SOFTWARE PROCESS MODEL

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Software development life cycle (SDLC) is a structured process that is used to design, develop, and test good-quality software. SDLC, or software development life cycle, is a methodology that defines the entire procedure of software development step-by-step. The goal of the SDLC life cycle model is to deliver high-quality, maintainable software that meets the user's requirements. SDLC in software engineering models outlines the plan for each stage so that each stage of the software development model can perform its task efficiently to deliver the software at a low cost within a given time frame that meets users requirements.

SDLC : https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/

WATERFALL MODEL – sequential software development model

STEPS OF THIS MODEL

1. Requirement Analysis
    - Address the problem
    - Identify the feasible and non-feasible requirements
    - Identify how the software will meet the customer requirements
2. Design
    - Create logical and physical design of the project
3. Coding
    - Convert the design in actual running software
    - Design is split into blocks, and blocks are converted to code modules one after another
4. Testing
    - Test the software if it fulfils the requirements set at requirement analysis phase
5. Deployment

    - Once the previous steps are approved, it is deployed to the production environment.
6. Maintenance
    - It involves fixing any issues that arise after the software has been deployed and ensuring that it continues to meet the requirements over time.

WHEN TO CHOOSE WATERFALL MODEL

- Requirements are well known
- Small scale and short-term project
- Resources are available and trained
- Technological tools required are not dynamic, instead are stable

ADVANTAGES AND DISADVANTAGES OF WATERFALL MODEL

ADVANTAGES

1. Simple to use and easy
2. Stages go one by one, so sudden changes can't create confusions
3. All changes are done only in development stage, so no need to get back and change everything

DISADVANTAGES

1. While completing a stage, it freezes all the subsequent stages
2. No way to verify the design
3. Once in testing phase, no more features can be added
4. Code Reuse not possible

WHY CODE REUSE IS NOT POSSIBLE IN WATERFALL MODEL?

Code reuse is challenging in the Waterfall model because it follows a rigid, sequential flow where each phase (requirements, design, implementation, testing, etc.) is completed before moving to the next. There is little to no scope for iteration or revisiting earlier stages. As a result, identifying reusable components or integrating reusable code becomes difficult, since the focus is on completing predefined tasks rather than building adaptable, modular code. Additionally, late discovery of reusable opportunities often cannot be incorporated without disrupting the project's timeline or structure.

Waterfall Model: https://www.geeksforgeeks.org/waterfall-model/

V MODEL – sequential software development model

STEPS OF THIS MODEL

Verification - Validation

- Requirement Analysis - Acceptance Testing
- Architecture Design - System Testing
- Component/Module Design - Integration Testing
- Code Generation - Unit Testing

WHEN TO CHOOSE V MODEL

- Clear, stable requirements.
- Projects with high quality and safety needs.
- Limited or no requirement changes.
- Smaller, well-defined projects.

ADVANTAGES AND DISADVANTAGES OF V MODEL

ADVANTAGES

1. Along with waterfall advantages, it emphasizes planning for verification and validation of the product from the very beginning of requirement collection.
2. Test activities planned before testing
3. Saves time over waterfall, higher chance of success

DISADVANTAGES

1. Changes are not welcomed
2. Software developed at the end of all phases, so no dummy prototypes can be found
3. If any test fails, then test document and code both needs to be updated

Verification – Building the product in the right way is called verification

Validation – Building the right product is called validation

V Model : https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/

CHOOSE V-MODEL OVER WATERFALL WHEN:

- Testing needs to be done in parallel with each development phase.
- Quality and safety are critical (e.g., in regulated industries).
- Requirements are stable, with no expected changes.

CHOOSE WATERFALL OVER V-MODEL WHEN:

- The project is larger, with a focus on sequential, phase-by-phase completion.
- Minimal testing is required until the final stages.
- Requirements are clear, but testing at each phase isn't necessary.

V Model vs Waterfall Model : https://www.geeksforgeeks.org/difference-between-v-model-and-waterfall-model/

EVOLUTIONARY PROCESS MODEL

- Can change requirements as you want
- Can go back to previous phases, such as after coding we can go back to communication phase for requirement collection again.

INCREMENTAL PROCESS MODEL

- Communication
- Planning
- Modeling (analysis, design)
- Construction (code, test)
- Deployment (delivery, feedback)

WHEN TO CHOOSE INCREMENTAL PROCESS MODEL

- You are ready with thin slices of the overall software pieces
- Customer wants prototype version of the software from the beginning of the project
- Parallel stages such as requirement collection, planning etc. can take place
- Increments needs to be prioritized by customers, so better chance of success
- Better for small or medium size projects

ADVANTAGES AND DISADVANTAGES OF INCREMENTAL PROCESS MODEL

ADVANTAGES

- Delivers functional software in stages, allowing earlier product releases.
- Easier to identify and address risks early, improving project success.
- Flexible for adding or modifying features in future increments.

DISADVANTAGES

- Requires careful planning to integrate increments effectively.
- Late-stage changes to earlier increments may be costly or complex.
- Can result in an inconsistent user experience if increments are not well-aligned.

Incremental Process Model : https://www.geeksforgeeks.org/software-engineering-incremental-process-model/

ITERATIVE PROCESS MODEL

Iterative Model : https://www.geeksforgeeks.org/software-engineering-iterative-waterfall-model/

WHEN TO CHOOSE ITERATIVE PROCESS MODEL

- Requirements are not fixed
- Technological tools or requirements are not identified yet
- Instead of fixed time, quality of the features is refined with time

- Customer feedback with repetitive iterations increase the product quality
- Better for long-term and complex projects

ADVANTAGES AND DISADVANTAGES OF ITERATIVE PROCESS MODEL

ADVANTAGES

- Allows continuous refinement based on feedback from each iteration.
- Easier to incorporate changes and address evolving requirements.
- Enhances risk management by addressing issues in smaller cycles.

DISADVANTAGES

- Can be time-consuming if too many iterations are needed.
- May lead to scope creep if requirements are not well-controlled.
- Requires robust testing to ensure consistency across iterations.

KEY DIFFERENCE BETWEEN INCREMENTAL AND ITERATIVE MODELS

Incremental : Focuses on delivering small, fully functional parts of the system step by step.

Iterative : Focuses on refining and improving the system in cycles, where each iteration builds upon the previous one.

INCREMENTAL PROCESS MODEL VS. WATERFALL MODEL

CHOOSE INCREMENTAL OVER WATERFALL WHEN:

- Early delivery of functional software is a priority.
- Requirements are likely to evolve, or feedback is needed after each stage.
- The project can benefit from phased delivery (e.g., different user groups need different features).

CHOOSE WATERFALL OVER INCREMENTAL WHEN:

- Requirements are well-defined, stable, and unlikely to change.

- The project must be completed in a single, sequential pass.
- Strict documentation and regulatory standards are required from the start.

Incremental vs Waterfall Model : https://www.geeksforgeeks.org/difference-between-waterfall-model-and-incremental-model/

ITERATIVE PROCESS MODEL VS. WATERFALL MODEL

CHOOSE ITERATIVE OVER WATERFALL WHEN:

- Requirements are not fully defined and need to evolve through feedback.
- The project is complex, requiring repeated refinement and testing.
- Regular user feedback is essential to improve each version.

CHOOSE WATERFALL OVER ITERATIVE WHEN:

- Requirements are stable and well-understood, minimizing the need for feedback.
- The project can be completed sequentially without repeated revisions.
- Clear deadlines and scope limit the possibility of repeated iterations.

Iterative vs Waterfall Model : https://www.geeksforgeeks.org/waterfall-vs-iterative-sdlc-model/

INCREMENTAL PROCESS MODEL VS. ITERATIVE PROCESS MODEL

CHOOSE INCREMENTAL OVER ITERATIVE WHEN:

- The project can be divided into independent, functional modules.
- Each increment represents a complete feature, allowing partial deployment.
- There’s a need for phased delivery, with functional modules provided at each stage.

CHOOSE ITERATIVE OVER INCREMENTAL WHEN:

- The product needs continuous improvement based on feedback, rather than feature completion.

- Requirements may change significantly, requiring multiple refinements.
- A prototype or proof of concept is needed to understand or refine requirements further.

Iterative vs Incremental Model : https://www.geeksforgeeks.org/iterative-vs-incremental-model-in-software-development/

REQUIREMENTS ENGINEERING

FUNCTIONAL REQUIREMENTS

These define what the software system **must do**. They **specify the behavior or functions** of the system under specific conditions. Essentially, they describe the interactions between the system and its users or other systems.

Characteristics:

- They are specific to the application's goals and tasks.
- Directly related to business processes or user needs.
- Describes inputs, outputs, and the behavior of the system.

Examples:

- The system must allow users to log in with a username and password.
- The application must send an email notification when a new user registers.
- The system must calculate the total cost of items in a shopping cart, including taxes and discounts.
- The software must support multi-user access to a shared database.

NON-FUNCTIONAL REQUIREMENTS

These specify **how** the system should perform, rather than what it should do. They define **quality attributes, constraints, or standards** the system must meet to ensure it functions well in a real-world environment.

Characteristics:

- Focus on the quality of the system.
- Often related to the system's environment or operational context.
- Can be difficult to measure precisely (often defined in terms like “response time should be less than X” or “scalable to Y users”).

Examples:

- The system must handle 500 transactions per second.
- The software must be available 99.9% of the time (i.e., availability).
- The application must load a page in less than 2 seconds (i.e., performance).
- The system must be able to run on both Android and iOS devices (i.e., portability).
- The application must be secure, ensuring user data is encrypted (i.e., security).

Key Differences:

Aspect	Functional Requirements	Non-Functional Requirements
Definition	Describes what the system will do (features, functionality).	Describes how the system will behave (performance, quality).
Aspect	Functional Requirements	Non-Functional Requirements
Focus	The actions or processes the system must support.	The qualities the system must possess (e.g., reliability).
Examples	User authentication, data processing, transaction handling.	System availability, performance, scalability, security.
Testability	Easy to test directly (e.g., Does the system allow login?).	More abstract, often tested indirectly (e.g., Does the system perform under load?).

Functional vs Non-functional Requirements : https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/

AGILE DEVELOPMENT PROCESS

WHY WE NEED AGILE?

It allows for continuous updates and flexibility, adapting quickly to changing requirements. This is essential for modern software, especially for live systems like Amazon or Facebook, where frequent updates and zero downtime are crucial for uninterrupted user experience.

WHAT IS AGILITY?

Agile is a set of principles and values. It is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software.

WHAT IS AGILE SOFTWARE DEVELOPMENT?

Agile software development is a conceptual framework for software engineering that promotes development iterations throughout the life cycle of the project. Software developed during one unit of time is referred to as an iteration, which may last from one to four weeks. These methods also emphasize working software as the primary measure of progress.

Agile Software Development : https://www.geeksforgeeks.org/software-engineering-agile-software-development/

HOW AGILE METHODOLOGY IS A COMBINATION OF ITERATIVE AND INCREMENTAL PROCESS MODELS?

Agile methodology combines iterative and incremental models by delivering small, functional parts of the software (incremental) while continuously refining and improving each part through repeated cycles (iterative). Each Agile sprint produces a working version of the product, with feedback used to enhance features and add new ones in the next iteration, enabling both gradual improvement and flexibility to adapt to change. Every iteration/cycle is called a sprint.

for _ in range 1 to n:

- Plan
- Design
- Build
- Test
- Review
- Launch

AGILE MANIFESTO

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

AGILE CHARACTERISTICS

The Agile methodology has distinct characteristics that define its approach to software development and project management. Here's an explanation of the listed Agile characteristics:

1. MODULARITY

What it is: Agile projects are divided into smaller, manageable units or modules.

Why it matters: This division makes it easier to focus on specific aspects of the project, allowing for more effective planning, development, and testing.

2. ITERATIVE

What it is: Agile development occurs in repeated cycles (iterations), each delivering a working version of the product.

Why it matters: This allows for continuous improvement and adaptation based on feedback from stakeholders and users.

3. TIME-BOUND

What it is: Agile works within fixed timeframes, such as sprints (typically 1-4 weeks), to deliver specific outcomes.

Why it matters: Time constraints ensure focused work, quick results, and regular progress evaluation.

4. INCREMENTAL

What it is: Agile delivers the product in small, functional increments rather than as a complete package at the end.

Why it matters: This allows teams to provide value early and gather feedback to refine the product in subsequent increments.

5. CONVERGENT

What it is: Agile teams work toward a shared goal, continuously refining and converging on the desired solution.

Why it matters: It ensures that the team aligns efforts, adapts to changes, and steadily progresses toward the final product.

6. PEOPLE-ORIENTED

What it is: Agile prioritizes individuals and interactions over processes and tools.

Why it matters: Recognizes that a motivated, skilled team is essential for project success, fostering innovation and collaboration.

7. COLLABORATIVE

What it is: Agile emphasizes teamwork, communication, and collaboration among all stakeholders, including developers, testers, and customers.

Why it matters: Collaboration ensures that the product meets user needs and adapts to changes in requirements or priorities.

AGILE METHODOLOGY

- Extreme Programming (XP)
- Agile Unified Process (AUP)
- Scrum

EXTREME PROGRAMMING (XP)

- **New version maybe built several times per day**
- **Increments are delivered to customers every 2 weeks**
- All tests must be run for every build and the build is only accepted if tests run successfully

Extreme Programming (XP) : https://www.geeksforgeeks.org/software-engineering-extreme-programming-xp/

XP PRINCIPLE OR PRACTICE

These are principles and practices of Extreme Programming (XP), an agile software development methodology focused on flexibility, responsiveness, and delivering high-quality software. Below is an explanation of each:

1. INCREMENTAL PLANNING

What it is: Breaking down the planning process into smaller, manageable iterations rather than planning everything upfront.

Why it matters: This allows flexibility to adapt to changing requirements and ensures frequent delivery of working software.

2. SMALL RELEASES

What it is: Delivering small, functional pieces of software to users frequently.

Why it matters: It enables quicker feedback from stakeholders and reduces the risk of delivering irrelevant or faulty software.

3. SIMPLE DESIGN

What it is: Designing the simplest solution that works for the current requirements without over-engineering for future possibilities.

Why it matters: Keeps the codebase understandable and maintainable, minimizing complexity.

4. TEST-FIRST DEVELOPMENT

What it is: Writing automated tests for features before implementing the actual functionality.

Why it matters: Ensures that the code meets the requirements and helps prevent defects by clarifying the goals upfront.

5. REFACTORING

What it is: A technique for continuously restructuring existing code by changing its internal structure without changing its external behavior. It reduces duplicate codes, breaking long classes/methods into small ones, appropriate variable naming and many more.

Why it matters: Keeps the code clean, reduces technical debt, and makes it easier to implement future changes.

6. PAIR PROGRAMMING

What it is: Two developers work together at one workstation, with one writing code (the "driver") and the other reviewing it (the "navigator").

Why it matters: Promotes knowledge sharing, reduces errors, and improves the quality of the code, less business cost.

7. COLLECTIVE OWNERSHIP

What it is: Any team member can work on any part of the codebase.

Why it matters: Increases flexibility, prevents bottlenecks, and fosters a shared sense of responsibility for the quality of the code.

8. CONTINUOUS INTEGRATION

What it is: Frequently merging all code changes into a shared repository and running automated tests to detect integration issues early.

Why it matters: Ensures the software works as expected and avoids integration problems.

9. SUSTAINABLE PACE

What it is: Working at a pace that can be maintained indefinitely, avoiding overworking or burnout.

Why it matters: Leads to a healthier team and consistent productivity over the long term.

10. ON-SITE CUSTOMER

What it is: Having a customer representative or stakeholder available on-site to provide immediate feedback and clarify requirements.

Why it matters: Ensures that the development team builds the right product by maintaining close communication with the customer.

XP WORKFLOW

1. Select user stories for this release
2. Break down stories to tasks
3. Plan release
4. Develop/Integrate/Test software
5. Release software
6. Evaluate system

UNIT TESTING

- Writes unit test for each method to identify problems and resolve it
- Test First Development (TDD) is used to ensure better code quality and less error. It refers to write test code first and after that write the development code.

AGILE UNIFIED PROCESS (AUP)

It's a simplified version of the Rational Unified Process (RUP)

Agile Unified Process (AUP) : https://www.agilelonestar.com/knowledge-base/agile-unified-process

AUP PHASES

- Inception –
  - Try to identify the business scope of the project, initial requirements and potential solution of the problem
- Elaboration –
  - Design and prove the solution architecture, more requirements can be extracted
- Construction –
  - Continuous implementation and testing (specially unit testing) in form of iterations
- Transition –
  - Lastly, deploy the system in production environment and adjust feedback.

AUP DISCIPLINES

- Business Modelling (requirements and design)
- Implementation
- Test
- Deployment
- Configuration Management –
  - Manage access to project artifacts/versions. This includes not only tracking artifact versions over time but also controlling and managing changes to them.
- Project Management –
  - Direct the activities that take place within the project. This includes managing risks, directing people (assigning tasks, tracking progress, etc.), and coordinating with people and systems outside the scope of the project to be sure that it is delivered on time and within budget.
- Environment –

    - Support the rest of the effort by ensuring that the proper process, guidance (standards and guidelines), and tools (hardware, software, etc.) are available for the team as needed.

SCRUM

Most widely used Agile software/web development method for project management. The full software is **delivered in 14-30 days** iterations.

Scrum : https://www.geeksforgeeks.org/scrum-software-development/

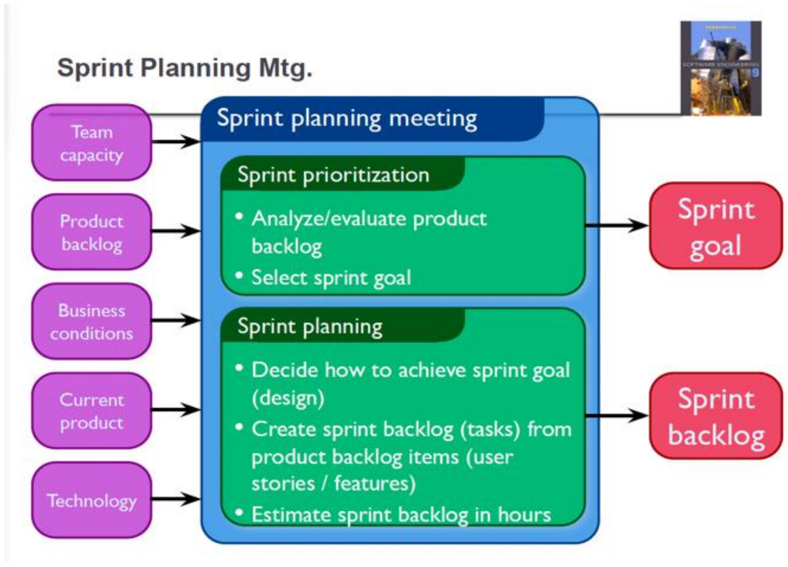
SCRUM CHARACTERISTICS

- Prioritized work is done
- Completion of backlog items
- Progress is explained
- Agile Software Development

SCRUM FRAMEWORK

- Roles –
    - Product owner –
        - Possibly a product manager or project sponsor who is typically responsible for defining the vision, managing the product backlog, deciding the release date, and prioritizing features to ensure that the team works on high-value items.
    - Scrum Master –
        - Typically, a project manager or team leader who facilitates Scrum practices, removes impediment/politics (obstacles), and supports the team in staying focused on their goals (productive). Acts as a bridge between the team and stakeholders.
    - Team –
        - A cross-functional group that develops, tests, and delivers the product. They work collaboratively to meet sprint goals and produce increments of value.

- Ceremonies –
    - Sprint planning –
        - A meeting to set goals and select backlog items for the upcoming sprint. The team and product owner define what can be delivered within the sprint.

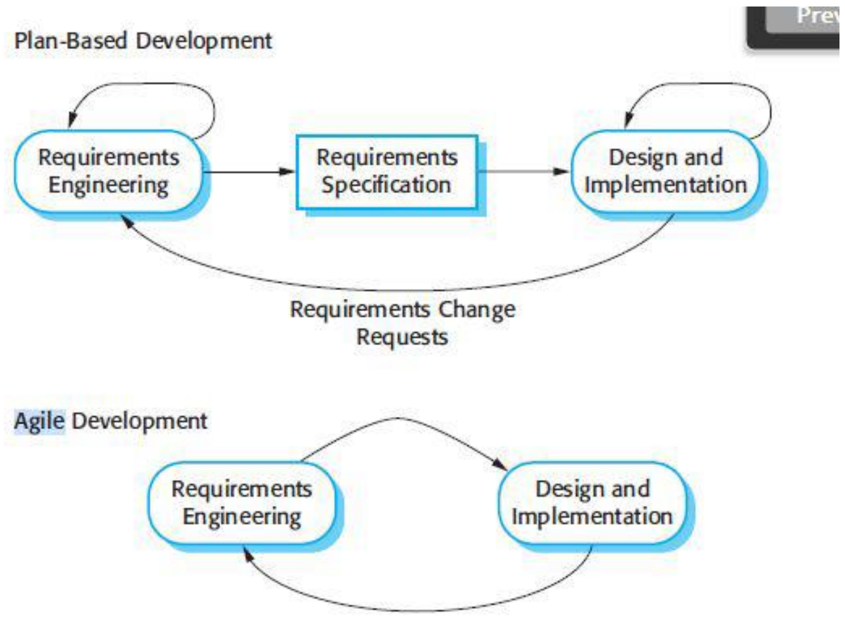

Sprint Planning Mtg.
Team capacity
Product backlog
Business conditions
Current product
Technology
Sprint planning meeting
Sprint prioritization
• Analyze/evaluate product backlog
• Select sprint goal
Sprint planning
• Decide how to achieve sprint goal (design)
• Create sprint backlog (tasks) from product backlog items (user stories / features)
• Estimate sprint backlog in hours
Sprint goal
Sprint backlog

- Sprint review and Sprint retrospective –
    - Sprint Review: Held at the end of the sprint to demonstrate completed work to stakeholders and gather feedback.

      Sprint Retrospective: A reflective meeting where the team discusses what went well, what didn't, and areas for improvement in the next sprint.
- Daily scrum meeting –
    - A brief, daily meeting where the team discusses progress, shares challenges, and plans work for the day. It keeps everyone aligned and on track.

- Artifacts –
    - Product backlog –
        - A prioritized list of all desired features, fixes, and requirements for the product, managed by the product owner.
    - Sprint backlog –
        - A subset of the product backlog items chosen for the current sprint, along with a plan for completing them.

- Burndown charts –
  - A visual representation of what work has been completed and what is left to complete. It helps the team track whether they're on pace to complete sprint goals.


Plan-Based Development
Requirements Engineering
Requirements Specification
Design and Implementation
Requirements Change Requests
Agile Development
Requirements Engineering
Design and Implementation
Prev

CLASS DIAGRAM IN UML

WHAT IS A CLASS?

A class is a blueprint or a template for creating objects. It defines the attributes (data/variables) and the methods (functions or behaviors) that the objects created from the class will have. A class defines what an object will look like and what it can do, but it does not represent a specific instance of an entity.

- In simpler terms: A class is like a recipe or a plan for creating objects. It defines the structure, but no actual data or behavior until instantiated.

WHAT IS AN OBJECT?

An object is an instance of a class. When a class is defined, no memory is allocated until you create an object from that class. An object represents a specific, tangible entity based on the class, and it holds actual data in its attributes.

- In simpler terms: An object is like an individual instance created from the class, with specific values or states.

Each object has an identity

- It can be referred individually
- It is distinguishable from other objects

ANALOGY:

Think of a class like a blueprint for a house. It defines the layout (rooms, windows, doors), but it's not a house you can live in. An object is like a specific house built from that blueprint — it has actual walls, rooms, and windows.

SUMMARY:

- A class is a template or blueprint that defines the structure (attributes) and behavior (methods) of objects.

- An object is a specific instance of that class with concrete data or state.

2 KINDS OF CLASSES DURING ANALYSIS

- Concrete
  - Class from the application domain
  - Example: Customer class and Employee class
- Abstract
  - Useful abstractions
  - Example: Person class

CLASSES

A class is a description of a set of objects that share the same attributes, operations, relationships and semantics.

Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate designated compartments.

ATTRIBUTES

An attribute is a named property of a class that describes the object being modeled.

- Properties of the class about which we want to capture information
- Represents a piece of information that is relevant to the description of the class within the application domain
- Only add attributes that are primitive or atomic types
- Derived attribute – calculated or derived from other attributes, denoted by placing slash(/) before name
- Attributes are usually listed in the form: attributeName : Type
- Example –
  - name : String
  - address : Address
  - birthdate : Date
  - /age : date
  - ssn : Id

OPERATIONS

- Represents the action or functions that a class can perform
- Describe the actions to which the instances of the class will be capable if responding
- Can be classified as a constructor, query or update operation
- Example – eat(), sleep(), work(), play()

- You can specify an operation by stating its signature: listing the name, type and default value of all parameters, and in the case of functions, a return type
- Example –
    - newEntry (n : Name, a : Address, p : PhoneNumber, d : Description)
    - getPhone (n : Name, a : Address) : PhoneNumber

VISIBILITY OF ATTRIBUTES AND OPERATIONS

- Public (+) → Visible everything to everyone
- Protected (#) → Visible to the owner class and its sub/child class
- Private (-) → Visible to the owner class only.

RELATIONSHIPS AMONG CLASSES

- Represents a connection between multiple classes or a class and itself
- Categories –
    - Association relationships –
        - Aggregation – denoted by a hollow diamond adornment on the association
        - Composition – denoted by a filled diamond on the association
    - Generalization relationships

ASSOCIATION RELATIONSHIP

- A bidirectional semantic connection between classes
- Type –
    - Name of relationship
    - Role that classes play in the relationship

GENERALIZATION RELATIONSHIPS

- Enables the analyst to create classes that inherit attributes and operations of other classes
- Represented by a kind of relationship

KEY DIFFERENCES BETWEEN ASSOCIATION AND GENERALIZATION:

Aspect	Association	Generalization (Inheritance)
Definition	Describes a relationship where objects of one class are associated with objects of another class.	Represents an inheritance relationship where a subclass inherits attributes and behavior from a superclass.
Type of Relationship	"Has-a" relationship (one class has a reference to another class).	"Is-a" relationship (subclass is a type of the superclass).
Example	A Person owns a Car (one-to-many, bidirectional).	A Car is a type of Vehicle.
Directionality	Can be unidirectional or bidirectional.	Unidirectional (subclass inherits from superclass).
Multiplicity	Can have various multiplicities (e.g., one-to-one, one-to-many).	Typically, one-to-one (a subclass inherits from one superclass).
Concept	Represents how different objects are related (e.g., through ownership, usage).	Represents inheritance and shared characteristics between classes.

UML Class Diagram : https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/

SOFTWARE ARCHITECTURE

- Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system.
- Software Architecture is how the defining components of a software system are organized and assembled. How they communicate each other. And how the constraint of the whole system is ruled by.

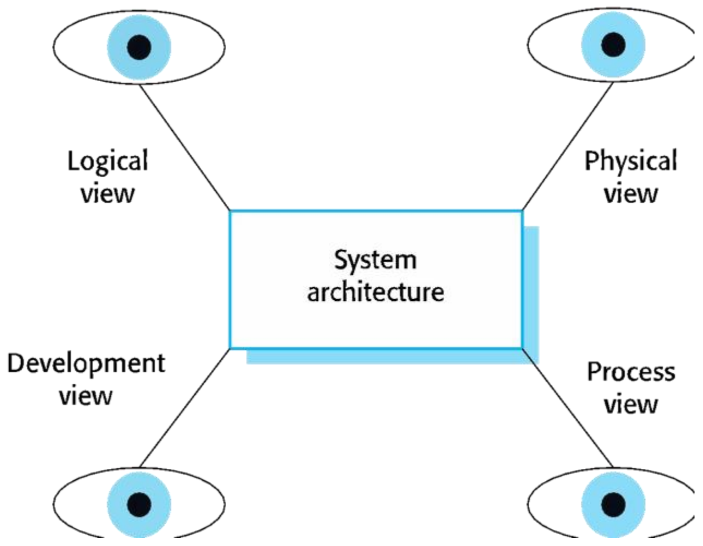
- The architectural model serves as an input to the development phase.

ADVANTAGES OF EXPLICIT ARCHITECTURE

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems

ARCHITECTURAL VIEW

- There are four views from which the architecture of a software can be observed
- Each architectural diagram only shows one view or perspective of the system.



4 + 1 VIEW MODEL OF SOFTWARE ARCHITECTURE

- A logical view, which shows the key abstractions in the system as objects or object classes. (class/state diagrams)

- A process view, which shows how, at run-time, the system is composed of interacting processes. (activity diagram)
- A development view, which shows how the software is decomposed for development. (Component/package diagram)
- A physical view, which shows the system hardware and how software components are distributed across the processors in the system. (Deployment diagram.)
- Related using use cases or scenarios (+1)

ARCHITECTURAL PATTERN

- An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.
- It's a solution to an existing and commonly occurring problem.
- Patterns include information about when they are and when they are not useful.

MVC PATTERN

- MVC goes for Model-View-Controller Pattern
- Separates presentation and interaction from the data handling logic.
- The system is structured into three logical components that interact with each other- Model, View and Controller

MVC : https://www.geeksforgeeks.org/mvc-framework-introduction/

CONTROLLER

- Handles all the user inputs through URL
- The interaction to an application starts here by the user interactions – mouse click, key press etc.
- Process http URL requests (GET, POST, PUT, DELETE)
  - GET: for getting a data
  - POST: for posting / inserting a data
  - PUT: for updating a data

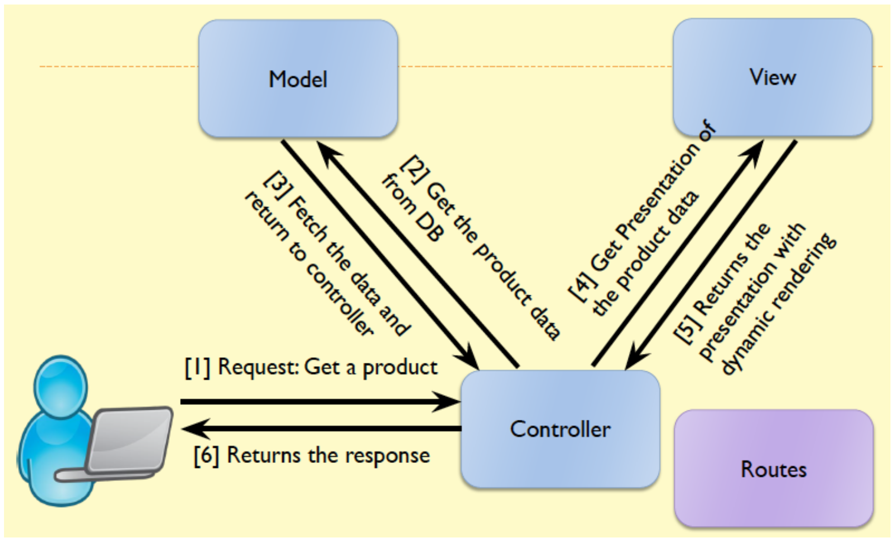
    - DELETE: for removing a data
- Communicates with both Model and View
- Contains all server-side logic
- Basically, **in Class diagram all the classes are controller**
- In the example – ProductController, UserController, AccountController etc.

MODEL

- It refers to the Data Related Logic
- Interaction with database (such as SELECT, INSERT, UPDATE, DELETE)
- It communicates with the controllers
- Can sometimes update or collaborate with the view (Depends on framework)
- In the example – Product, User, Transaction, Cart etc. are model classes.

VIEW

- What the end users see (UI)
- Usually Consists of html/css
- Communicates with the controller
- While coding is passed as dynamic values from the controller
- A controller can have multiple associated views.
- In the example – product.html, user.html to view a html file


Model
View
[2] Get the product data from DB
[3] Fetch the data and return to controller
[4] Get Presentation of the product data
[5] Returns the presentation with dynamic rendering
[1] Request: Get a product
[6] Returns the response
Controller
Routes

ROUTES

- Are urls to access a resource
- General structure –
    - http://domainName/{controller}/{action}/{id}
- http://YourApp.com/Users/Profile/25

WHEN TO USE MVC:

- Complex Applications –
    - Use MVC for applications with complex business logic and user interfaces, as it separates concerns for easier management.
- Scalable Projects –
    - Ideal for projects expected to grow, as MVC's modular structure supports scalability and easier maintenance.
- Multiple Views –
    - When the application requires multiple views for the same data (e.g., web and mobile versions), MVC enables efficient data sharing and reusability.
- Collaborative Development –
    - MVC is suitable for team projects, as developers can work on different components (Model, View, Controller) simultaneously without conflicts.
- Rapid Development and Testing –
    - Its separation of concerns simplifies debugging and testing, making it suitable for iterative development environments.

By dividing the application into distinct layers, MVC enhances flexibility, maintainability, and adaptability, especially for medium to large-scale projects.

ADVANTAGES OF MVC

- Codes are easy to maintain, and they can be extended easily.
- The MVC model component can be tested separately.
- The components of MVC can be developed simultaneously.

- It reduces complexity by dividing an application into three units. Model, view, and controller.
- It supports Test Driven Development (TDD).
- It works well for Web apps that are supported by large teams of web designers and developers.
- This architecture helps to test components independently as all classes and objects are independent of each other
- Search Engine Optimization (SEO) Friendly.

DISADVANTAGES OF MVC

- It is difficult to read, change, test, and reuse this model
- It is not suitable for building small applications.
- The inefficiency of data access in view.
- The framework navigation can be complex as it introduces new layers of abstraction which requires users to adapt to the decomposition criteria of MVC.
- Increased complexity and Inefficiency of data