Unit-4

Software Requirements and Specifications

Software Requirements

Software requirements define the functional and non-functional attributes of a software system, detailing what the system should accomplish and how it should behave to meet the needs of users, stakeholders, and the organization. They serve as the foundation for the design, development, and testing of software solutions.

Software Requirements Types

Functional Requirements

- Functional requirements describe the specific behaviors, features, and interactions that the software system must perform to meet user needs and business objectives.
- They define the system's inputs, processing logic, and outputs, as well as any constraints or conditions that must be satisfied.
- Functional requirements typically answer questions like "What does the system do?" and "How does the system behave?"
- Example: "The system shall allow users to log in with a username and password."

Software Requirements Types

Non-Functional Requirements

- Non-functional requirements specify the quality attributes, constraints, and characteristics that the software system must possess.
- They include aspects such as performance, reliability, usability, security, scalability, maintainability, and compliance.
- They focus on how the system should behave and include aspects beyond functional capabilities.
- Example: "The system shall respond to user actions within 2 seconds under peak load conditions."

Software Requirements Types

User Requirements

- Represent the needs, goals, and expectations of the system's end-users, stakeholders, or customers.
- Focus on the functionality and features that users desire and how they interact with the system.
- Example: "As a customer, I want to be able to browse products, add them to my shopping cart, and complete the checkout process."

Software Requirements Types

System Requirements

- Specify the technical capabilities, infrastructure, and environmental factors needed to support the software system.
- Include hardware specifications, software dependencies, compatibility requirements, and deployment constraints.
- Example: "The software shall be compatible with Windows 10 and macOS 11 operating systems and require a minimum of 4GB RAM."

Software Requirements Types

Interface Requirements

- Interface requirements specify how the software system interacts with external systems, devices, or users.
- They include user interface (UI) requirements, application programming interfaces (APIs), data exchange formats, and communication protocols.
- Example: "The system shall provide a RESTful API for integration with third-party payment gateways."

Software Requirements Types

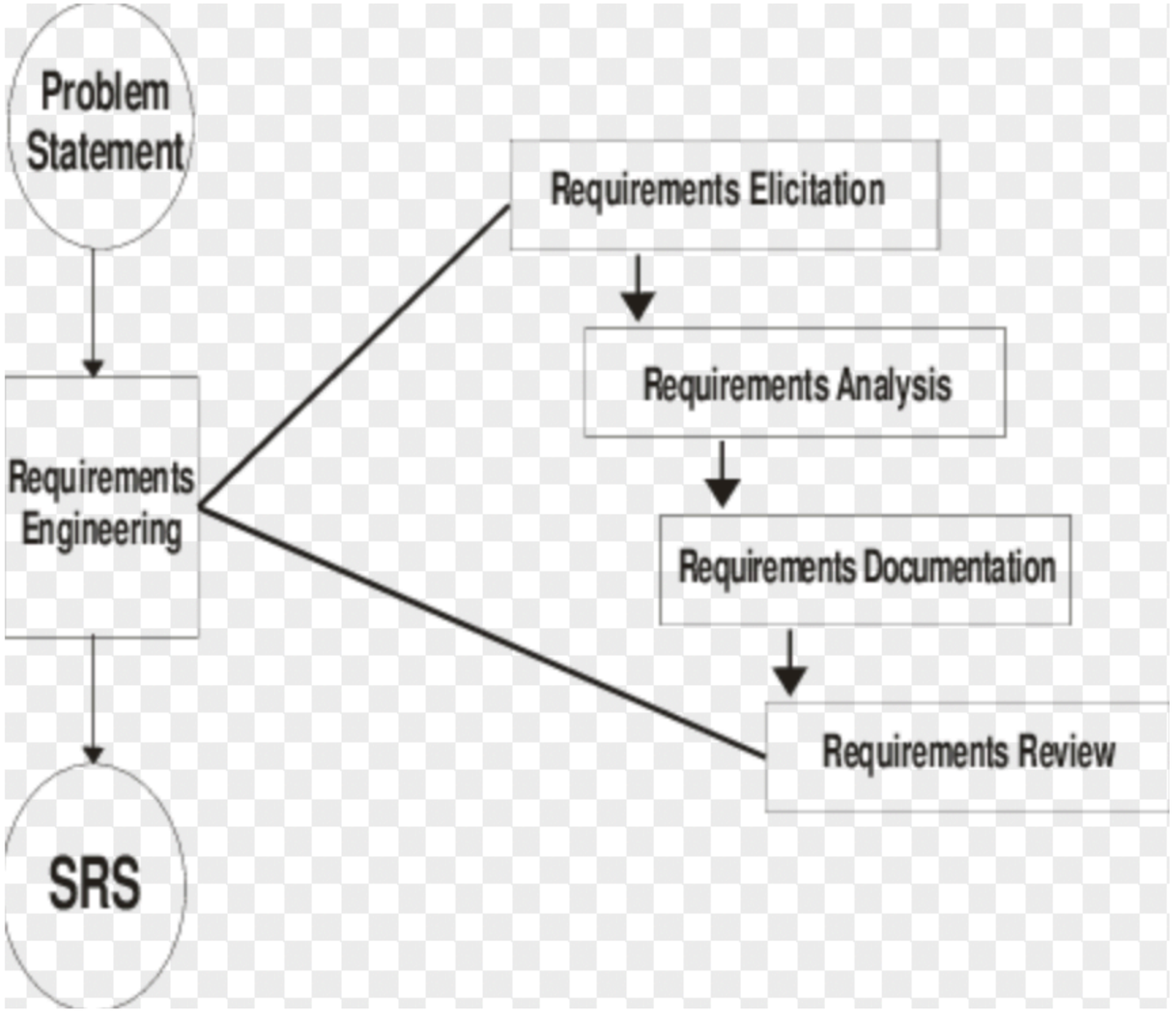
Security Requirements

- Security requirements specify the measures and controls needed to protect the software system from unauthorized access, data breaches, and malicious attacks.
- They address aspects such as authentication, authorization, encryption, data privacy, and compliance with security standards.
- Example: "The system shall encrypt sensitive user data using AES-256 encryption algorithm."

Regulatory Requirements

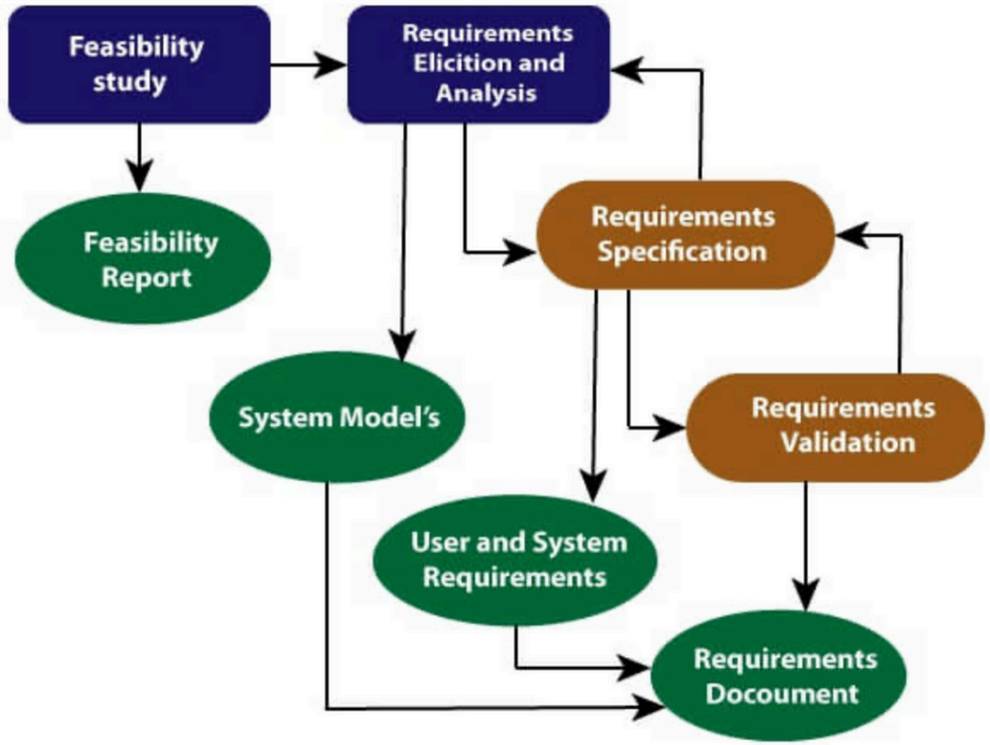
- Regulatory requirements outline the legal and regulatory standards, guidelines, and compliance requirements that the software system must adhere to.
- They include industry-specific regulations, data protection laws, privacy regulations, and accessibility standards.
- Example: "The system shall comply with GDPR (General Data Protection Regulation) requirements for handling personal data."

Requirement Engineering Process

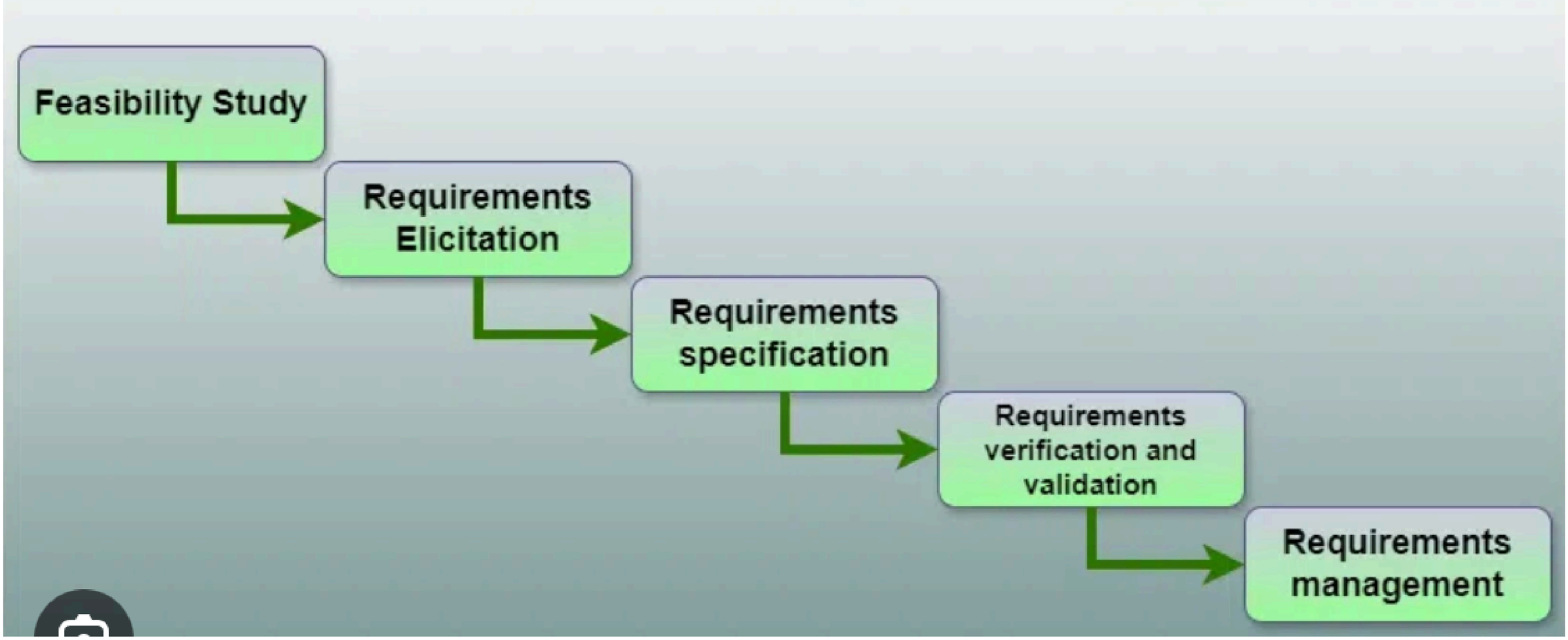


Requirement Engineering Process

- The software requirement engineering process is a systematic and iterative approach to gathering, analyzing, documenting, validating, and managing software requirements throughout the software development lifecycle.
- It involves understanding stakeholder needs, translating them into detailed specifications, and ensuring that the resulting software system meets user



Requirement Engineering Process



Requirement Engineering Process

Feasibility Studies

- Feasibility studies aim to assess the viability of a software project before committing resources. They evaluate technical, economic, and organizational aspects to determine if the project is worth pursuing.
  - **Technical Feasibility:** Assess the availability of technology, resources, and expertise needed to develop the software system. Consider factors such as hardware, software, and development tools.
  - **Economic Feasibility:** Evaluate the cost-effectiveness of the project by analyzing development costs, potential revenue, return on investment (ROI), and cost-benefit analysis. Consider factors such as development, maintenance, and operational costs.
  - **Organizational Feasibility:** Examine the alignment of the project with organizational goals, policies, and resources. Assess factors such as organizational culture, stakeholder commitment, and support.
- Feasibility study report summarizing findings, recommendations, and conclusions regarding the project's viability.

Requirement Engineering Process

Requirements Elicitation and Analysis

- Requirements elicitation and analysis involve understanding stakeholder needs and transforming them into detailed and unambiguous requirements for the software system.
  - **Stakeholder Identification:** Identify and engage all relevant stakeholders, including end-users, customers, domain experts, and project sponsors.
  - **Elicitation Techniques:** Employ various techniques such as interviews, surveys, workshops, brainstorming sessions, and observation to gather requirements from stakeholders.
  - **Requirement Analysis:** Analyze gathered requirements to identify inconsistencies, conflicts, ambiguities, and missing information. Prioritize requirements based on importance, feasibility, and urgency.
  - **Modeling Techniques:** Use modeling techniques such as use case diagrams, activity diagrams, and data flow diagrams to represent system requirements visually.
- Requirement specification documents detailing functional and non-functional requirements, use cases, user stories, and system models are obtained after this process.

Requirement Engineering Process

Requirements Validation

- Requirements validation ensures that the identified requirements accurately reflect stakeholder needs and are feasible and acceptable for implementation.
  - **Verification and Validation:** Verification ensures that requirements are correctly stated, complete, and consistent, while validation confirms that the requirements meet stakeholder needs and objectives.
  - **Review Techniques:** Conduct reviews with stakeholders, domain experts, and development team members to verify the correctness, completeness, and consistency of requirements.
  - **Prototyping:** Build prototypes or mockups to demonstrate key functionalities and gather feedback from stakeholders. Use prototypes to validate requirements and refine them iteratively.
  - **Testing Techniques:** Perform requirement validation tests, such as requirements traceability testing, usability testing, and acceptance testing, to confirm that the system will meet user needs and expectations.
- Validation reports documenting the results of requirement validation activities, including any identified issues, concerns, or changes.

Requirement Engineering Process

Requirement Management

- Requirement management involves managing changes to requirements and ensuring that they remain consistent, traceable, and aligned with project objectives throughout the software development lifecycle.
  - **Change Control Process:** Establish a formal change control process to manage requirement changes, including documentation, review, and approval. Ensure that changes are captured, evaluated, and implemented systematically.
  - **Traceability Management:** Maintain traceability between requirements and other project artifacts, such as design documents, test cases, and code. Establish bi-directional traceability to track changes and dependencies.
  - **Communication and Collaboration:** Facilitate communication and collaboration among stakeholders to ensure that their feedback and concerns are addressed effectively. Use collaboration tools and techniques to engage stakeholders in the requirement management process.
  - **Version Control:** Implement version control mechanisms to manage changes to requirement documents and ensure that stakeholders have access to the latest versions. Track revisions, updates, and comments to maintain an audit trail.
- Requirement management include Requirement traceability matrix, change control logs, updated requirement documentation reflecting approved changes, and updates.

Requirement Engineering Process

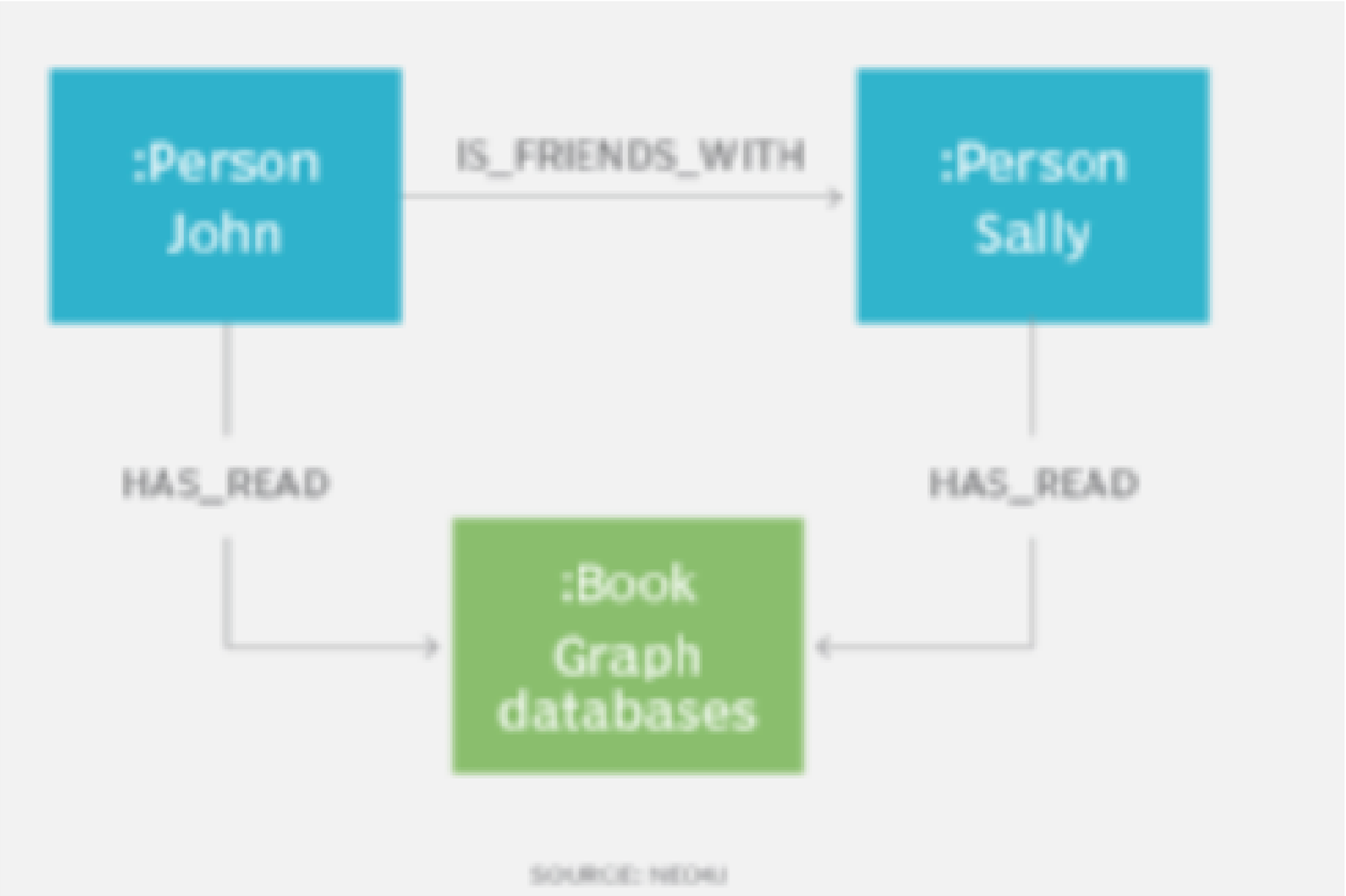
Requirement Management

- Requirement management involves managing changes to requirements and ensuring that they remain consistent, traceable, and aligned with project objectives throughout the software development lifecycle.
  - **Change Control Process:** Establish a formal change control process to manage requirement changes, including documentation, review, and approval. Ensure that changes are captured, evaluated, and implemented systematically.
  - **Traceability Management:** Maintain traceability between requirements and other project artifacts, such as design documents, test cases, and code. Establish bi-directional traceability to track changes and dependencies.
  - **Communication and Collaboration:** Facilitate communication and collaboration among stakeholders to ensure that their feedback and concerns are addressed effectively. Use collaboration tools and techniques to engage stakeholders in the requirement management process.
  - **Version Control:** Implement version control mechanisms to manage changes to requirement documents and ensure that stakeholders have access to the latest versions. Track revisions, updates, and comments to maintain an audit trail.
- Requirement management include Requirement traceability matrix, change control logs, updated requirement documentation reflecting approved changes, and updates.

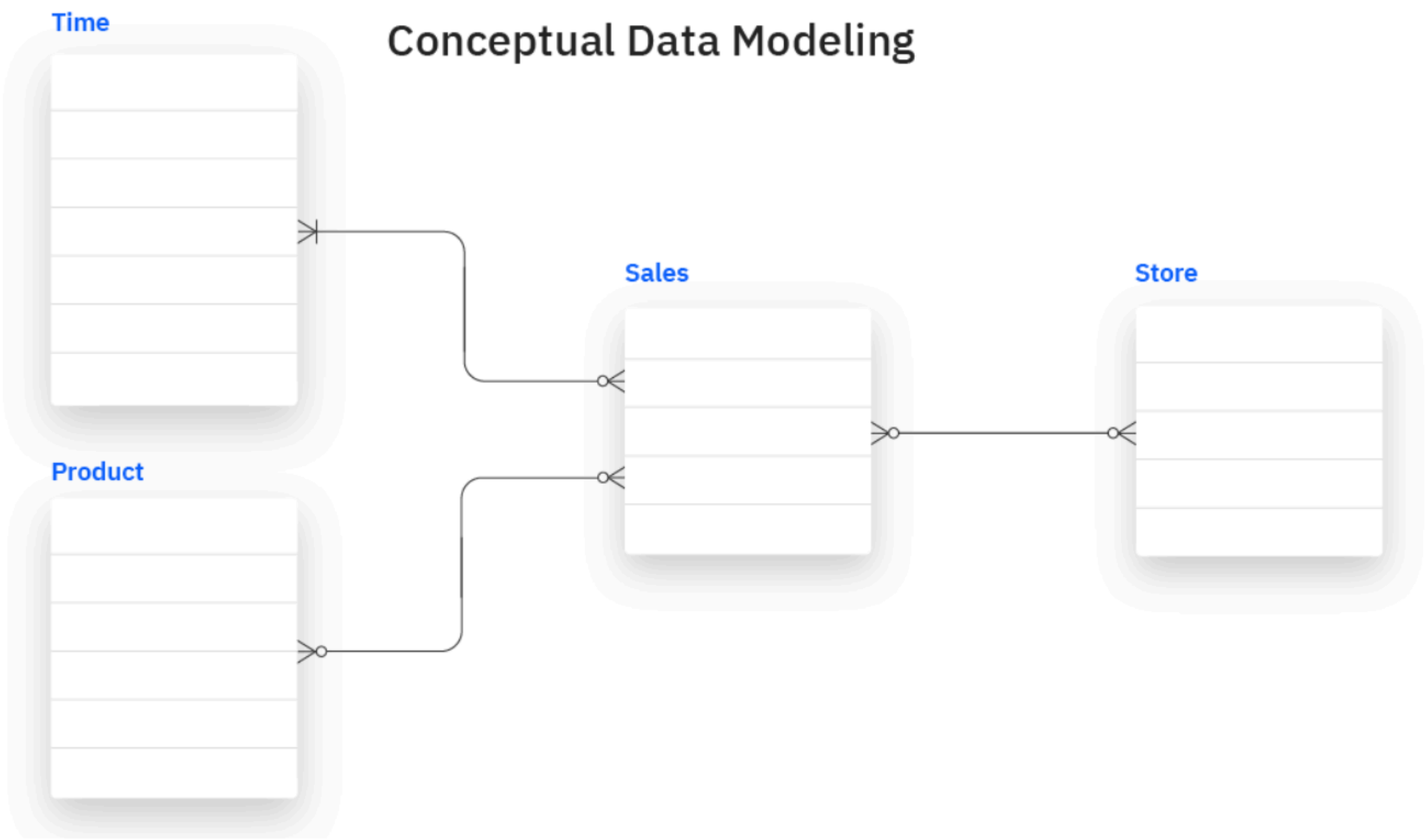
Data Modeling

- Data modeling is the process of creating a visual representation of the data requirements for a software system. It involves defining data entities, their attributes, relationships, and constraints.
- The primary purpose of data modeling is to ensure that the software system stores, processes, and manages data effectively and accurately.
- Components:
    - **Entities:** Represent real-world objects or concepts, such as customers, products, orders, etc.
    - **Attributes:** Describe properties or characteristics of entities, such as name, age, address, etc.
    - **Relationships:** Define associations between entities, indicating how they are connected or related to each other.
    - **Constraints:** Specify rules or conditions that govern the structure and behavior of data, such as uniqueness constraints, cardinality constraints, etc.

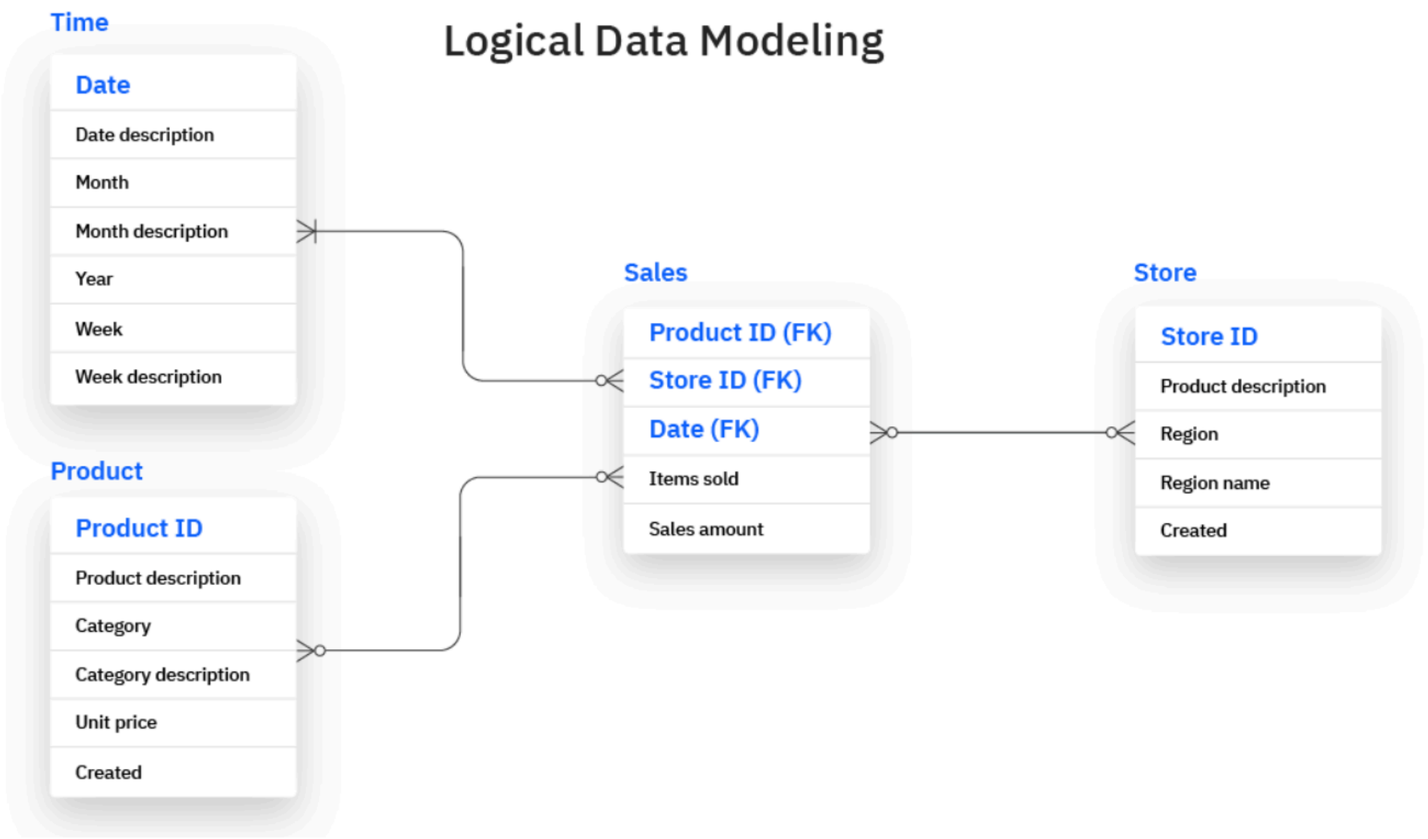
Data Modeling



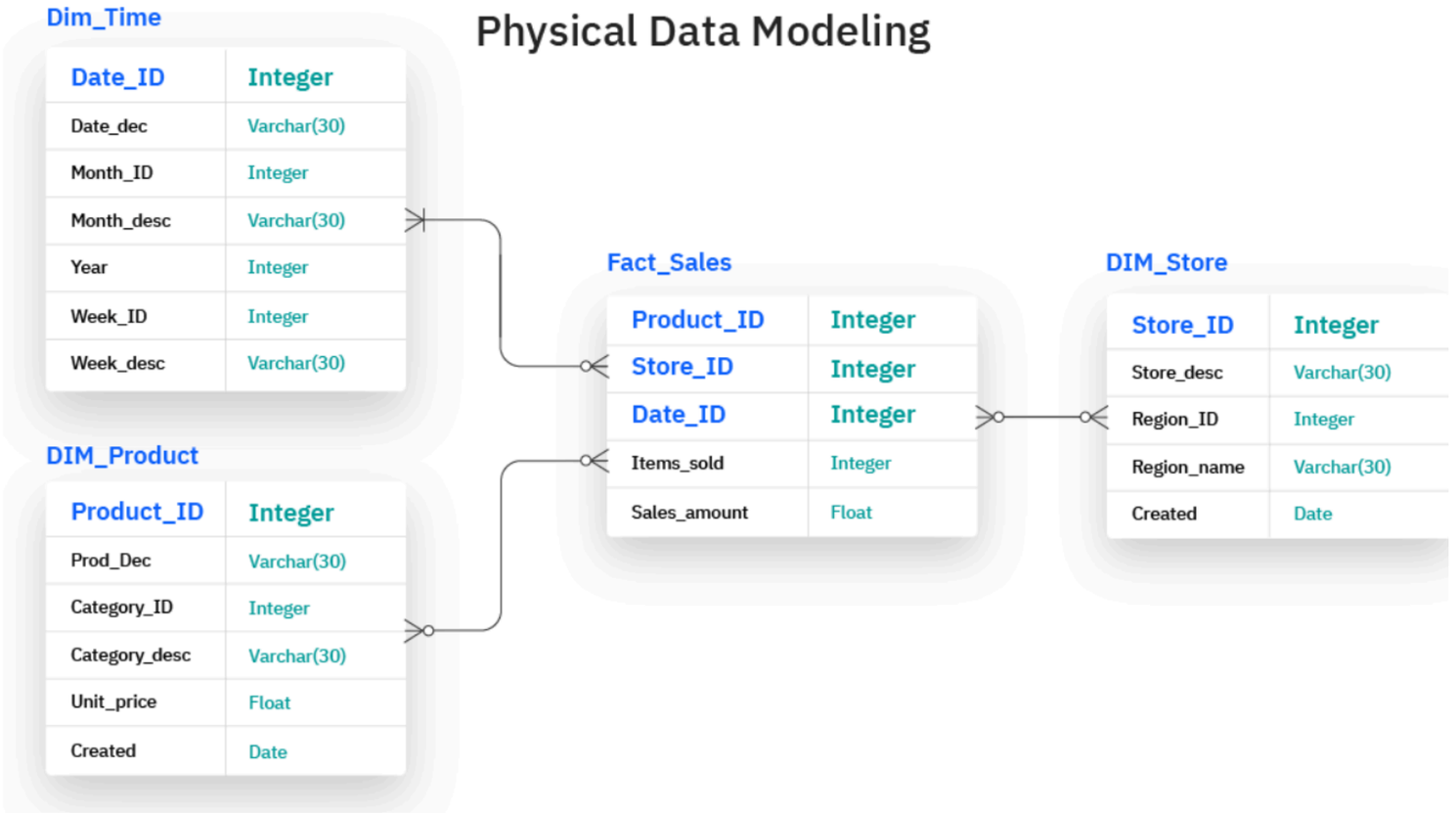
Data Modeling



Data Modeling



Data Modeling



Data Modeling

- **Benefits:**
  - Facilitates communication between stakeholders by providing a common understanding of data requirements.
  - Helps identify and resolve inconsistencies, ambiguities, and redundancies in data structures.
  - Guides database design and implementation by providing a blueprint for creating database schemas and tables.

Flow Diagram

- A flow diagram, also known as a flowchart, is a visual representation of the sequence of steps or processes in a system. It illustrates how data, information, or control flows through different stages of a process.
- Flow diagrams help analyze, design, and document processes, workflows, and algorithms in software systems.
- Components:
  - **Symbols:** Represent different elements of a process, such as activities, decisions, inputs, outputs, and flow control.
  - **Arrows:** Connect symbols to indicate the flow of control or data between process steps.
  - **Start/End Points:** Mark the beginning and end of the process flow.

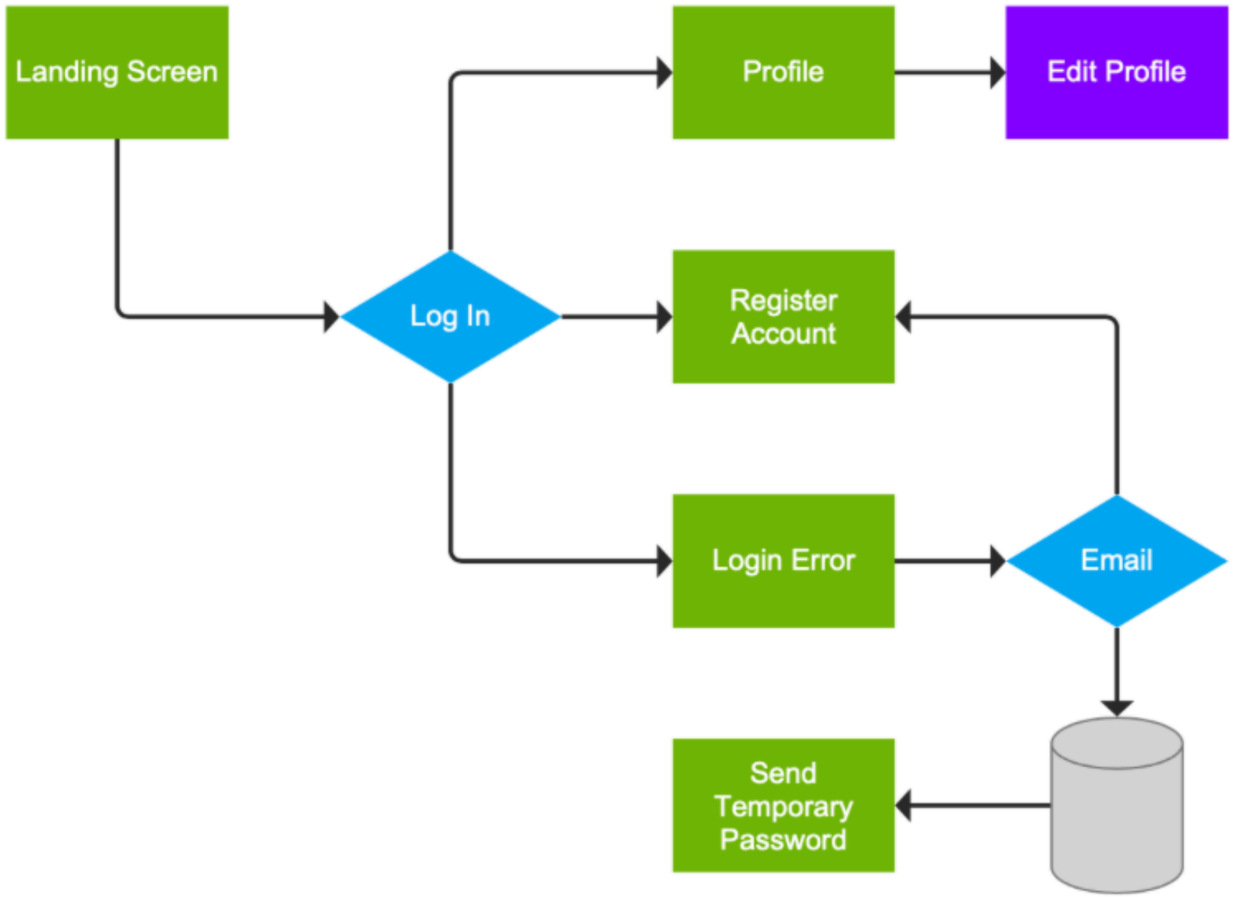
Flow Diagram

- Types:
  - **Functional Flowchart:** Illustrates the sequence of activities or steps involved in performing a specific function or task.
  - **Data Flow Diagram (DFD):** Focuses on the flow of data through the system, showing how data is input, processed, stored, and output.
  - **Business Process Model and Notation (BPMN):** Provides a standardized notation for modeling business processes, including activities, events, gateways, and flows.

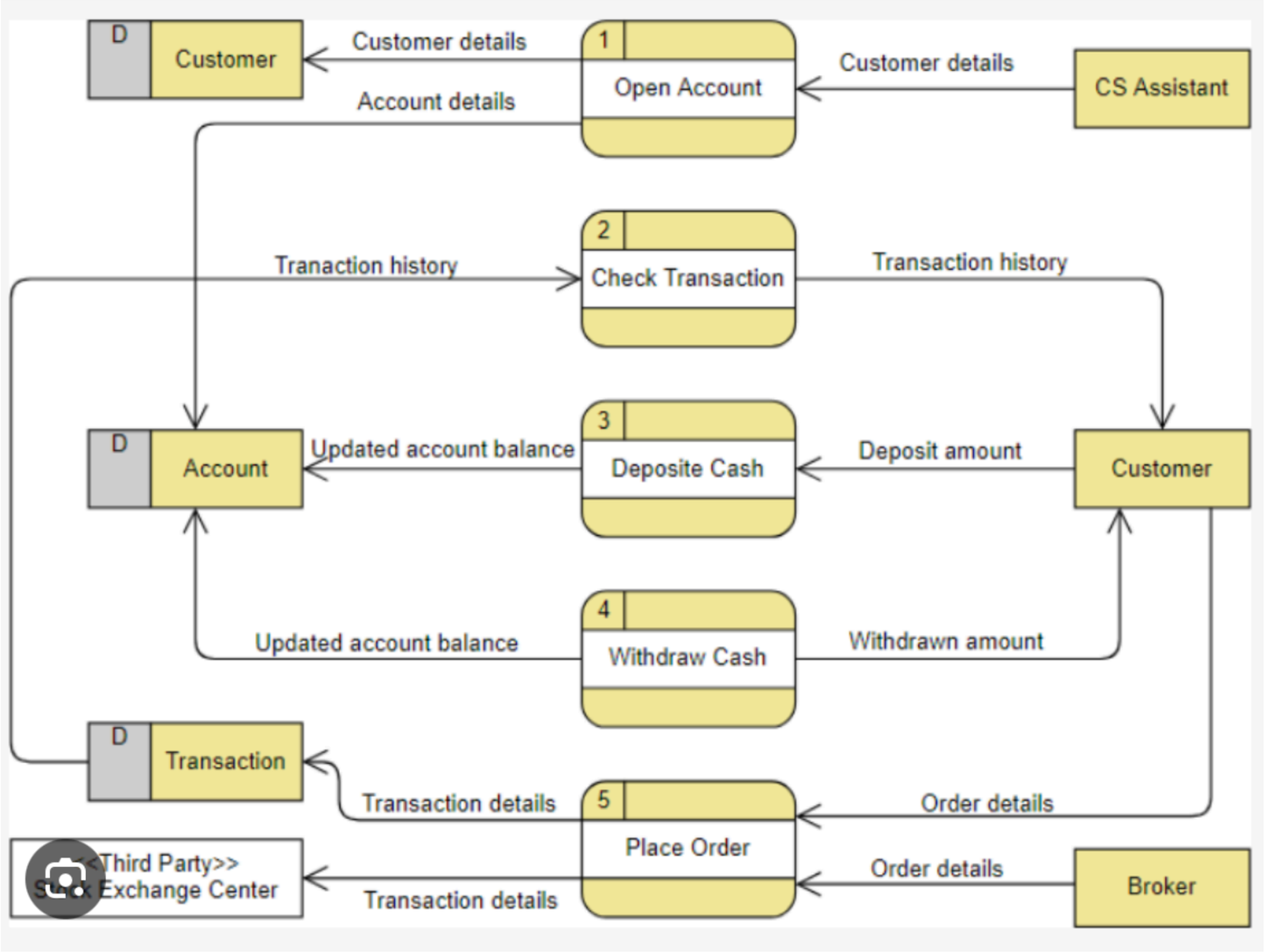
Flow Diagram

- Benefits:
  - Provides a visual representation of complex processes, making them easier to understand and analyze.
  - Facilitates process optimization, identification of bottlenecks, and improvement opportunities.
  - Serves as a communication tool for stakeholders to discuss, review, and refine process workflows.

Flow Diagram



Flow Diagram



Software Prototyping Techniques

- Throwaway or Rapid Prototyping
- Evolutionary Prototyping
- Incremental Prototyping

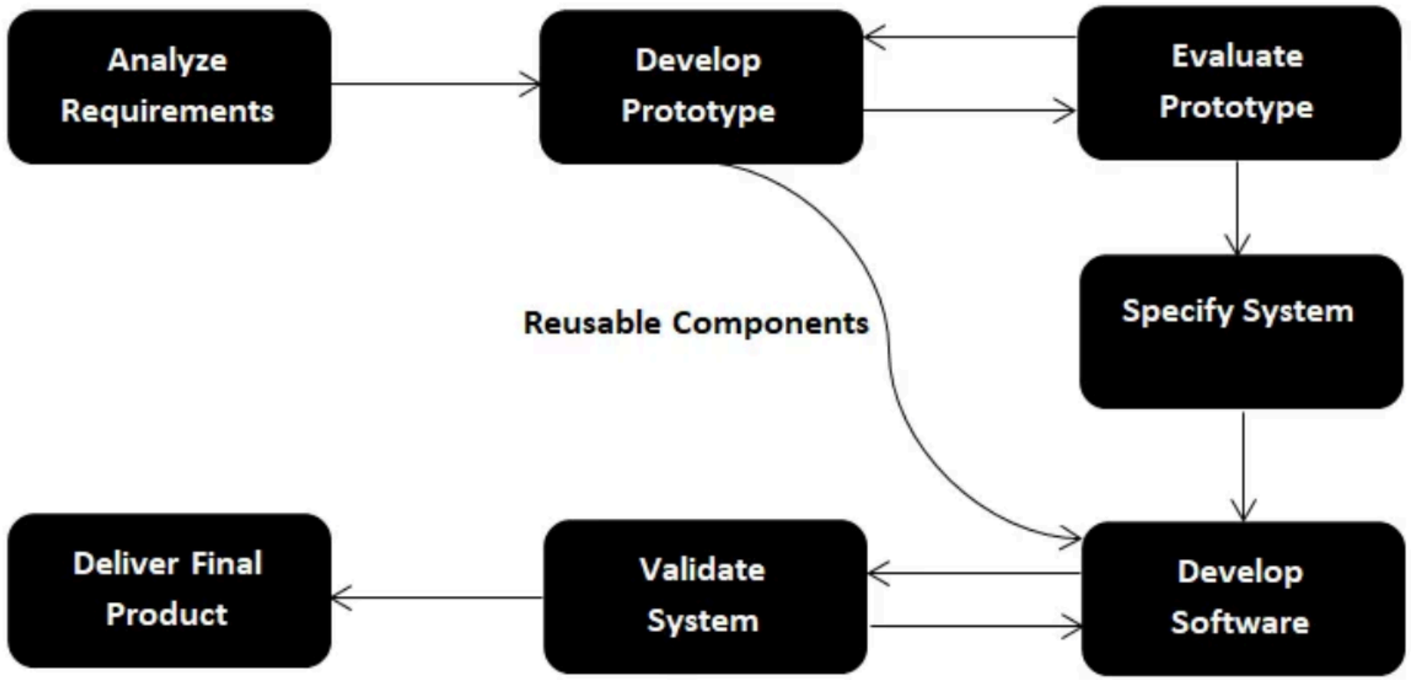
Throwaway or Rapid Prototyping

- Throwaway prototyping, also known as rapid prototyping, involves the creation of a quick, disposable prototype that is used to explore design ideas, validate requirements, and gather feedback from stakeholders.
- The primary purpose of throwaway prototyping is to quickly visualize and test different aspects of the software system, such as user interface designs, workflows, and functionalities, without investing significant time and resources.

Throwaway or Rapid Prototyping

- Process:
  - **Requirements Gathering:** Gather initial requirements from stakeholders through interviews, discussions, or other elicitation techniques.
  - **Prototype Development:** Develop a basic prototype using low-fidelity tools like wireframes, mockups, or interactive prototypes. Focus on demonstrating key features or interactions.
  - **Feedback Collection:** Share the prototype with stakeholders and gather feedback on its usability, functionality, and alignment with requirements.
  - **Iteration or Discard:** Based on the feedback received, iterate on the prototype to incorporate suggested changes and refinements. Alternatively, discard the prototype if it no longer serves its purpose and start over with a new iteration.

Throwaway or Rapid Prototyping



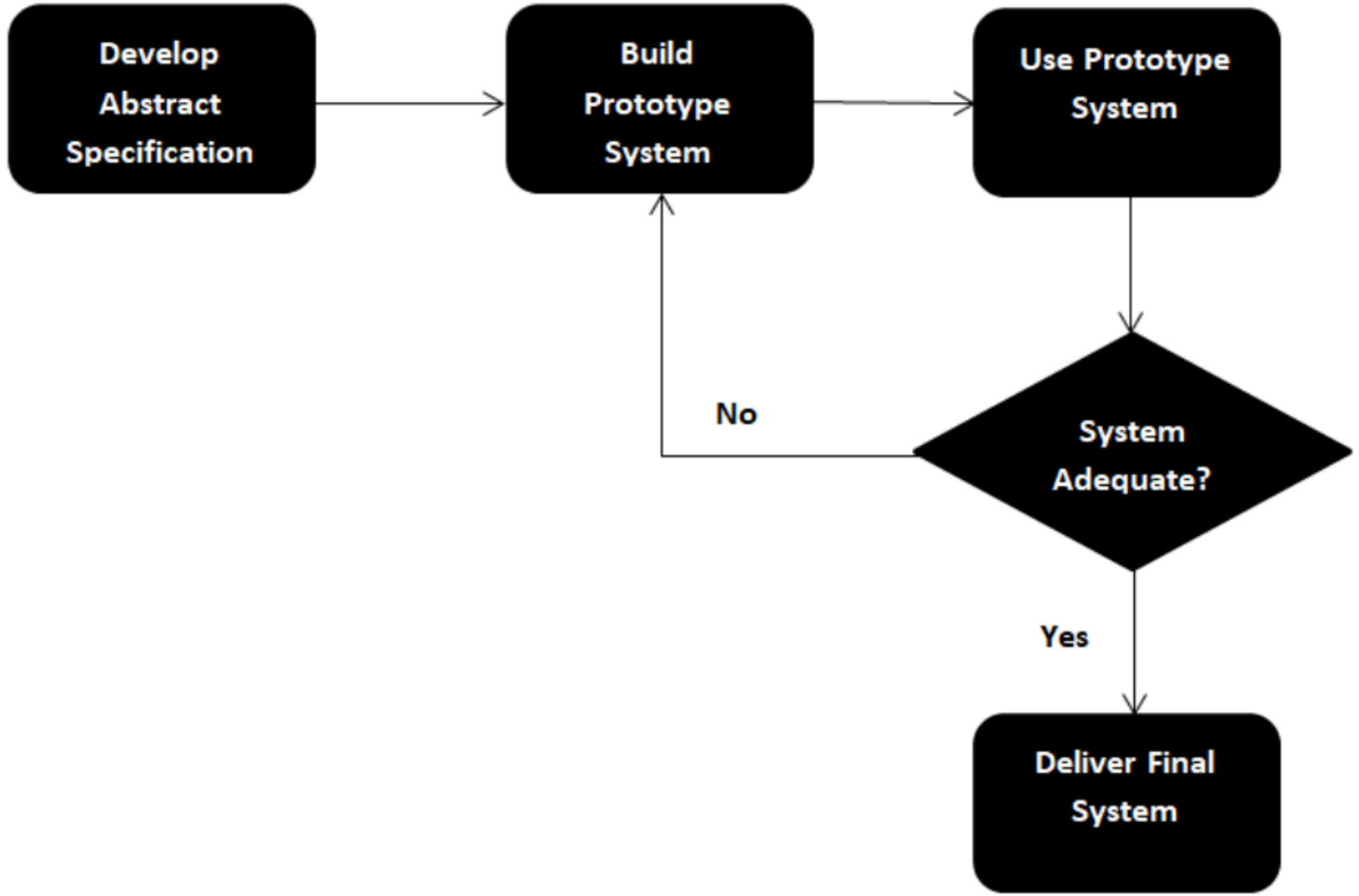
Evolutionary Prototyping

- Evolutionary prototyping involves the development of an initial prototype with basic functionality, which is then refined and enhanced through multiple iterations based on ongoing feedback and testing.
- The main purpose of evolutionary prototyping is to gradually evolve the software system from a basic prototype to a fully functional product, incorporating stakeholder feedback and refining requirements along the way.

Evolutionary Prototyping

- Process:
  - **Initial Prototype:** Develop an initial prototype with core features and functionalities that address the most critical requirements.
  - **Feedback and Iteration:** Share the prototype with stakeholders and gather feedback on its usability, performance, and alignment with requirements. Use this feedback to identify areas for improvement and plan the next iteration.
  - **Incremental Refinement:** Iterate on the prototype, adding new features, refining existing functionalities, and addressing any issues or concerns raised during testing and feedback sessions.
  - **Validation and Finalization:** Continue iterating on the prototype until it meets all requirements and stakeholders' expectations. Validate the final prototype through extensive testing and validation activities before moving to full-scale development.

Evolutionary Prototyping



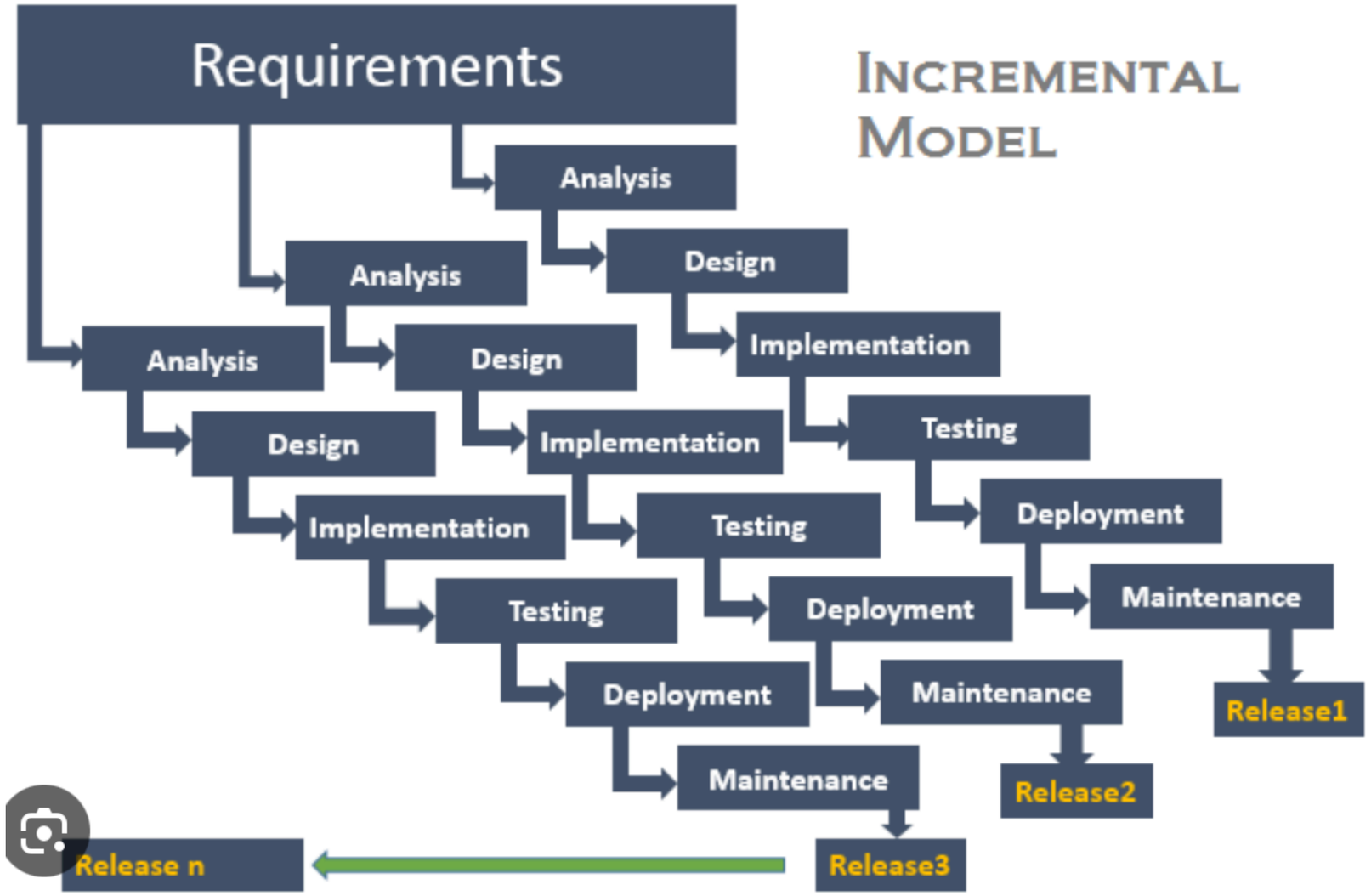
Incremental Prototyping

- Incremental prototyping involves developing and delivering the software system in increments or modules, with each increment adding new functionality and features.
- The primary purpose of incremental prototyping is to deliver working software early and frequently, enabling stakeholders to validate requirements, provide feedback, and make adjustments throughout the development process.

Incremental Prototyping

- Process:
  - **Module Identification:** Divide the software system into manageable increments or modules based on functional or architectural considerations.
  - **Incremental Development:** Develop and deliver each increment sequentially, starting with the most critical or foundational modules and progressively adding new features and functionalities in subsequent increments.
  - **Feedback and Adjustments:** Share each increment with stakeholders and gather feedback on its usability, functionality, and alignment with requirements. Use this feedback to make adjustments and refinements to subsequent increments.
  - **Integration and Testing:** Integrate newly developed increments with existing modules and conduct integration testing to ensure that the entire system functions seamlessly. Repeat the incremental development process until all requirements are met and the software system is ready for deployment.

Incremental Prototyping



PRODUCTIVITY BEGINS TODAY!

The secret of getting ahead is getting started.

- MARK TWAIN