**First Phase**

**A.Software engineering** is the discipline that applies engineering principles to the development of software. In simpler terms, it's a systematic approach to building high-quality, reliable, and maintainable software.

**Software engineering entails:-**

**A.1.Structured Development**: Software engineering follows a defined process, unlike random coding. This process typically involves phases like planning, requirement analysis, design, development, testing, deployment, and maintenance.

**A.2.Engineering Principles:** Software engineering applies engineering practices like requirement gathering, design, testing, and version control to ensure the software is well-designed, predictable, and can be maintained over time.

**A.3.Focus on Quality:** A core tenet of software engineering is building high-quality software. This means the software should be functional, reliable, efficient, secure, and maintainable.

**B.Software engineers** are the professionals who apply these principles. They use their knowledge of programming languages, algorithms, and software development methodologies to design, develop, test, and maintain software applications.

**B1.1 Difference Between Programmer and Software Engineer**

**B.1.1.A Programmer**: A programmer focuses primarily on writing code to achieve a specific functionality.

**B.1.2.B Software Engineer**: A software engineer takes a broader approach. They consider the entire software development lifecycle, from understanding requirements to designing, building, testing, deploying, and maintaining the software.

**C.Evolution of Software**

**C.1 Colossus Computing**- Colossus computer worked to break the Lorenz cipher during World War II: - **Early Days (1940s-1950s):**  - machine code, a series of complex instructions directly understood by the hardware.

**C.2 High-Level Languages (1960s-1970s)**: These languages used instructions closer to human language, requiring translation into machine code by compilers.

**C.3 Structured Development (1980s):** These methodologies emphasized planning, design, and coding phases to improve organization and reduce errors.

**C.4 Object-Oriented Programming (1990s-2000s):** Object-oriented programming (OOP) became a dominant paradigm.

**C.5 Modern Software Development (2000s-Present):**agile, Cloud computing, artificial intelligence, and machine learning

**D. SDLC**

**D.1 Planning and Requirement Analysis:**

**Activities:**

* Define project goals and scope.
* Identify stakeholders and their needs.
* Conduct feasibility studies (technical, economic, operational).
* Gather and analyze user requirements (what the software needs to do).
* Estimate project cost and timelin**e.**

**Project Charter:** Defines the project goals, scope, stakeholders, and high-level plan.

**Statement of Work (SOW):** Formal agreement outlining project deliverables, timelines, and costs (often used in client-contractor relationships).

**Business Requirements Document (BRD):** Captures the business needs and functionalities the software should address.

**User Requirements Document (URD):** Details the functional and non-functional requirements from the user's perspective.

**Use Case Documents:** Describe specific user interactions with the system and the expected outcomes.

**Feasibility Study**: Assesses the technical, economic, and operational feasibility of the project.

**D.2 Design   
Activities:**

* **Translate requirements into a technical blueprint.**
* **Design system architecture (overall system structure).**
* **Design user interface (UI) layout and user experience (UX) elements.**
* **Design data structures and databases.**
* **Create detailed specifications for different components.**

**Deliverables**

**System Architecture Document:** Outlines the overall system design, components, and their interactions.

**Software Design Document (SDD):** Details the technical design of the software, including algorithms, data structures, and class diagrams.

**User Interface (UI) Design Documents:** Specifications for the user interface layout, screens, and user experience (UX) considerations.

**Data Flow Diagrams (DFDs):** Visual representation of data flow through the system.

**Entity-Relationship Diagrams (ERDs):** Illustrate the relationships between data entities within the system.

**D.3 Development (Implementation):  
Activities:**

* **Developers write code based on design specifications.**
* **Develop and integrate application functionalities.**
* **Perform unit testing on individual code modules.**

**Deliverables**

**Source Code:** The actual code written by developers to build the software functionalities.

**Unit Test Cases:** Specifications for testing individual units or modules of the code.

**D.4 Testing  
Activities:**

* **Conduct rigorous testing to identify and fix bugs.**
* **Ensure the software meets all requirements.**
* **Different testing types might include:**
  + **Unit testing (individual code modules)**
  + **Integration testing (integrated components work together)**
  + **System testing (entire system functionality)**
  + **User Acceptance Testing (UAT) (real users test the software)**

**Deliverables**

**Test Plan:** Defines the overall testing strategy, test cases, and expected results.

**Test Cases:** Detailed instructions for testing specific functionalities or user scenarios.

**Test Reports:** Document test execution results, identified bugs, and resolutions.

**D.5 Deployment  
Activities:**

* **Release the software to the production environment (where users access it).**
* **Configure and manage the deployed software.**

**Deliverables**

**Deployment Plan:** Outlines the strategy for releasing the software to production, including configuration and rollback procedures.

**Release Notes:** Communicate new features, bug fixes, and other changes included in a specific software release.

**D.6 Maintenance**

**Bug Reports:** Documents encountered issues and their proposed fixes.

**Change Request Documents:** Formal requests for new features, modifications, or bug fixes.

**Knowledge Base Articles:** Document solutions to common problems or user guides for ongoing maintenance.

**C. Project Management:**

* **Activities:**
  + Oversee the entire development process.
  + Manage resources (people, budget, time).
  + Identify and mitigate risks.
  + Ensure project stays on track and meets deadlines.
  + Facilitate communication between stakeholders.
* **Deliverables:** Project plan, status reports, risk management plans, communication plans

**D . FlowChart and Pseudocode  
  
Pre-code planning, pseudocode, verify algorithm, and flowchart** are all techniques used by programmers to plan and design the logic of their program before they write the actual code.  
  
**1. Pre-code Planning**

This is the broad term encompassing all the activities you do before writing code. It involves understanding the problem you're trying to solve, defining the requirements, and designing the overall approach. Pre-code planning helps to:

* **Identify potential problems:** By thinking through the logic beforehand, you can often identify potential issues early on and adjust your approach.
* **Improve code quality:** A well-planned program is more likely to be well-structured, efficient, and maintainable.
* **Save time:** Spending time upfront planning can save you time and frustration in the long run by avoiding dead ends and re-work later in the coding process.

There are many pre-code planning techniques, and flowchart and pseudocode, which we'll discuss next, are two popular tools used in this phase.

**2. Pseudocode**

Pseudocode is a way of describing the steps of your program in a natural language format that resembles programming code, but it's not actual code that a computer can understand. It uses keywords like "if," "else," "while," and "for" to control the program flow, along with variable names and basic mathematical operations.

Here are some benefits of using pseudocode:

* **Easy to understand:** Because it uses familiar language constructs, pseudocode is easier for both programmers and non-programmers to understand compared to formal code.
* **Focuses on logic:** By using pseudocode, you can focus on the core logic of your program without getting bogged down in the specifics of a particular programming language.
* **Iterative development:** Pseudocode allows you to easily modify and refine your program's logic before writing any actual code.

**3. Verify Algorithm**

Verifying your algorithm is essential to ensure that your pre-code plan actually solves the problem you intend it to. There are a few ways to verify your algorithm:

* **Walkthrough:** Manually step through your pseudocode or flowchart using sample inputs and trace the program's execution. This helps identify any logical errors or unexpected behavior.
* **Test Cases:** Create a set of test cases with different inputs and expected outputs. Running your algorithm with these test cases helps ensure it handles various scenarios correctly.
* **Peer Review:** Have another programmer review your pseudocode or flowchart. A fresh perspective can help identify potential flaws or areas for improvement.

**4. Flowchart**

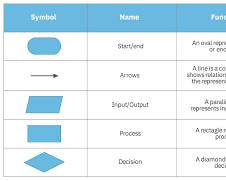
A flowchart is a visual representation of your program's logic using a set of symbols and arrows. Here are some common flowchart symbols:

* **Process:** A rectangle represents a processing step, like calculations or data manipulation.
* **Decision:** A diamond represents a decision point where the program chooses between different paths based on a condition.
* **Input/Output:** A parallelogram represents input or output operations, like getting user input or displaying data.

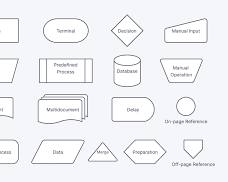
Flowcharts provide a clear visual representation of the program flow, making it easy to understand the logic and identify potential issues. Here are some advantages of using flowcharts:

* **Easy to understand:** Flowcharts are easy to understand for both programmers and non-programmers because they use visual elements.
* **Identify inefficiencies:** By visualizing the program flow, you can often identify inefficiencies or redundant steps.
* **Communication tool:** Flowcharts can be a useful communication tool to explain your program logic to others.

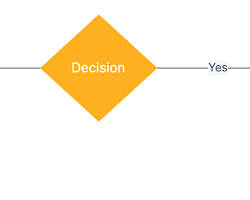
**FlowChart Keywords and Shapes -  
Terminator (Oval):**

* Definition: An oval shape representing the start or end point of a flowchart.
* When to Use: Mark the beginning and termination of the overall process.  
   [Opens in a new window www.smartdraw.com](https://www.smartdraw.com/flowchart/flowchart-symbols.htm) Flowchart Terminator symbol

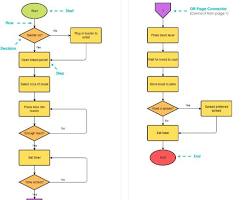
2**. Process (Rectangle):**

* Definition: A rectangle representing a specific step or action performed in the process.
* When to Use: Outline each task or function within the flowchart.  
   [Opens in a new window zenflowchart.com](https://www.zenflowchart.com/flowchart-symbols) Flowchart Process symbol

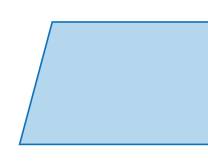
3. **Decision (Diamond):**

* Definition: A diamond shape representing a decision point where a condition is evaluated.
* When to Use: Indicate a point where a choice needs to be made based on a Yes/No or True/False condition. Branching arrows should lead from the diamond to the corresponding paths based on the decision outcome.  
   [Opens in a new window www.zenflowchart.com](https://www.zenflowchart.com/flowchart-symbols) Flowchart Decision symbol

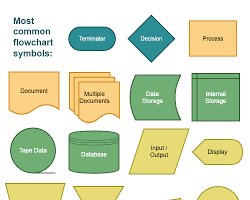
4. **Connector (Circle):**

* Definition: A circle used to connect flowchart elements that are separated due to space constraints.
* When to Use: Maintain flow continuity when elements are too far apart to connect with a straight line. A corresponding connector on another part of the flowchart (or a different page) shows where the flow resumes.  
   [Opens in a new window www.edrawmax.com](https://www.edrawmax.com/flowchart/flowchart-connector.html) Flowchart Connector symbol

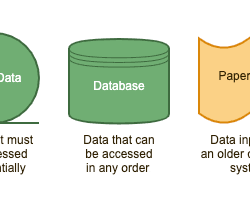
5. **Input/Output (Parallelogram):**

* Definition: A parallelogram representing an input or output operation in the process.
* When to Use: Show where data enters the flowchart (input) or where results are produced (output).  
   [Opens in a new window www.smartdraw.com](https://www.smartdraw.com/flowchart/flowchart-symbols.htm) Flowchart Input/Output symbol

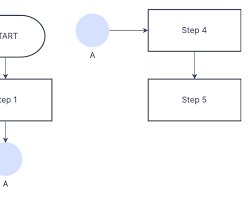
6. **Document (Rectangle with wavy bottom):**

* Definition: A rectangle with a wavy bottom signifying a document or printed output generated during the process.
* When to Use: Indicate the creation of a physical document or printed report.  
   [Opens in a new window gliffy.com](https://www.gliffy.com/blog/guide-to-flowchart-symbols) Flowchart Document symbol

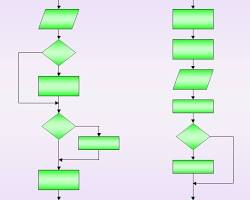
7. **Data Storage (Divided rectangle):**

* Definition: A rectangle divided diagonally to represent data storage elements like databases or files.
* When to Use: Show where data is stored or retrieved during the process.  
   [Opens in a new window www.gliffy.com](https://www.gliffy.com/blog/guide-to-flowchart-symbols) Flowchart Data Storage symbol

8. **Off-page Connector (Rectangle with arrow):**

* Definition: A rectangle with an arrow pointing in or out, indicating the flowchart continues on another page.
* When to Use: Maintain flow continuity when the flowchart doesn't fit on a single page. The arrow points to the corresponding symbol on the other page where the flow resumes.  
   [Opens in a new window www.zenflowchart.com](https://www.zenflowchart.com/blog/flowchart-connector-explanation-guide) Flowchart Offpage Connector symbol

9. **Terminal On-page Connector (Two dots):**

* Definition: Two dots used within a page to connect flowchart elements that are too far apart for a straight line.
* When to Use: Maintain flow on a single page by connecting distant elements with these dots. A corresponding two-dot symbol elsewhere on the same page shows where the flow continues.  
   [Opens in a new window www.edrawmax.com](https://www.edrawmax.com/flowchart/flowchart-connector.html) Flowchart Terminal Onpage Connector symbol

10. **Annotation (Any shape with text):**

* Definition: Any shape with explanatory text attached, used to provide additional information or comments about a step or decision.
* When to Use: Clarify specific steps, decisions, or data flows within the process whenever additional details are needed.

FlowChart - https://slickplan.com/diagram/symbols#:~:text=The%20circle%20flowchart%20symbol%2C%20or,where%20it%20continues%20in%20another.  
  
**Pseudocode Keywords list -  
1. Input/Output:**

* **READ:** Reads data from the user or an external source.
  + **Example:** READ number
* **PRINT/DISPLAY/OUTPUT:** Displays information on the screen.
  + **Example:** PRINT "The result is: ", result

**2. Control Flow:**

* **IF/ELSE:** Makes decisions based on conditions.
  + **Example:** IF age >= 18 THEN PRINT "You are eligible to vote." ELSE PRINT "You are not eligible to vote."
* **WHILE:** Repeats a block of code as long as a condition is true.
  + **Example:** WHILE counter < 10 DO PRINT counter; counter := counter + 1; END WHILE
* **FOR:** Repeats a block of code a specific number of times.
  + **Example:** FOR i := 1 TO 5 DO PRINT "Iteration: ", i; END FOR

**3. Variables:**

* **SET:** Initializes a variable with a value.
  + **Example:** SET name = "John"
* **INCREMENT/DECREMENT:** Increases or decreases the value of a variable by 1.
  + **Example:** counter := counter + 1 (increment) OR counter := counter - 1 (decrement)

**4. Functions:**

* **FUNCTION name(parameters):** Defines a reusable block of code.
  + **Example:** FUNCTION CalculateArea(length, width): RETURN length \* width; END FUNCTION

**5. Other Keywords:**

* **CALL:** Calls a previously defined function.
  + **Example:** CALL CalculateArea(5, 3)
* **RETURN:** Exits a function and optionally returns a value.
  + **Example:** RETURN result (in a function)
* **END:** Marks the end of a code block (e.g., IF statement, WHILE loop).

Re

* Flow Chart - link - https://www.codecademy.com/article/pseudocode-and-flowcharts