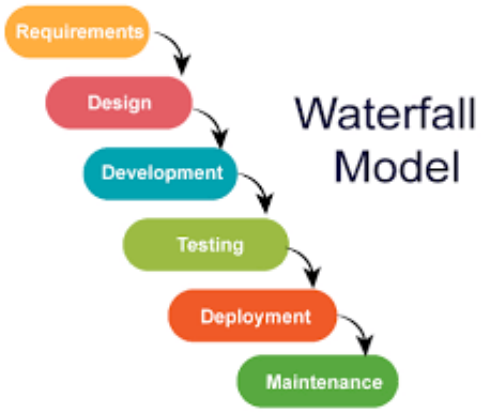
- Structured and Well-Documented: These approaches emphasize thorough documentation, making it easier to manage and understand the project's progress.
- Clear Project Phases: Phases are well-defined and typically have distinct objectives, which can provide clarity on project status.
- Predictable Timelines: With detailed planning, timelines and milestones are more predictable, making it easier to manage resources and expectations.
- Well-Suited for Stable Requirements: Predictive approaches work well when project requirements are well-defined and unlikely to change significantly.
- Risk Assessment: These approaches often include comprehensive risk analysis and management at an early stage.

Disadvantages

- Rigidity: One of the main drawbacks is the rigidity of these approaches. Once a phase is completed, it's challenging to make changes without costly and time-consuming rework.
- Limited Adaptability: Predictive approaches are not well-suited for projects with evolving or unclear requirements. They may struggle to accommodate changing customer needs.
- Long Delivery Time: Due to the sequential nature of these approaches, the delivery of the final product often takes a considerable amount of time, which may not be acceptable in fast-paced markets.
- Higher Risk of Scope Creep: In cases where requirements change during development, there is a higher risk of scope creep, where new features are added without proper planning.
- Late Customer Feedback: In traditional models like the Waterfall, customer feedback often comes late in the project, making it challenging to address potential issues early in the process.
- Higher Upfront Costs: The extensive planning, documentation, and validation stages can lead to higher initial project costs.

Methodologies in Predictive Model

Waterfall Model: A sequential approach where each phase is completed before moving to the next. It is known to be the oldest model in software development. It's suitable for projects with well-defined requirements and limited changes.
Note: The outcome of each phase is called a *deliverable* or *end product*.



Advantages of Waterfall Model

1. The model is user friendly and understandable
2. Each phase has a clear end product and process review
3. System development goes one phase after another
4. It is suitable for small projects with clear and unambiguous requirements
5. Key points can be determined easily in development cycle

Disadvantages of Waterfall Model

- Rigidity: Once a phase is completed, it's challenging to revisit or make changes without extensive and costly rework.
- Limited Adaptability: The Waterfall Model is not well-suited for projects with evolving or unclear requirements. It does not accommodate changes well and can lead to difficulties if customer needs evolve during the project.
- Late Feedback: Customer feedback typically comes late in the process, often after the product is built.
- Long Delivery Time: The sequential nature of the Waterfall Model means that the final product is not delivered until all phases are complete.
- Higher Risk of Scope Creep: In cases where requirements change during development, there is a higher risk of scope creep, where new features or changes are added without proper planning, impacting timelines and budgets.

- Resource-Intensive: The Waterfall Model requires extensive planning, documentation, and validation at each stage.

- Complex and Costly Corrections: If defects or issues are identified late in the project, correcting them can be complex, time-consuming, and expensive. The cost of fixing a problem typically increases as the project progresses.

- Not Suitable for All Projects: It is not well-suited for projects where a working prototype or product is required early in the development process, such as projects with innovative or exploratory aspects.

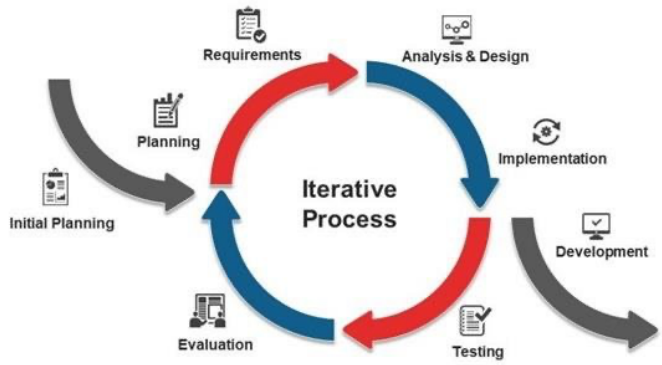
- Limited Stakeholder Involvement: Stakeholders are generally involved only at the beginning and the end of the project, potentially leading to misunderstandings or dissatisfaction with the final product

The Iterative Model

The iterative model in system development is a software development methodology that focuses on incremental and repetitive development.
Iterative development is when teams gradually build up the features and functions but don't wait until each of these is complete before releasing. They release a basic version of each feature and the add to that feature in subsequent iterative releases, usually based on feedback from the basic version released.

Each iteration goes through the following stages:

- *Planning:* Defining the scope, objectives, and features for the iteration.
- *Design and Development*: Designing and implementing the features specific to that iteration.
- *Testing*: Testing the features within that iteration.
- *Review and Feedback*: Gathering feedback from stakeholders and evaluating the iteration's results.
- *Adjustment and Improvement*: Using feedback to make necessary adjustments and improvements to the project.



Advantages of the Iterative Model:

- ***Flexibility:*** It allows for changes and improvements as the project progresses.
- ***Early Feedback:*** Stakeholders see partial results early, which helps in course correction.
- **Risk Management:** Issues are identified and resolved early, reducing project risks.
- **Customer Involvement:** Continuous collaboration ensures the product aligns with customer needs.
- **Higher Quality:** Testing and refinement throughout lead to a higher-quality final product.

Disadvantages of the Iterative Model

- *Complexity:* Managing multiple iterations can be complicated, especially for larger projects.
- Higher Costs: Continuous development and customer involvement may increase project expenses.
- Time-Consuming: The iterative process can take longer than a linear approach due to multiple iterations.
- Uncertain Timelines: It can be challenging to predict an exact project completion date.
- Resource Requirements: Continuous involvement of stakeholders and resources can be intensive, requiring a dedicated team and customer availability.
- Scope Creep: There's a risk of scope expansion if changes and new features are continuously introduced without clear control.

Adaptive SDLC:

Adaptive SDLC is a software development approach that prioritizes flexibility, responsiveness to change, and continuous improvement. It acknowledges that software requirements are often unclear or may evolve over time.

Key Characteristics:

- Iterative and Incremental: Adaptive SDLC divides the project into small iterations, with each iteration delivering a partial, functional product.
- Collaboration: It emphasizes continuous collaboration with stakeholders, including customers and end-users.
- Customer-Centric: The focus is on delivering value to customers and responding to their needs.

Advantages:

- *Adaptability:* Adaptive SDLC can accommodate changing requirements, making it suitable for projects with evolving needs.
- *Customer Involvement:* Continuous collaboration with customers results in a product that better aligns with their expectations.
- *Early Delivery:* Stakeholders see a working product early, providing transparency and the opportunity for early feedback.
- *Risk Mitigation:* Regular feedback and adjustments help identify and address risks early in the process.

Challenges/disadvantages:

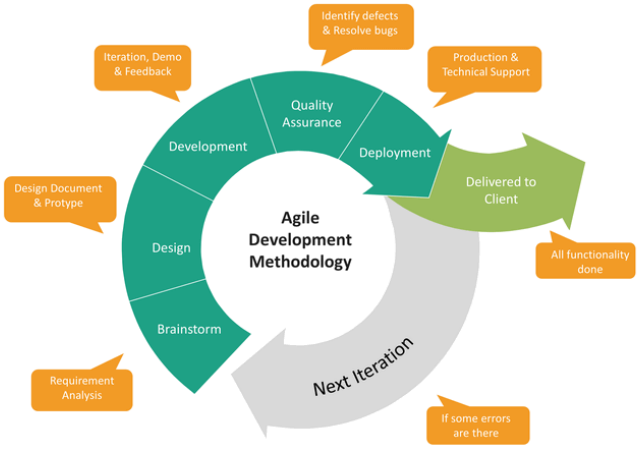
Complexity: Managing multiple iterations and ongoing customer involvement can be complex.
Resource-Intensive: Continuous collaboration and development require dedicated resources and may increase costs.
Uncertain Timelines: The adaptive approach can make it challenging to predict project completion dates.

When to Use:

Adaptive SDLC is suitable for projects with evolving or unclear requirements.
It is ideal for customer-centric projects where continuous feedback and collaboration are essential.

Agile Model

Agile is an approach to software development that emphasizes collaboration, adaptability, and delivering value in small, frequent increments. It's like building a puzzle one piece at a time, with a focus on flexibility and customer satisfaction.



Key Characteristics:

Iterative Development: Work is divided into small iterations, typically lasting 2-4 weeks. Each iteration delivers a piece of working software.

Customer Collaboration: Regular interaction with customers and stakeholders ensures that the product meets their needs.

Adaptability: Agile embraces change. Requirements can evolve, and the project can adapt without a lot of rework.

Agile Methodologies/Models:

Agile is not just one model; it's a family of methodologies. Common Agile models include Scrum, Kanban, and Extreme Programming (XP).

How Agile Works:

Sprints (in Scrum): Agile often works in time-boxed periods called "sprints." During a sprint, a team focuses on a set of prioritized tasks and aims to complete them.

Daily Stand-ups: Teams have short daily meetings to discuss progress, challenges, and next steps.

Frequent Demos: After each sprint, a demo or review is held to showcase what's been achieved to stakeholders.

Retrospectives: Teams reflect on what went well and what could be improved in a sprint, aiming for continuous enhancement.

Applications:

Agile is used in various software development projects, including web development, mobile app development, and product development. It's also applied in non-software fields, such as marketing and project management.

Advantages:

- *Customer Satisfaction:* Agile prioritizes customer feedback, leading to products that better meet customer needs.
- *Adaptability:* It allows for changes even late in the project.
- *Faster Delivery:* Incremental releases mean you get working features sooner.
- *Transparency:* Regular updates keep all stakeholders informed.

Disadvantages:

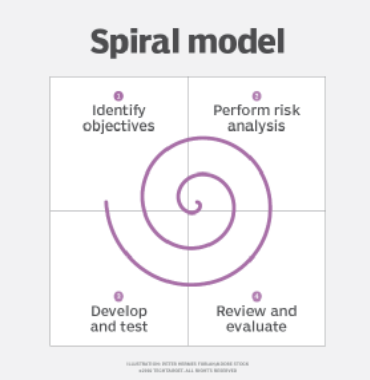
- Complexity: Agile can be challenging to manage for large or complex projects.
- Resource-Intensive: Continuous involvement and frequent meetings can be resource-intensive.
- Uncertain Timelines: It can be difficult to predict exact project completion dates.

Agile in a Nutshell:

Agile is like building a jigsaw puzzle. You start with a few pieces, consult others as you go along, and adapt if you find a better way to fit the pieces together.

Spiral Model :

The Spiral Model is a flexible approach to software development that's like a series of cycles or loops. Each cycle includes planning, building, and testing. As you go around the spiral, the product becomes more complete.



Key Characteristics:

- ***Iterative:*** It repeats the same steps (planning, building, testing) in cycles.
- ***Risk Management:*** It focuses on identifying and managing risks early in the process.
- ***Customer Feedback:*** Stakeholder feedback is essential and can lead to changes in each cycle.

Advantages:

- ***Risk Reduction:*** It's good at handling projects with lots of uncertainty or risks because it addresses them early.
- ***Flexibility:*** Changes are expected and can be easily incorporated.
- ***Quality:*** Continuous testing and refinement lead to higher-quality results.

Disadvantages:

- Complexity: Managing multiple cycles can be challenging.
- Time-Consuming: It can take more time than some other models due to repeated cycles.

Application in Real-World Projects:

- Large and Complex Systems: The Spiral Model is suitable for big projects where risks are high, like developing a new operating system.

- High-Stake Projects: It's used in critical systems, such as healthcare software, where safety is a priority.

- Evolving Projects: When requirements change often, like in e-commerce platforms, the Spiral Model helps adapt.

Prototyping Model:

The Prototyping Model is like making a rough draft of your project before creating the final version. You build a basic version of the software, get feedback, and then make improvements until you get the final product.

Key Characteristics:

- Build and Improve: You create a simple version of the software, gather feedback, and keep refining it.

- Customer Collaboration: Users provide feedback, which shapes the final product.

Advantages:

- User-Centered: It ensures the software meets user needs because users are involved from the start.

- Early Feedback: Quick feedback means you can make changes early, reducing the risk of costly mistakes.

- Reduced Misunderstandings: Prototypes help clarify requirements, reducing misunderstandings between developers and users.

Disadvantages:

- Time-Consuming: Building and revising prototypes can be time-intensive.

- May Oversimplify: Early versions might not capture the complexity of the final product.

Applications in Real-World Projects:

- User Interface Design: Prototyping is commonly used in web and mobile app design to create a visual representation of the user interface.

- Custom Software: It's used for developing specialized software, like inventory management systems, tailored to a specific organization's needs.

- Complex Systems: In projects like large-scale e-commerce platforms, prototypes help ensure the system's functionality aligns with user requirements.

Object-Oriented Approach Model:

The object-oriented approach is a software development model that organizes software into reusable "objects," which contain both data (attributes) and functions (methods).

Key Characteristics:

Objects: Everything in the software is represented as an object, which can be used and reused.
Encapsulation: Objects hide their internal details and expose only what's necessary.
Inheritance: Objects can inherit attributes and methods from other objects, promoting code reuse.
Polymorphism: Objects of different types can be treated as if they were of a common type.

Advantages:

- **Reusability:** Objects can be reused in different parts of a project, saving time and effort.
- **Modularity:** Software is organized into manageable, independent units (objects).
- **Ease of Maintenance**: Changes can be made to individual objects without affecting the entire system.

Disadvantages:

- Complexity: Object-oriented systems can be complex, requiring careful design and planning.
- Learning Curve: Developers may need to learn new concepts and techniques.
- Performance Overhead: There can be a small performance overhead due to the use of objects.

Real-World Examples:

Java: A popular programming language that follows the object-oriented approach. For example, objects can represent real-world entities in a banking system (e.g., customer, account).

Video Games: Object-oriented programming is used extensively in video game development. Game characters, items, and interactive elements can all be represented as objects.

Social Media Platforms: User profiles, posts, comments, and likes can be objects in social media systems, allowing for easy updates and interactions.