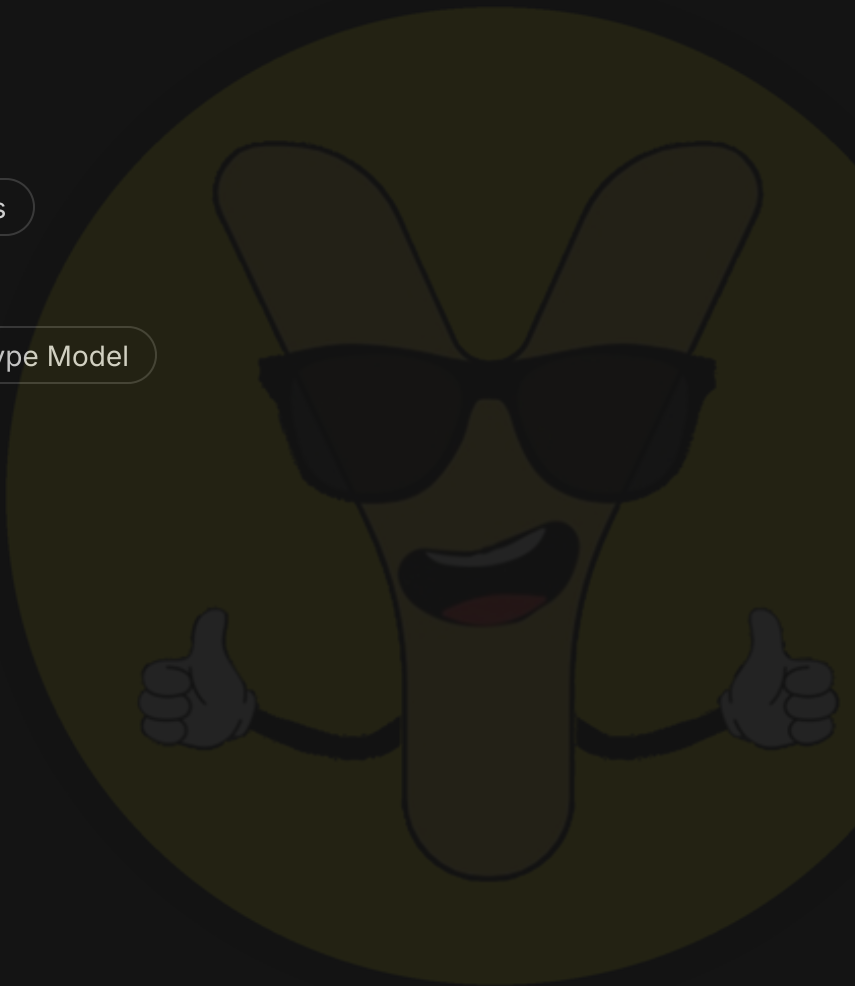
AKTU · 3RD YEAR · SEMESTER 6 · BCS601

Software Engineering

Unit 1 — Introduction & SDLC Models

- Introduction to Software Engineering
- Software Components
- Software Characteristics
- Software Crisis
- Software Engineering Processes
- Similarity and Differences from Conventional Engineering Processes
- Software Quality Attributes
- Software Development Life Cycle (SDLC) Models
- Waterfall Model
- Prototype Model
- Spiral Model
- Evolutionary Development Models
- Iterative Enhancement Models



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

Introduction to Software Engineering

SOFTWARE ENGINEERING

Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

- It involves applying engineering principles to software development to achieve high-quality, reliable, and maintainable software products.
- Focuses on systematic methods for producing software that is delivered on time and within budget.

Goals and Importance of Software Engineering

Software Engineering aims to address the challenges in software development and ensure quality outcomes.

- **Quality:** To produce high-quality software that meets user requirements and performs reliably.
- **Cost-Effectiveness:** To develop software within budget constraints, optimizing resource utilization.
- **Timeliness:** To deliver software products on schedule, avoiding delays.
- **Maintainability:** To create software that is easy to understand, modify, and enhance over its lifecycle.
- **Reliability:** To ensure software functions correctly under specified conditions without failure.
- **Efficiency:** To develop software that uses system resources (memory, CPU) effectively.

Software Components and Characteristics

Understanding software's nature helps in its engineering process.

- Software Components
  - **Definition:** Reusable, pre-built units of software that perform specific functions.
  - **Types:** Executable code (e.g., libraries, APIs), data (e.g., databases, configuration files), and documentation (e.g., user manuals, design documents).
- Software Characteristics
  - **Developed/Engineered, Not Manufactured:** Software is created, not assembled from components like hardware.
  - **Doesn't 'Wear Out':** Software does not degrade physically over time, but it can 'deteriorate' due to increasing complexity or changing requirements.
  - **Custom-Built:** A significant portion of software is custom-built, rather than off-the-shelf, to meet unique business needs.

Software Crisis: Causes and Impact

The 'software crisis' refers to the difficulties encountered in developing complex software systems.

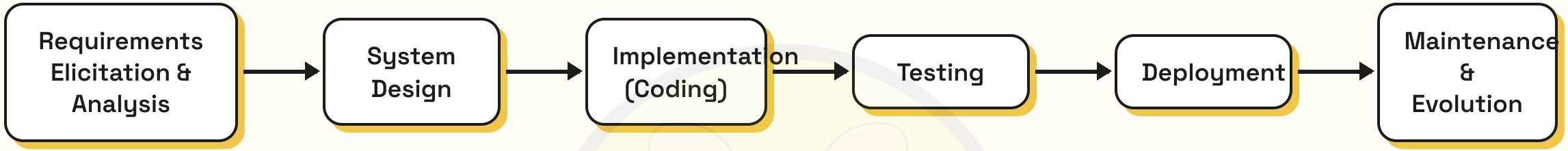
- Causes of Software Crisis (1960s-1980s)
  - **Late Delivery:** Projects frequently exceeded deadlines.
  - **Budget Overruns:** Development costs often far exceeded initial estimates.
  - **Low Quality:** Software was often unreliable, inefficient, and full of bugs.
  - **Difficult to Maintain:** Existing software was hard to modify or enhance.
  - **Poor Estimation:** Inaccurate estimations of time, cost, and resources required.
- Impact and Solutions
  - Led to the emergence of Software Engineering as a discipline.
  - Promoted the development of structured methodologies, tools, and standards.
  - Emphasized the need for systematic processes, project management, and quality assurance.

Software Engineering Processes

A software engineering process is a structured set of activities required to develop a software system.

- **Definition:** A framework for the activities, actions, and tasks that are required to build high-quality software.
- It defines the approach taken to develop software, from initial concept to deployment and maintenance.
- **Key Phases (Generic):**
- Software Specification
  - Defining what the software should do and its constraints.
- Software Design and Implementation
  - Designing the software architecture and writing the code.
- Software Validation
  - Checking that the software meets its specifications and user needs.
- Software Evolution
  - Modifying the software to meet changing customer and market requirements.

Generic Software Engineering Process Flow



A typical sequential flow of activities in a software development process.

Similarities with Conventional Engineering Processes

Software engineering shares fundamental principles with other engineering disciplines.

- **Systematic Approach:** Both emphasize a structured, methodical approach to problem-solving.
- **Design and Analysis:** Both involve detailed analysis of requirements and systematic design before construction.
- **Quality Focus:** Both aim to produce reliable, efficient, and high-quality products.
- **Project Management:** Both require planning, scheduling, resource allocation, and risk management.
- **Use of Tools and Standards:** Both rely on specialized tools, techniques, and industry standards to ensure consistency and quality.
- **Problem-Solving:** Both disciplines are essentially about solving real-world problems through systematic creation of artifacts.

Differences from Conventional Engineering Processes

Aspect	Software Engineering	Conventional Engineering (e.g., Civil, Mechanical)
Product Nature	Intangible, logical product; easily copied.	Tangible, physical product; difficult to copy.
Manufacturing	No 'manufacturing' phase; development is the creation.	Distinct manufacturing phase after design.
Wear and Tear	Does not physically wear out; deteriorates due to complexity/changes.	Physically wears out over time.
Complexity	High inherent complexity; often deals with abstract concepts.	Complexity often related to physical constraints and materials.
Changeability	Highly amenable to change; modifications can be frequent.	Changes are often costly and difficult once construction begins.
Replication Cost	Near-zero cost for replication.	Significant cost for each additional unit.
Testing	Testing is primarily logical and functional.	Testing involves physical stress, material properties, etc.

Software Quality Attributes

These attributes define the non-functional requirements and overall quality of a software system.

- **Functionality:** The degree to which the software performs its intended functions correctly.
- **Reliability:** The ability of the software to perform its required functions under specified conditions for a specified period.
- ****Usability:** The ease with which users can learn, operate, prepare input for, and interpret output from the software.
- **Efficiency:** The degree to which the software uses optimal resources (time, memory, CPU) to perform its functions.
- **Maintainability:** The ease with which the software can be modified to correct faults, improve performance, or adapt to a changed environment.
- **Portability:** The ease with which the software can be transferred from one hardware or software environment to another.
- **Security:** The degree to which the software protects information and data from unauthorized access or modification.

Software Development Life Cycle (SDLC) Models

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

SDLC is a structured framework that describes the stages involved in the development of a software product, from initial feasibility study to maintenance.

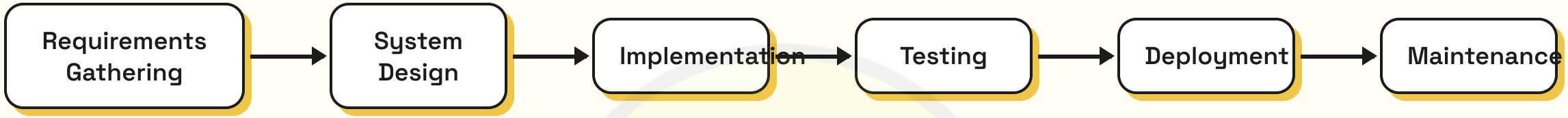
- It provides a roadmap for planning, creating, testing, and deploying an information system.
- Each phase of the SDLC has specific deliverables and entry/exit criteria.
- Different SDLC models exist to suit various project types and organizational needs.

Generic SDLC Phases

Most SDLC models incorporate a common set of fundamental phases.

- **1. Requirements Gathering & Analysis:** Understanding and documenting user needs and system specifications.
- **2. System Design:** Defining the overall architecture, components, interfaces, and data structures of the system.
- **3. Implementation (Coding):** Translating the design into actual executable code.
- **4. Testing:** Verifying that the software meets its requirements and is free of defects.
- **5. Deployment:** Releasing the software to the production environment for end-users.
- **6. Maintenance:** Ongoing support, bug fixing, enhancements, and adaptations to new environments after deployment.

Waterfall Model: Introduction and Diagram



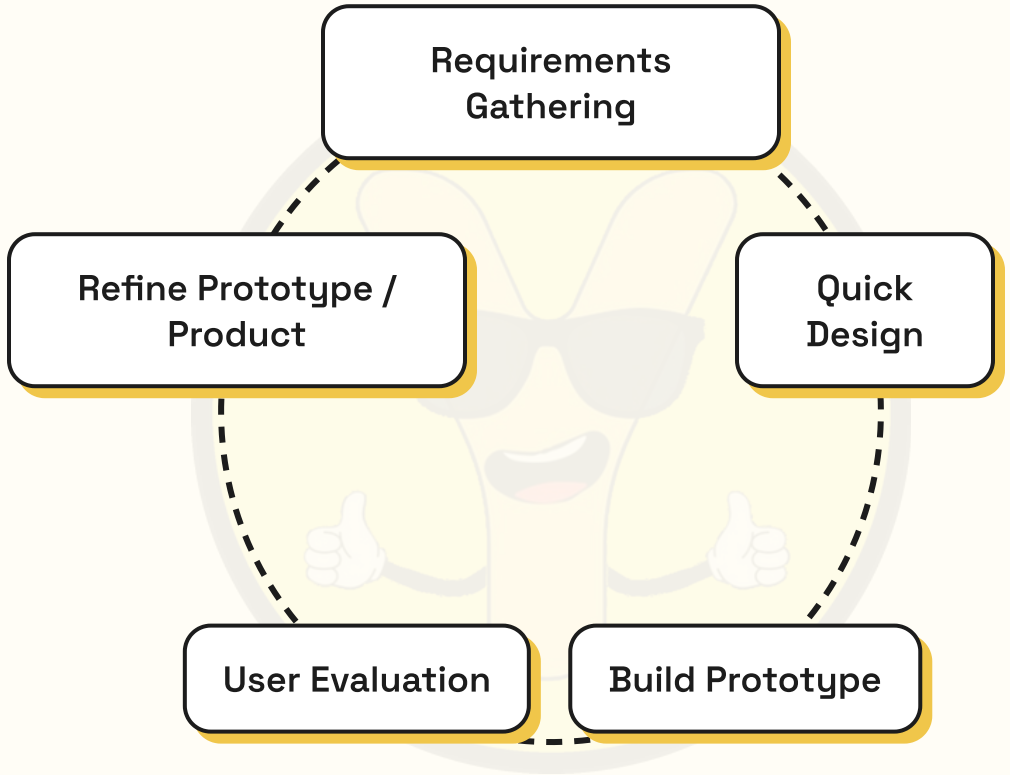
The Waterfall Model's sequential flow, with no overlapping or iteration between phases.

Waterfall Model: Advantages and Disadvantages

Understanding the pros and cons helps determine suitability for projects.

- Advantages
  - **Simplicity:** Easy to understand, use, and manage due to its rigid structure.
  - **Clear Deliverables:** Each phase has distinct deliverables and a review process.
  - **Well-Documented:** Emphasizes thorough documentation at each stage.
  - **Good for Small Projects:** Suitable for projects with well-defined and stable requirements.
- Disadvantages
  - **Inflexibility:** Difficult to accommodate changes once a phase is completed.
  - **Late Error Detection:** Errors are often found late in the lifecycle (during testing), making them costly to fix.
  - **No User Feedback:** Users are typically involved only at the requirements stage, leading to potential misalignment.
  - **High Risk:** Not suitable for complex or long-term projects with evolving requirements.

Prototype Model: Introduction and Diagram



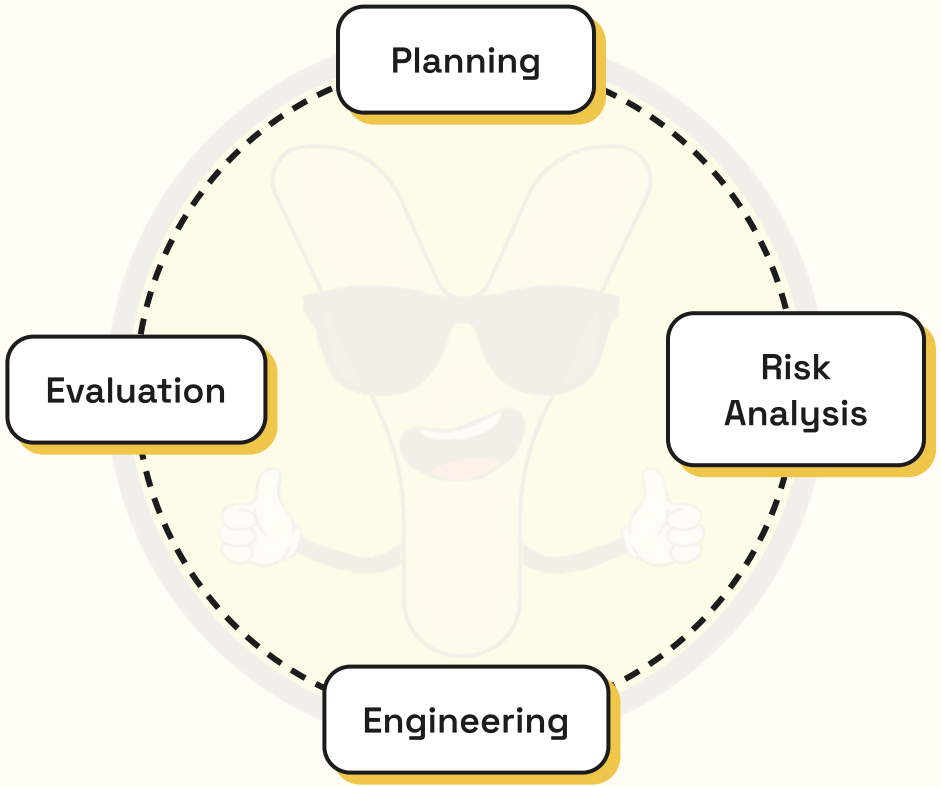
The iterative process of building and refining prototypes based on user feedback.

Prototype Model: Advantages and Disadvantages

Prototyping offers benefits in requirements clarity but also poses challenges.

- Advantages
  - **User Involvement:** High user participation leads to better understanding of requirements.
  - **Early Feedback:** Allows for early detection and correction of errors or misunderstandings.
  - **Reduced Risk:** Reduces the risk of building the wrong product.
  - **Improved User Satisfaction:** Users get a feel for the system early on.
- Disadvantages
  - **Time-Consuming:** Can prolong the development process if not managed well.
  - **Incomplete Requirements:** Users might focus on the prototype's current features, overlooking future needs.
  - **Scope Creep:** Continuous feedback can lead to uncontrolled expansion of requirements.
  - **Discarded Prototype:** The initial prototype might be poorly structured and discarded, leading to wasted effort.

Spiral Model: Introduction and Diagram



Each loop of the spiral represents a phase, with increasing levels of detail and risk assessment.

Spiral Model: Advantages and Disadvantages

The Spiral Model is robust for large, complex projects but requires significant expertise.

- Advantages
  - **Risk Management:** Strong emphasis on risk analysis and mitigation at each stage.
  - **Flexibility:** Accommodates changes and new requirements effectively.
  - **User Feedback:** Allows for user involvement and feedback throughout the development.
  - **Suitable for Large Projects:** Ideal for large, complex, and high-risk projects.
- Disadvantages
  - **Complexity:** More complex to manage than other models.
  - **Costly:** Can be more expensive due to extensive risk analysis and iterations.
  - **Expertise Required:** Requires considerable expertise in risk assessment and management.
  - **Time-Consuming:** Can be time-consuming, especially for smaller projects.

Evolutionary and Iterative Enhancement Models

These models emphasize developing software in increments, adapting to changes and user feedback.

- Evolutionary Development Models
  - **Concept:** Develop software in small, incremental releases, evolving over time based on user feedback and changing requirements.
  - **Characteristics:** Iterative, flexible, focuses on delivering a working system quickly and then refining it.
  - **Types:** Include Incremental Model, Spiral Model (which is also evolutionary), and Component-Based Development.
- Iterative Enhancement Models
  - **Concept:** Start with a simple, core version of the system and then progressively add functionality and refine it through successive iterations.
  - **Characteristics:** Each iteration builds upon the previous one, incorporating feedback and new requirements. The system grows incrementally.
  - **Focus:** Continuous improvement, refinement, and adaptation, often used when requirements are not fully clear upfront.

Exam Corner

- **Q1** Define Software Engineering. Explain its goals and importance in modern software development.
- **Q2** Discuss the key characteristics of software. How do they differ from physical products?
- **Q3** What was the 'Software Crisis'? Explain its major causes and how Software Engineering emerged as a solution.
- **Q4** Compare and contrast Software Engineering processes with conventional engineering processes, highlighting similarities and differences.
- **Q5** List and explain any five Software Quality Attributes. Why are they important for a successful software product?
- **Q6** What is SDLC? Describe the generic phases of a Software Development Life Cycle.
- **Q7** Explain the Waterfall Model with a neat diagram. Discuss its advantages and disadvantages.
- **Q8** Describe the Prototype Model, including its working mechanism and diagram. What are its benefits and drawbacks?

Unit 1 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 2 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

