Cavalier Institute - https://cavalierinstitutions.com

Date	XBL-xx-2024	Session No	x

Topic : Software Engineering

Software Engineering Overview:

Software engineering is the systematic application of engineering principles to software development. It aims to deliver high-quality software efficiently. The process typically includes:

- **Planning:** The project is defined, goals are set, and the feasibility is assessed. This phase outlines what the software should achieve.
- **Requirements Gathering:** Detailed requirements are gathered from stakeholders to understand what the software needs to do (both functional and non-functional).
- **Design:** The software's architecture is planned, including how the system will look, function, and be structured. This includes database design and selecting technologies.
- **Development (Implementation):** This is where the actual coding happens. Developers write the software according to the design specifications.
- **Testing:** The software is tested for bugs and errors. It checks whether the software meets the requirements and functions correctly.
- **Deployment:** The software is released and made available for users. It's moved to the live environment.
- **Maintenance:** After deployment, the software is monitored, and updates or fixes are applied as needed to keep it running smoothly.

Each phase is essential for delivering reliable, scalable, and maintainable software.

Some Important Questions

- **What is the importance of having a structured software engineering process?**
  It helps teams work in an organized way, reduces mistakes, and ensures the software is of good quality and delivered on time.

- **How do the different phases of software engineering work together to ensure successful project delivery?**
  Each phase depends on the previous one. Requirements tell us what to build, design shows how, development creates it, testing checks it, and operations make sure it works in the real world.

- **Why is requirements engineering considered a critical first step in the software development lifecycle?**
  It ensures everyone understands what the software should do. Without clear requirements, the project can go off track and fail to meet the user's needs.

- **How can poor design decisions in the early stages impact later phases like development and testing?**
  Bad design makes the code harder to build and test, leading to more bugs, delays, and higher costs.

- **What are some common challenges encountered during the transition from development to deployment?**
  Issues include bugs that weren't caught, differences between testing and live environments, and difficulties with setting up the software in real-world use.

- **What are the key challenges in translating user requirements into technical specifications during the requirements engineering phase?**
  It's hard to understand exactly what users want and then turn that into clear technical details. Misunderstandings can lead to problems later.

- **How do design patterns influence the scalability and maintainability of a software system?**
  Design patterns provide tried-and-true solutions that make the system easier to grow and fix in the future.

- **What role does testing play in ensuring the software meets both functional and non-functional requirements?**
  Testing checks if the software does what it should (functional) and if it performs well (non-functional), like speed and security.

- **Why is it important to involve operations and deployment considerations early in the software design process?**

Early planning for deployment helps avoid problems when moving the software to real-world environments, making sure it's easy to launch and maintain.

- **How can continuous integration and continuous deployment (CI/CD) practices enhance the software development lifecycle?**
  CI/CD helps developers find and fix problems faster by automating testing and deployment, making the process smoother and more reliable.

Assignment - Questions

Level 1

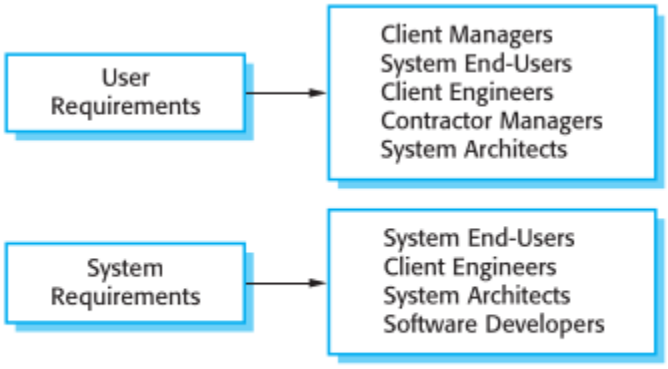
1. Write a for loop to print all even numbers between 1 and 20.
2. Write a for loop to print numbers between 30 and 50 that are divisible by 5.
3. Write a for loop to print the square of each number from 1 to 10.
4. Write a while loop to print all odd numbers between 1 and 15.
5. Write a while loop to print numbers from 100 down to 50 that are divisible by 10.
6. Write a for loop to print all prime numbers between 1 and 50.
7. Write a while loop to print the first 10 numbers of the Fibonacci sequence.
8. Write a for loop to print the multiplication table of 7.
9. Write a while loop to keep doubling a number until it exceeds 1000, starting from 1.
10. Write a for loop to print numbers between 1 and 20, but skip numbers divisible by 3.

Level 2

1. Write a for loop to print all numbers between 1 and 50, but stop the loop when you encounter a number divisible by both 8 and 9.
2. Write a while loop to calculate the sum of all numbers between 1 and 100 that are divisible by 4, and print the result.
3. Write a for loop to print all the digits of a given number in reverse order (e.g., for 1234, print 4 3 2 1).
4. Write a while loop to find the smallest number greater than 500 that is divisible by both 7 and 13.
5. Write a for loop to print the first 10 terms of the arithmetic sequence starting with 5, with a common difference of 3 (i.e., 5, 8, 11, ...).
6. Write a while loop to find the factorial of a given number, and print the result.

Requirements Engineering

Software system requirements are classified as functional or non-functional requirements.



Readers of different types of requirements specification

Functional Requirements

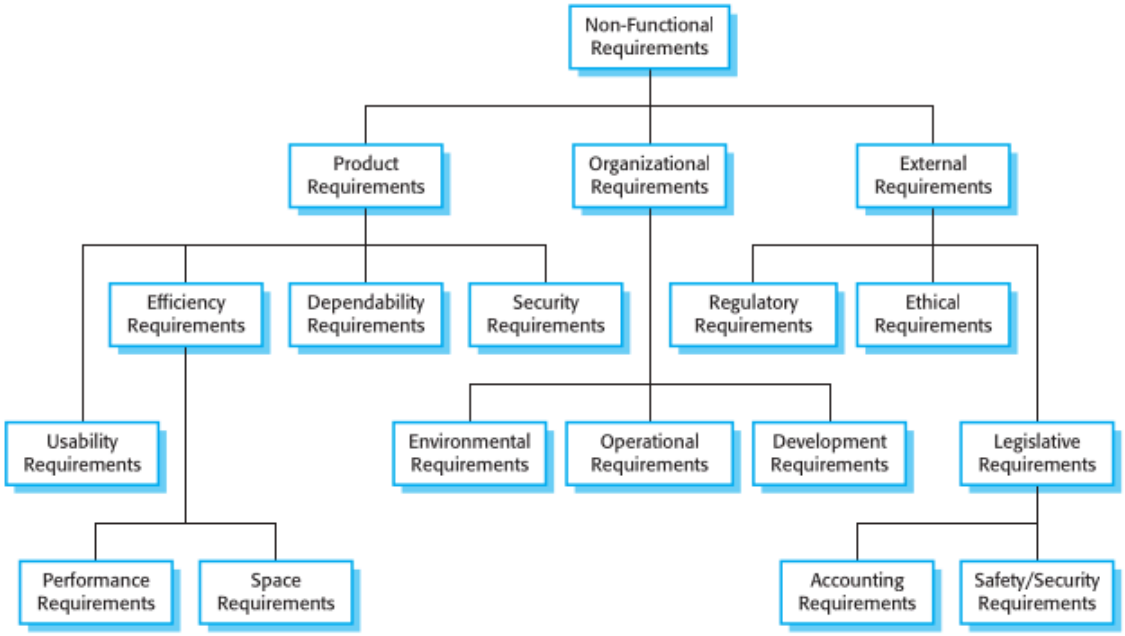
- **Definition**: Functional requirements define what a system should do. They describe the specific behaviors, tasks, and functionalities the system must perform to satisfy the needs of users and stakeholders.
- **Examples**:
    - **Login Functionality**: The system must allow users to log in with a username and password.
    - **Search Feature**: The system should allow users to search for products using keywords.
    - **Order Processing**: The system must process payments and send confirmation emails for completed orders.

Non-Functional Requirements

- **Definition**: Non-functional requirements describe how the system performs its functions. They define system attributes such as performance, security, usability, and reliability.
- **Examples**:
    - **Performance**: The system should load the homepage in under 2 seconds for all users.
    - **Security**: The system must use encryption to protect sensitive user data.
    - **Scalability**: The system should be able to handle up to 10,000 concurrent users without performance degradation.

Functional requirements describe **what** the system should do, while non-functional requirements describe **how well** the system should do it.

Non-Functional Requirements



Requirements Engineering Process

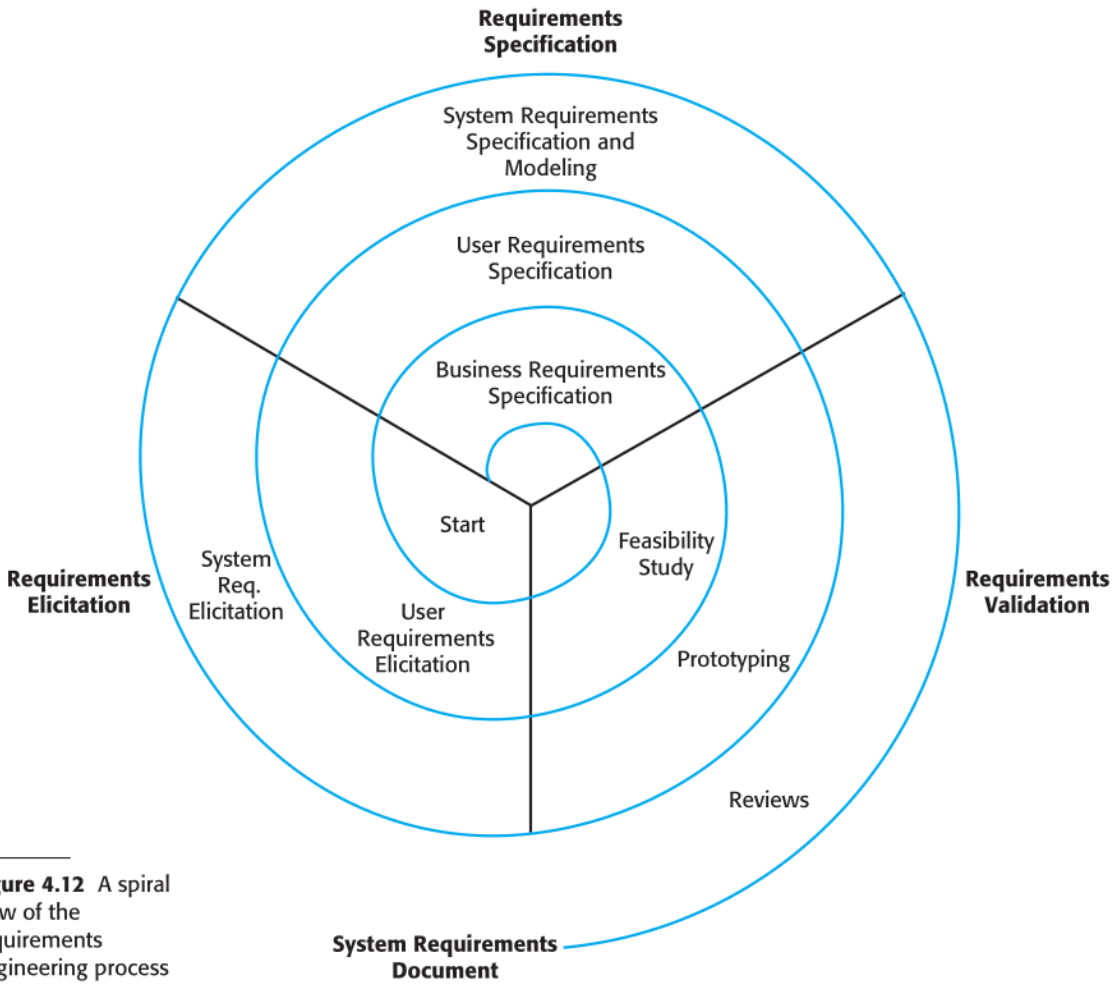


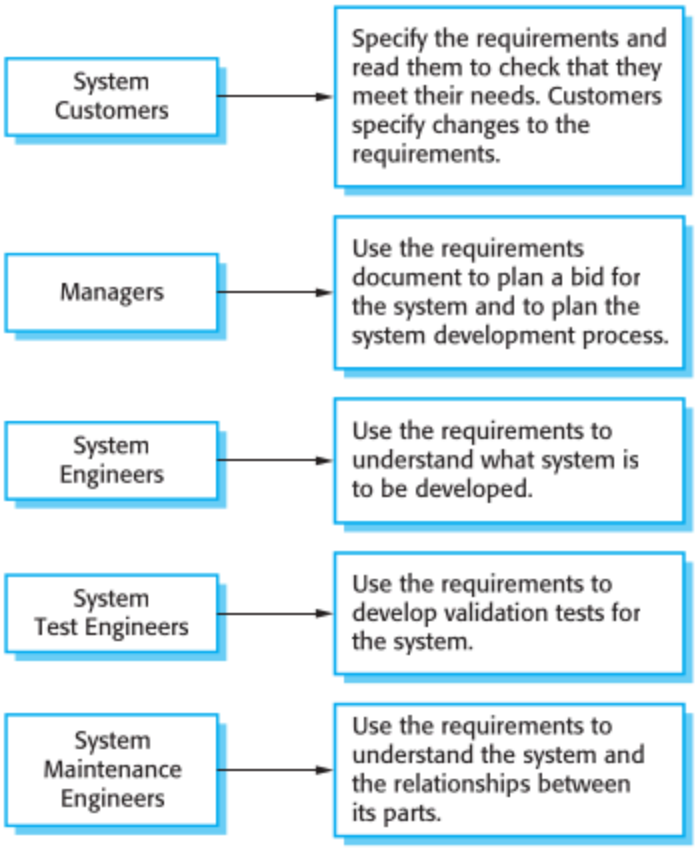
Figure 4.12 A spiral view of the requirements engineering process

Requirements Engineering (RE) is a critical process in software engineering that involves defining, documenting, and maintaining the needs and requirements for a software system. It is one of the foundational activities in the software development lifecycle and ensures that the final product meets the user's expectations, business goals, and technical constraints. Requirements engineering typically encompasses several stages, including elicitation, specification, validation, and management. Below is a detailed breakdown of these stages:

Requirements Document Structure

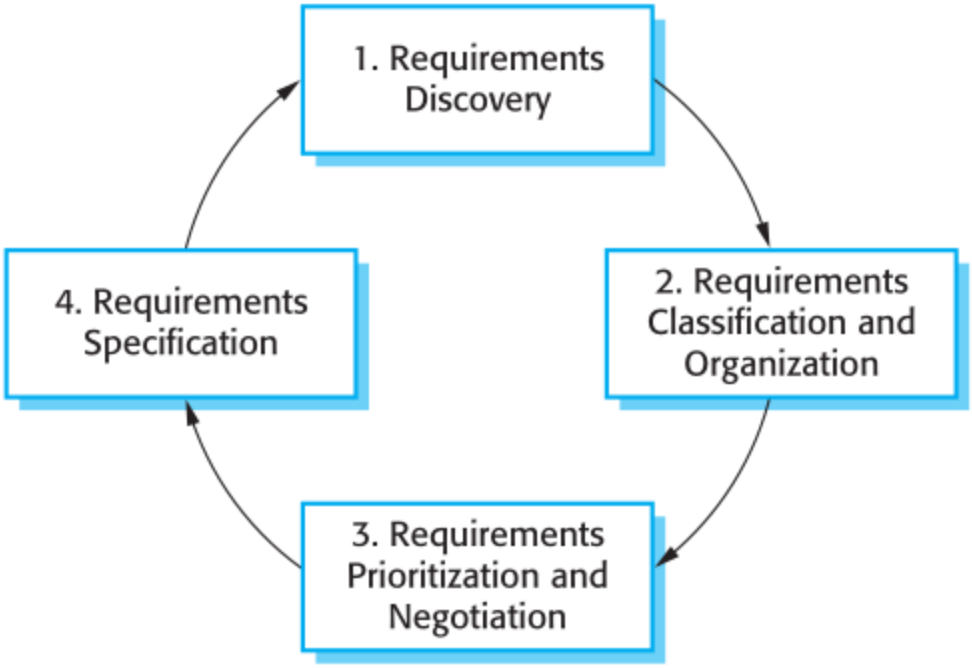
Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The non-functional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components, the system, and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Users of Requirements Document



1. Requirements Elicitation

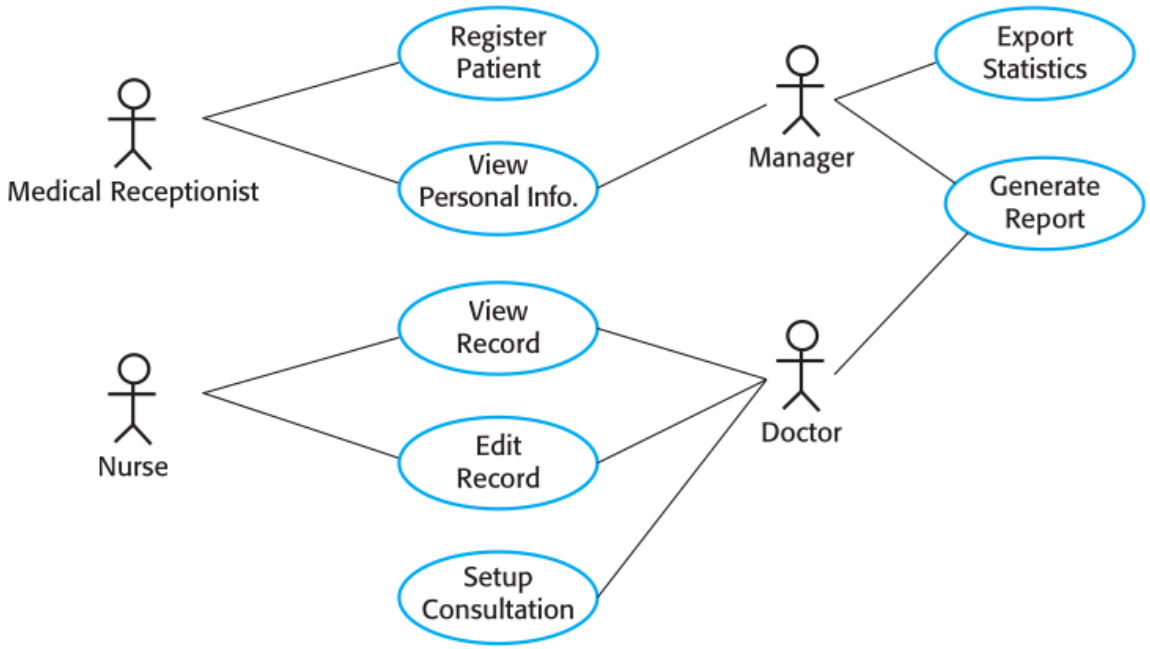
- **Purpose**: Gathering information from stakeholders to understand their needs and expectations from the software.
- **Methods**:
  - **Interviews**: Directly questioning stakeholders to understand their needs.
  - **Surveys and Questionnaires**: Collecting information from a broader group of stakeholders in a structured manner.
  - **Workshops and Brainstorming**: Group activities to encourage discussion and clarify requirements.
  - **Observation**: Watching how users interact with current systems to discover implicit requirements.
  - **Prototyping**: Developing a working model to gather feedback and refine requirements.
- **Challenges**:
  - Communicating with non-technical stakeholders.
  - Identifying conflicting requirements from different stakeholders.
  - Discovering implicit (unstated) requirements.


1. Requirements Discovery
2. Requirements Classification and Organization
3. Requirements Prioritization and Negotiation
4. Requirements Specification

2. Requirements Specification

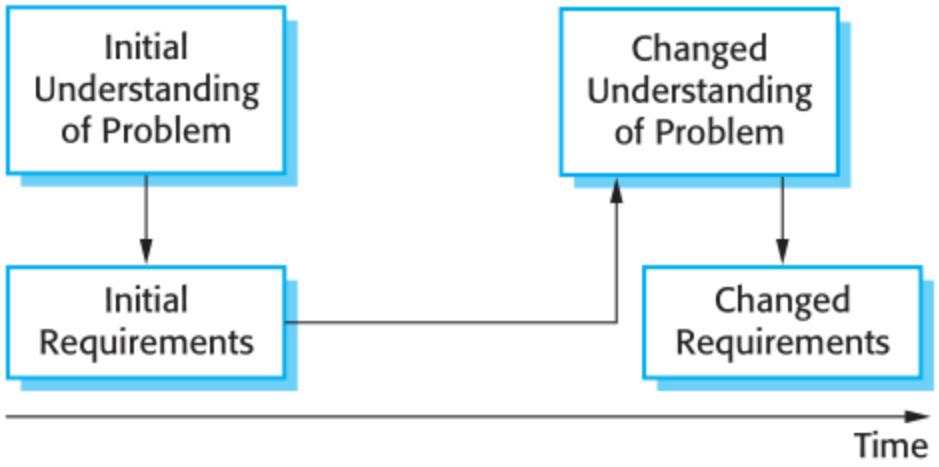
Notation	Description
Natural language sentences	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract.

- **Purpose**: Converting gathered requirements into a formal document that can be used throughout the software development lifecycle.
- **Key Documents**:
    - **Software Requirements Specification (SRS)**: A detailed document outlining functional and non-functional requirements.
    - **Use Case Diagrams**: Visual representations of interactions between users and the system.
    - **User Stories**: Simple descriptions of a feature from the user's perspective (common in Agile environments).
    - **Process Models**: Diagrams like flowcharts or activity diagrams to depict the steps in a business process.
- **Types of Requirements**:
    - **Functional Requirements**: What the system should do (e.g., login, data processing).
    - **Non-Functional Requirements**: How the system should behave (e.g., performance, security, scalability).



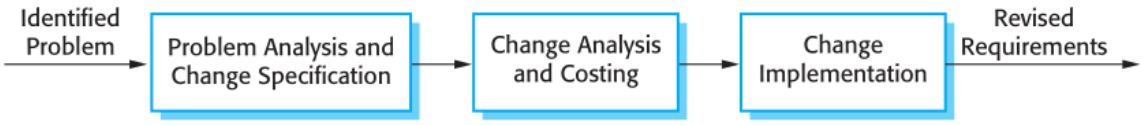
3. Requirements Validation and Verification

- **Purpose**: Ensuring that the requirements accurately reflect the needs of stakeholders and are feasible to implement.
- **Activities**:
  - **Reviews**: Conducting stakeholder and team reviews to ensure completeness and correctness.
  - **Prototyping**: Building mock-ups or simple implementations to validate key requirements.
  - **Test Cases**: Creating test cases based on requirements to verify that they can be validated later.
- **Challenges**:
  - Incomplete or ambiguous requirements leading to misunderstandings.
  - Detecting inconsistencies and conflicts in the early stages.



4. Requirements Management

- **Purpose**: Handling the evolution of requirements throughout the software development lifecycle.
- **Activities**:
  - **Version Control**: Tracking changes made to the requirements.
  - **Traceability**: Ensuring that each requirement can be traced back to its origin and is linked to corresponding design, code, and test cases.
  - **Change Management**: Defining processes for handling changes in requirements due to evolving business needs or market conditions.
- **Challenges**:
  - Handling changing requirements in Agile and dynamic environments.
  - Keeping documentation updated as the software evolves.
  - Maintaining alignment between stakeholder expectations and actual system capabilities.



Traceability Matrix

A traceability matrix helps in mapping the relationship between different types of requirements (such as user requirements, system requirements, design, and test cases) to ensure that each

requirement is adequately covered and validated throughout the software development process. Below is a basic example of a traceability matrix:

Traceability Matrix Example

Requirement ID	Requirement Description	Design/Module	Test Case ID	Status
FR-01	The system should send an SMS reminder 24 hours before the appointment.	Notification Module	TC-01	Implemented
FR-02	Users should be able to search for the weather in different cities.	Weather Search Module	TC-02	In Progress
FR-03	The system should allow users to adjust video resolution based on internet speed.	Adaptive Streaming Module	TC-03	Implemented
FR-04	The system should maintain a log of employee check-ins and check-outs.	Attendance Tracking Module	TC-04	Pending
FR-05	The system must send an email notification when a grade is posted for students.	Notification System	TC-05	Implemented
NFR-01	The system must handle up to 10,000 simultaneous users during peak hours.	Load Balancing	TC-06 (Load Test)	In Progress
NFR-02	The system must be available 24/7 with an uptime of 99.9%.	High Availability Architecture	TC-07	Pending
NFR-03	The system should support multiple languages including English, Spanish, and French.	Internationalization Module	TC-08	Implemented
FR-06	The system should detect the user's location using GPS when requesting a ride.	GPS Integration	TC-09	Implemented
NFR-04	The system should encrypt all data in transit and at rest.	Security Module	TC-10 (Security Test)	In Progress
NFR-05	Users must log in using two-factor authentication (2FA).	Authentication Module	TC-11 (Security Test)	Implemented

Explanation of the Columns:

- **Requirement ID:** A unique identifier for each functional (FR) or non-functional (NFR) requirement.
- **Requirement Description:** A brief summary of the requirement.
- **Design/Module:** The specific design component or module that implements the corresponding requirement.
- **Test Case ID:** The ID of the test case that validates the requirement.
- **Status**: The current status of the requirement implementation (e.g., "Implemented", "In Progress", "Pending").

This traceability matrix helps ensure that every requirement has been mapped to a design component and is being validated by a corresponding test case. It ensures traceability from requirements through to testing, which is crucial for verifying completeness and quality assurance.

5. Types of Requirements

- **Business Requirements**: High-level requirements that capture the goals of the organization for the system.
- **User Requirements**: Specifications from the user's perspective that define how the system should support user tasks.
- **System Requirements**: Detailed descriptions of the functionalities that the system must provide, including hardware, software, and network requirements.

6. Importance of Requirements Engineering

- **Minimizing Project Risks**: Well-defined requirements reduce the chances of scope creep, budget overruns, and project delays.
- **Improved Communication**: A thorough requirements document helps bridge the gap between stakeholders (business and technical teams).
- **Higher Quality Software**: Clear requirements lead to better design, implementation, and testing, which results in fewer defects and higher customer satisfaction.
- **Cost Control**: Requirements management helps in controlling cost by identifying changes early in the process and ensuring that they are justified.

7. Tools Used in Requirements Engineering

- **JIRA/Confluence**: Used for tracking requirements, creating user stories, and managing Agile projects.
- **IBM Rational DOORS**: A tool for capturing, tracking, analyzing, and managing changes to requirements.
- **Enterprise Architect**: Used for creating and managing requirements, especially with UML and modeling approaches.

- **Microsoft Excel/Word**: Simpler tools for documenting requirements in small projects.
- **ReqView**: A tool for managing structured requirements and traceability.

8. Challenges in Requirements Engineering

- **Ambiguity and Incompleteness**: Requirements may be vague or incomplete, leading to misinterpretation.
- **Changing Requirements**: Stakeholders' needs may evolve, causing the requirements to change, which can lead to scope creep.
- **Stakeholder Conflicts**: Different stakeholders may have conflicting priorities or needs, which need to be reconciled.
- **Technical Constraints**: Certain requirements may be impossible or too costly to implement due to technical limitations.

9. Best Practices in Requirements Engineering

- **Involve Stakeholders Early and Often**: Continuous collaboration with stakeholders ensures that requirements are well-understood and agreed upon.
- **Use Models and Visualizations**: Diagrams, flowcharts, and mock-ups help clarify and communicate requirements more effectively.
- **Focus on Clear, Testable Requirements**: Write requirements that can be validated through testing to ensure the end product meets expectations.
- **Iterate and Refine**: Review and refine requirements throughout the development process, especially in Agile environments.
- **Prioritize Requirements**: Not all requirements have the same importance. Use techniques like MoSCoW (Must have, Should have, Could have, Won't have) to prioritize them.

Requirements engineering is essential for creating software that satisfies the needs of users and stakeholders. By carefully eliciting, specifying, validating, and managing requirements, software engineers can reduce project risks, improve communication between teams, and deliver high-quality software products. Proper RE processes ensure that the software aligns with business objectives and provides real value to users.

Requirements Engineering / Questions / Scenarios

Scenario 1
A hospital management system is being developed to handle patient records, doctor

appointments, billing, and insurance claims. The system should provide different access levels for doctors, patients, and administrators.

Question
What type of requirement is "the system should provide different access levels for doctors, patients, and administrators"?

Solution
This is a **functional requirement**. It specifies how the system should behave by implementing role-based access control, ensuring that users can only access the information relevant to their role.

Scenario 2
A mobile application is being designed to allow users to order food from local restaurants. The app should load available restaurants and menus within 2 seconds and process payments securely.

Question
Which type of requirements would describe the "2-second load time" and "secure payment processing"?

Solution
The **2-second load time** is a **non-functional requirement** related to performance.
The **secure payment processing** is also a **non-functional requirement**, but related to security. Non-functional requirements describe how the system should perform or behave under specific conditions.

Scenario 3
A software team is working on an e-commerce website. One of the team members suggests adding a live chat support feature, but the project manager says this is not a part of the current system requirements.

Question
What is the likely cause of this confusion, and how can it be avoided?

Solution
The confusion likely arises from an **undefined or unclear scope** in the requirements specification. This can be avoided by having a well-documented and approved **requirements specification document** that clearly lists all functionalities and features. Any new ideas should go through a formal change management process.

Scenario 4
A banking system is being designed where customers can transfer money between accounts. The system needs to verify the user's identity before allowing any transactions and provide a confirmation message once the transfer is completed.

Question
What are the two primary functional requirements in this scenario?

Solution

1. The system must **verify the user's identity** before allowing transactions.
2. The system must **provide a confirmation message** after a successful money transfer.

These are both functional requirements because they define the specific behavior of the system.

Scenario 5
An online learning platform is being developed, where users can watch video tutorials and take quizzes. One of the non-functional requirements is that the platform should support up to 10,000 concurrent users without performance degradation.

Question
What type of non-functional requirement is this, and why is it important?

Solution
This is a **scalability requirement** (a type of non-functional requirement). It is important because it ensures the system can handle a large number of users simultaneously, which is critical for maintaining performance during peak usage times.

Scenario 6
A project management tool is being developed that allows users to create tasks, assign them to team members, and set deadlines. The system should automatically send email reminders to team members 24 hours before a task deadline.

Question
What type of requirement is the "automatic email reminders 24 hours before a task deadline"?

Solution
This is a **functional requirement** because it defines a specific feature that the system must perform: sending email reminders based on task deadlines.

Scenario 7
A social media platform is under development where users can post photos, comment on posts,

and send direct messages. A requirement states that any user data stored in the system must be encrypted to ensure privacy.

Question
What type of requirement does "data must be encrypted" represent, and why is it necessary?

Solution
This is a **non-functional requirement** related to **security**. Encryption ensures the privacy and confidentiality of user data, protecting it from unauthorized access or breaches.

Scenario 8
A fitness tracker app is being built to record users' daily steps, heart rate, and calories burned. One of the stakeholders requests that the app should have a feature to compare a user's activity data with friends who use the app.

Question
What type of activity is needed to address this new request from the stakeholder?

Solution
This requires a **change in the functional requirements**. The stakeholder's request introduces a new feature (comparing data with friends), which wasn't part of the original requirements. A formal **change request process** should be followed to assess the impact of adding this feature and update the requirements accordingly.

Scenario 9
A travel booking website is being developed where users can search for flights, hotels, and car rentals. One of the requirements specifies that the system must process search queries and return results within 3 seconds, regardless of the number of users.

Question
What type of requirement does "return results within 3 seconds" represent, and why is it important?

Solution
This is a **non-functional requirement** related to **performance**. It is crucial because users expect quick responses from search queries, especially when multiple users are accessing the system simultaneously. Performance-related requirements directly impact the user experience.

Scenario 10
A library management system is being developed to track borrowed books, late fees, and

inventory. A requirement states that only librarians can update the inventory and modify book records, while regular users can only borrow and return books.

Question
What is this type of requirement called, and why is it critical to the system's function?

Solution
This is a **functional requirement** involving **role-based access control**. It is critical because it ensures that users have permissions appropriate to their roles, which maintains the integrity of the library's inventory data and ensures only authorized personnel can make changes.

Scenario 11
A healthcare application is being developed to allow patients to book appointments with doctors. A requirement states that patients should receive an SMS reminder 24 hours before their appointment.

Question
What type of requirement is the "SMS reminder 24 hours before the appointment," and how does it impact the system?

Solution
This is a **functional requirement**. It defines the specific action that the system must perform — sending SMS reminders. This impacts the system by requiring integration with an SMS service and scheduling functionality.

Scenario 12
A weather forecasting website allows users to search for the weather in different cities. A non-functional requirement specifies that the website must support different languages, including English, Spanish, and French.

Question
What type of non-functional requirement is described here, and why is it important?

Solution
This is a **usability requirement** related to **internationalization**. Supporting multiple languages ensures that users from different regions can access the service in their preferred language, improving the platform's accessibility.

Scenario 13
A video streaming platform is being developed. The system should allow users to watch videos in different resolutions (480p, 720p, 1080p, and 4K) based on their internet speed.

Question
What type of requirement is this, and what would be a potential challenge in implementing it?

Solution
This is a **functional requirement**. It specifies that the system must offer different video resolutions. A potential challenge is ensuring the system can dynamically adjust video quality based on varying internet speeds, requiring efficient bandwidth detection and adaptive streaming.

Scenario 14
A company wants to implement an employee attendance tracking system. The system must maintain a log of employee check-ins and check-outs, and generate reports at the end of each month.

Question
What is the primary functional requirement for this system?

Solution
The primary **functional requirement** is that the system should track employee check-ins and check-outs. It must store this information in a log, which can then be used to generate reports.

Scenario 15
An e-learning platform allows students to submit assignments and teachers to grade them. A requirement states that the system should notify the student via email when a grade is posted.

Question
What type of requirement is this, and why is it necessary?

Solution
This is a **functional requirement**. It is necessary because students need timely feedback on their assignments, and notifications ensure they are aware when a grade is posted, enhancing communication and usability.

Scenario 16
A retail company is building an online store where users can browse products, add items to a cart, and complete purchases. One of the requirements is that the system must be able to handle up to 10,000 simultaneous users during peak sales events.

Question
What type of non-functional requirement is this, and what challenge might it present?

Solution
This is a **scalability requirement** (non-functional). The challenge here is ensuring that the system can handle high traffic volumes without performance degradation, requiring load balancing, efficient database management, and server optimization.

Scenario 17
A ride-sharing app is under development where users can request rides, and drivers can accept ride requests. A requirement specifies that the app should use GPS to automatically detect the user's current location when they request a ride.

Question
What type of requirement is this, and what system component would be involved?

Solution
This is a **functional requirement** that involves integrating with a GPS system. The app must accurately detect the user's location and pass it on to the server to match them with nearby drivers.

Scenario 18
A university is developing a portal for students to check their grades and download transcripts. A requirement specifies that the system must be available 24/7, with an expected uptime of 99.9%.

Question
What type of non-functional requirement is this, and how could it be implemented?

Solution
This is a **reliability requirement**. It can be implemented through server redundancy, regular backups, and monitoring tools to ensure the system remains available, even during high traffic or server failures.

Scenario 19
A banking application allows customers to transfer funds between accounts and pay bills. One of the requirements states that users must log in using two-factor authentication (2FA).

Question
What type of requirement is two-factor authentication, and why is it critical?

Solution
This is a **security requirement** (non-functional). It is critical for ensuring that only authorized

users can access sensitive banking information, providing an extra layer of protection against fraud and unauthorized access.

Scenario 20
A government agency is developing a system to manage citizen data. One of the requirements specifies that all data stored in the system must be encrypted both in transit and at rest.

Question
What type of requirement is data encryption, and why is it necessary?

Solution
This is a **non-functional requirement** related to **security**. Data encryption is necessary to protect sensitive information from being accessed or stolen during transmission or while stored in the system, ensuring compliance with data privacy laws.

Exercise on Requirements Engineering

4.1 Identify and briefly describe four types of requirements that may be defined for a computer-based system.

4.2 Discover ambiguities or omissions in the following statement of requirements for part of a ticket-issuing system:

An automated ticket-issuing system sells rail tickets. Users select their destination and input a credit card and a personal identification number. The rail ticket is issued and their credit card account charged. When the user presses the start button, a menu display of potential destinations is activated, along with a message to the user to select a destination. Once a destination has been selected, users are requested to input their credit card. Its validity is checked and the user is then requested to input a personal identifier. When the credit transaction has been validated, the ticket is issued.

4.3 Rewrite the above description using the structured approach described in this chapter. Resolve the identified ambiguities in an appropriate way.

4.4 Write a set of non-functional requirements for the ticket-issuing system, setting out its expected reliability and response time.

4.5 Using the technique suggested here, where natural language descriptions are presented in a standard format, write plausible user requirements for the following functions:

- An unattended petrol (gas) pump system that includes a credit card reader. The customer swipes the card through the reader, then specifies the amount of fuel required. The fuel is delivered, and the customer's account debited.
- The cash-dispensing function in a bank ATM.
- The spelling-check and correcting function in a word processor.

4.6 Suggest how an engineer responsible for drawing up a system requirements specification might keep track of the relationships between functional and non-functional requirements.

4.7 Using your knowledge of how an ATM is used, develop a set of use cases that could serve as a basis for understanding the requirements for an ATM system.

4.8 Who should be involved in a requirements review? Draw a process model showing how a requirements review might be organized.

4.9 When emergency changes have to be made to systems, the system software may have to be modified before changes to the requirements have been approved. Suggest a model of a process for making these modifications that will ensure that the requirements document and the system implementation do not become inconsistent.

4.10 You have taken a job with a software user who has contracted your previous employer to develop a system for them. You discover that your company's interpretation of the requirements is different from the interpretation taken by your previous employer. Discuss what you should do in such a situation. You know that the costs to your current employer will increase if the ambiguities are not resolved. However, you also have a responsibility of confidentiality to your previous employer.

4.1. Identify and briefly describe four types of requirements that may be defined for a computer-based system.

Scenario:

When defining requirements for a computer-based system, various types of requirements may need to be identified and described to ensure a clear understanding of what the system must accomplish. These include functional requirements, non-functional requirements, domain requirements, and user requirements.

Solutions:

1. **Functional Requirements**: These specify the specific behaviors or functions the system must support. They describe what the system should do, such as generating reports, calculating totals, or processing transactions.
    - *Example*: The system should allow users to select a rail destination and issue tickets.

2. **Non-Functional Requirements**: These specify criteria that judge the system's operation, rather than specific behaviors. Non-functional requirements focus on qualities such as performance, security, usability, and reliability.
    - *Example*: The system must process a transaction in under 2 seconds.
3. **Domain Requirements**: These originate from the system's operating environment and may be specific to the domain in which the system will operate. They might impose specific constraints or rules.
    - *Example*: In a rail ticketing system, the system should follow the rail network's pricing rules and regulations.
4. **User Requirements**: These are statements in natural language that express what the system must provide to fulfill the user's goals. They are usually written from the user's perspective.
    - *Example*: The user must be able to select a destination from a displayed list of options.

4.2. Discover ambiguities or omissions in the following statement of requirements for part of a ticket-issuing system:

Scenario:

The system description mentions an automated ticket-issuing system that sells rail tickets, but it contains several ambiguities and omissions.

- "An automated ticket-issuing system sells rail tickets."
- "Users select their destination and input a credit card and a personal identification number."
- "The rail ticket is issued, and their credit card account charged."
- "When the user presses the start button, a menu display of potential destinations is activated, along with a message to the user to select a destination."

Solutions:

1. **Ambiguity**: It is not specified what happens if the card is invalid or the transaction is declined.
    - **Omission**: There is no mention of error handling for incorrect input (e.g., wrong PIN or card).
2. **Ambiguity**: It is unclear if the system supports multiple languages or if the user can cancel the operation at any stage.
    - **Omission**: There is no clarity on how the system handles input errors or invalid destinations.
3. **Ambiguity**: The description does not explain what happens after the transaction fails—does the user retry or get a refund if the card is charged incorrectly?

    - **Omission**: No mention of the timing between steps (e.g., how long the system waits for input before canceling the operation).
4. **Ambiguity**: The statement doesn't specify if users are presented with a confirmation screen after selecting the destination or after entering credit card information.

4.3. Rewrite the above description using the structured approach described in this chapter.

Scenario:

The initial description is ambiguous and lacks structure. A better-structured description should clarify these ambiguities and omissions.

Solutions:

1. **Start Button**: When the user presses the start button, the system activates a menu that displays potential rail destinations. A message prompts the user to select a destination from the displayed list.
2. **Destination Selection**: Once the user selects a destination, the system requests the user to input a credit card.
3. **Card Validation**: The system checks the validity of the credit card. If the card is invalid, the user is informed, and the transaction is canceled. If valid, the system prompts the user to input their personal identification number (PIN).
4. **PIN Validation**: Upon entering the PIN, the system validates the personal identifier. If the identifier is invalid, the user is prompted to retry. If valid, the transaction is processed.
5. **Ticket Issuance**: Once the transaction is successfully processed, the ticket is printed, and the user's account is debited. If the transaction fails at any step, an error message is displayed, and no ticket is issued.
6. **Cancellation**: At any point, the user should be able to cancel the transaction.

4.4. Write a set of non-functional requirements for the ticket-issuing system, setting out its expected reliability and response time.

Scenario:

Non-functional requirements set performance, reliability, and quality benchmarks for the system to ensure it functions as expected under real-world conditions.

Solutions:

1. **Reliability**: The system must be available 99.9% of the time and have less than 1% transaction failure rate.
2. **Response Time**: The system should respond to any user input (e.g., button presses or card entry) within 2 seconds.
3. **Security**: The system must comply with PCI DSS (Payment Card Industry Data Security Standard) for handling credit card transactions and personal identification numbers.
4. **Capacity**: The system should handle up to 500 transactions per hour without a degradation in performance.
5. **Maintainability**: The system should be easy to update, with system downtime for updates limited to non-peak hours (between 12 AM and 4 AM).

4.5. Write plausible user requirements using a standard format.

Scenario:

Here are plausible user requirements for three systems based on their functionalities.

Solutions:

1. **Unattended Petrol Pump System**:
    - **Description**: The system allows customers to purchase fuel without assistance by using a credit card.
    - **User Action**: The customer swipes the card and inputs the desired amount of fuel.
    - **System Action**: The system verifies the card and dispenses the fuel. Once the delivery is complete, the customer's account is debited.
2. **Bank ATM Cash-Dispensing Function**:
    - **Description**: The system allows users to withdraw cash from their bank accounts.
    - **User Action**: The user inserts their card, inputs their PIN, and selects the amount of cash to withdraw.
    - **System Action**: The system verifies the card, checks the account balance, and dispenses the requested cash.
3. **Word Processor Spelling-Check Function**:
    - **Description**: The spelling check automatically identifies and highlights spelling errors in a document.
    - **User Action**: The user writes a document, and the system checks spelling in real-time or when prompted.
    - **System Action**: The system underlines misspelled words, suggests corrections, and allows the user to apply the correction.

4.6. Suggest how an engineer might track the relationships between functional and non-functional requirements.

Scenario:

Tracking relationships between functional and non-functional requirements is crucial to ensure consistency and coverage in system development.

Solutions:

1. **Traceability Matrix**: Engineers can use a traceability matrix that maps each functional requirement to its corresponding non-functional requirements, ensuring that all aspects of performance, reliability, and security are addressed for each feature.
2. **Requirement Management Tools**: Tools like Jira, IBM DOORS, or Rational RequisitePro can be used to link functional requirements with their non-functional counterparts.

4.7. Develop a set of use cases for an ATM system.

Scenario:

Understanding ATM use cases can help in defining the system requirements.

Solutions:

- **Use Case 1**: Withdraw Cash
  - User inserts card, enters PIN, selects withdrawal option, chooses amount, and collects cash.
- **Use Case 2**: Check Balance
  - User inserts card, enters PIN, and selects the balance inquiry option.
- **Use Case 3**: Deposit Cash
  - User inserts card, enters PIN, selects the deposit option, inserts cash, and confirms.

4.8. Who should be involved in a requirements review?

Scenario:

A thorough review of system requirements requires input from various stakeholders to ensure completeness and accuracy.

Solutions:

1. **Review Participants**: Project managers, developers, testers, domain experts, customers, and end-users should all be involved in the review process.
2. **Process Model**:
    - Collect requirements.
    - Review by stakeholders.
    - Conduct feedback sessions.
    - Make revisions.
    - Final approval.

4.9. Suggest a model of a process for emergency software changes.

Scenario:

In some cases, emergency changes to system software must be made without updating the requirements first.

Solutions:

1. **Model**:
    - Log the emergency change request.
    - Perform an impact analysis.
    - Implement and test the change.
    - Update the requirements document post-implementation.
    - Conduct a review to ensure consistency.

4.10. What should you do when there is a disagreement in requirement interpretation?

Scenario:

When two companies interpret requirements differently, there can be cost and confidentiality implications.

Solutions:

- **Step 1**: Report the ambiguity to both employers, emphasizing the need for clarity.
- **Step 2**: Facilitate a discussion to resolve the ambiguity while maintaining confidentiality.
- **Step 3**: Document the final agreed-upon requirements to avoid future misunderstandings.

Assignment Questions

Level 1

- Write a function that takes a list as input and returns the sum of all elements in the list.
- Write a function that takes a list of numbers and returns the largest number in the list.
- Write a function that takes a list and returns the reverse of the list.
- Write a function that removes all duplicates from a list.
- Write a function that takes two lists and returns their intersection (common elements).

Level 2

- Write a function that takes a list of strings and returns a list of the strings sorted by length.
- Write a function that takes a list and a number `n` and returns the first `n` elements of the list.
- Write a function that takes a list of numbers and returns a new list containing only the even numbers.
- Write a function that flattens a nested list of lists.
- Write a function that takes a list of numbers and returns the second largest number.

Level 3

- Write a function that takes a list and returns all possible subsets of that list.
- Write a function that takes a list and returns the list rotated to the right by `n` places.
- Write a function that takes a list of numbers and returns `True` if the list is sorted in ascending order, otherwise `False`.
- Write a function that returns the most frequent element in a list. If there are ties, return any one of the most frequent elements.
- Write a function that takes two lists and returns `True` if they are permutations of each other.

Solutions

Level 1:

Write a function that takes a list as input and returns the sum of all elements in the list.

```
def sum_list(lst):
    return sum(lst)
```

Write a function that takes a list of numbers and returns the largest number in the list.

```
def max_in_list(lst):
    return max(lst)
```

Write a function that takes a list and returns the reverse of the list.

```
def reverse_list(lst):
    return lst[::-1]
```

Write a function that removes all duplicates from a list.

```
def remove_duplicates(lst):
    return list(set(lst))
```

Write a function that takes two lists and returns their intersection (common elements).

```
 def list_intersection(lst1, lst2):
    return list(set(lst1) & set(lst2))
```

Level 2:

Write a function that takes a list of strings and returns a list of the strings sorted by length.

```
def sort_by_length(lst):
    return sorted(lst, key=len)
```

Write a function that takes a list and a number n and returns the first n elements of the list.

```
def first_n_elements(lst, n):
    return lst[:n]
```

Write a function that takes a list of numbers and returns a new list containing only the even numbers.

```
def filter_even_numbers(lst):
    return [x for x in lst if x % 2 == 0]
```

Write a function that flattens a nested list of lists.

```
def flatten_list(nested_lst):
    return [item for sublist in nested_lst for item in sublist]
```

Write a function that takes a list of numbers and returns the second largest number.

```
def second_largest(lst):
    return sorted(list(set(lst)))[-2]
```

Level 3:

Write a function that takes a list and returns all possible subsets of that list.

```
from itertools import combinations
def all_subsets(lst):
    subsets = []
```

```
    for i in range(len(lst)+1):
        subsets.extend(list(combinations(lst, i)))
    return subsets
```

Write a function that takes a list and returns the list rotated to the right by n places.

```
def rotate_list(lst, n):
    return lst[-n:] + lst[:-n]
```

Write a function that takes a list of numbers and returns True if the list is sorted in ascending order, otherwise False.

```
def is_sorted(lst):
    return lst == sorted(lst)
```

Write a function that returns the most frequent element in a list. If there are ties, return any one of the most frequent elements.

```
from collections import Counter
def most_frequent(lst):
    return Counter(lst).most_common(1)[0][0]
```

Write a function that takes two lists and returns True if they are permutations of each other.

```
def are_permutations(lst1, lst2):
    return sorted(lst1) == sorted(lst2)
```

Previous :

Assignment - Questions :

Level 1

1. Write a for loop to print all even numbers between 1 and 20.
2. Write a for loop to print numbers between 30 and 50 that are divisible by 5.
3. Write a for loop to print the square of each number from 1 to 10.
4. Write a while loop to print all odd numbers between 1 and 15.
5. Write a while loop to print numbers from 100 down to 50 that are divisible by 10.
6. Write a for loop to print all prime numbers between 1 and 50.
7. Write a while loop to print the first 10 numbers of the Fibonacci sequence.
8. Write a for loop to print the multiplication table of 7.
9. Write a while loop to keep doubling a number until it exceeds 1000, starting from 1.
10. Write a for loop to print numbers between 1 and 20, but skip numbers divisible by 3.

Level 2

1. Write a for loop to print all numbers between 1 and 50, but stop the loop when you encounter a number divisible by both 8 and 9.
2. Write a while loop to calculate the sum of all numbers between 1 and 100 that are divisible by 4, and print the result.
3. Write a for loop to print all the digits of a given number in reverse order (e.g., for 1234, print 4 3 2 1).
4. Write a while loop to find the smallest number greater than 500 that is divisible by both 7 and 13.
5. Write a for loop to print the first 10 terms of the arithmetic sequence starting with 5, with a common difference of 3 (i.e., 5, 8, 11, ...).
6. Write a while loop to find the factorial of a given number, and print the result.

Data Modeling

References : https://www.lucidchart.com/pages/er-diagrams

Data modeling in software engineering helps in organizing the structure of a system's data to ensure efficient storage, retrieval, and relationships between entities. By using concepts like entities, relationships, normalization, and ER diagrams, software developers can design databases that are both efficient and scalable.

In software engineering, data modeling is the process of creating a visual representation of a system's data and its structure. It defines how data is stored, accessed, and related within a system. The goal is to establish a blueprint for the data architecture, helping developers understand the system's data flow and database design. Let's explore key data modeling concepts with examples:

1. Entities

An entity represents a real-world object or concept that has data stored about it. Each entity is characterized by attributes.

- Example: In a student management system, entities could be `Student`, `Course`, and `Instructor`.

2. Attributes

Attributes are the properties or characteristics of an entity. They define the data that each entity stores.

- Example: For the entity `Student`, attributes could include `StudentID`, `Name`, `DateOfBirth`, `Email`.

3. Relationships

Relationships describe how entities are related to each other. These relationships can be one-to-one, one-to-many, or many-to-many.

- Example: In a school system:
  - A one-to-many relationship: One `Instructor` teaches many `Courses`.
  - A many-to-many relationship: Many `Students` can enroll in many `Courses`.

4. Primary Key

A primary key is a unique identifier for an entity, ensuring that each record in a table can be uniquely identified.

- Example: For the `Student` entity, `StudentID` could be the primary key, uniquely identifying each student.

5. Foreign Key

A foreign key is an attribute in one entity that refers to the primary key of another entity, establishing a relationship between the two entities.

- Example: In a `Course` entity, `InstructorID` could be a foreign key that refers to the `InstructorID` in the `Instructor` entity, linking courses to their respective instructors.

6. Normalization

Normalization is the process of organizing data to reduce redundancy and dependency. It involves dividing large tables into smaller ones and establishing relationships between them to ensure data integrity.

- Example: Suppose there is a single table that stores both student and course information, leading to data redundancy (e.g., storing the same course details for multiple students). By normalizing the table, the `Student` and `Course` entities **would** be split into separate tables, and a new `Enrollment` table would handle the relationship between students and courses.

7. Cardinality

Cardinality refers to the number of instances of one entity that can be related to another entity.

- Example: In a library system:
    - A one-to-many cardinality: One `Library` has many `Books`.
    - A many-to-many cardinality: Many `Members` can borrow many `Books`.

8. ER Diagrams (Entity-Relationship Diagrams)

An ER Diagram is a visual representation of the entities, their attributes, and relationships within a system. It's widely used in data modeling to describe how data is structured and interrelated.

- Example: In an e-commerce system:
    - Entities: `Customer`, `Order`, `Product`
    - Relationships: `Customer` places many `Orders`; `Order` contains many `Products`.

9. Data Types

Data types define the kind of data that attributes can store (e.g., integer, string, date).

- Example: In a banking system:
    - `AccountNumber` could be an integer.
    - `AccountHolderName` would be a string.
    - `DateOfTransaction` could be of date type.

10. Schema

A schema represents the structure of the database, including the tables, fields, and relationships between tables. It defines how data is organized in the system.

- Example: In a payroll system, the schema may consist of tables such as `Employee`, `Salary`, `Department`, each with its own attributes and relationships.

Example Scenario: Library Management System

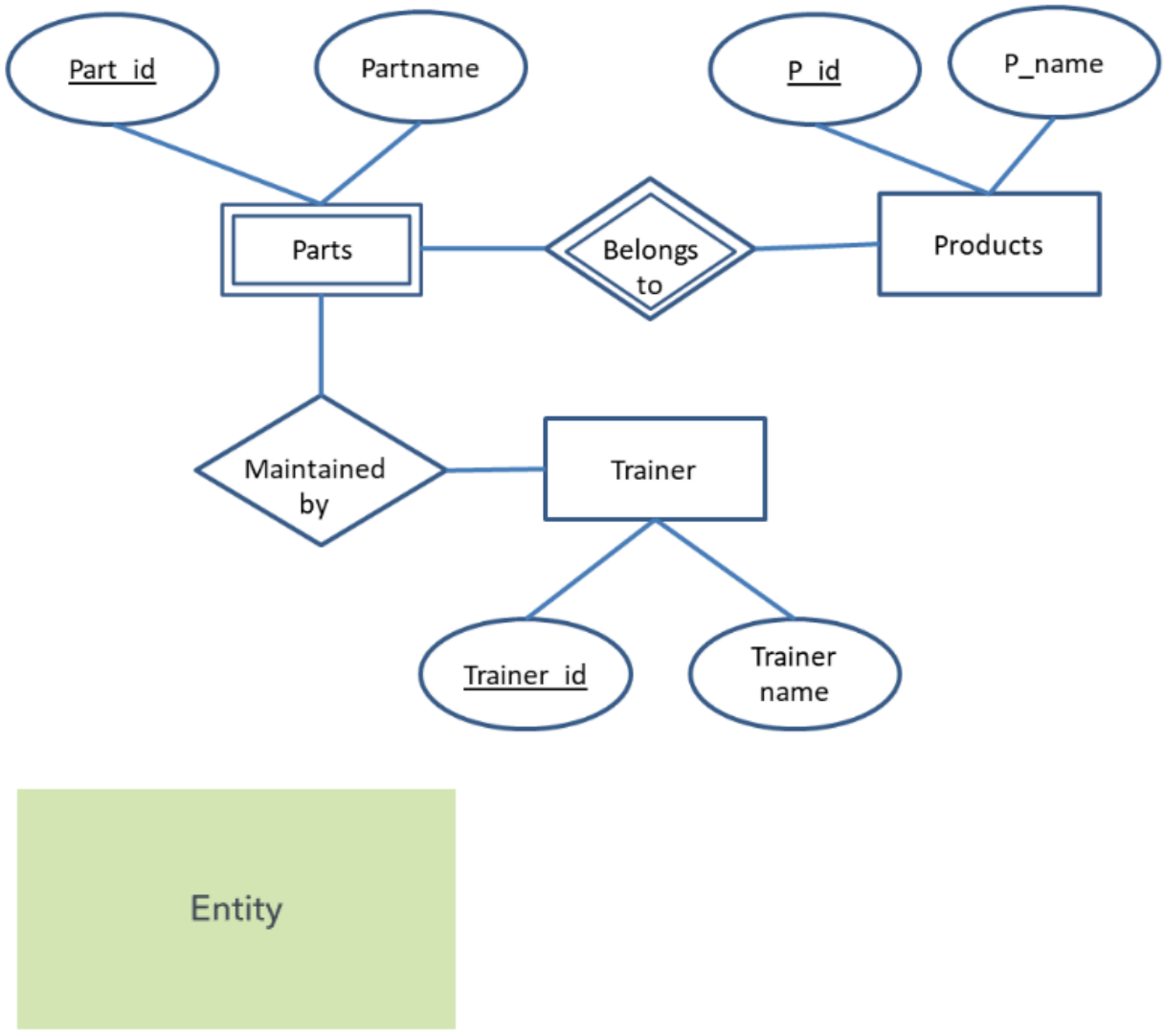
Let's build a data model for a Library Management System:

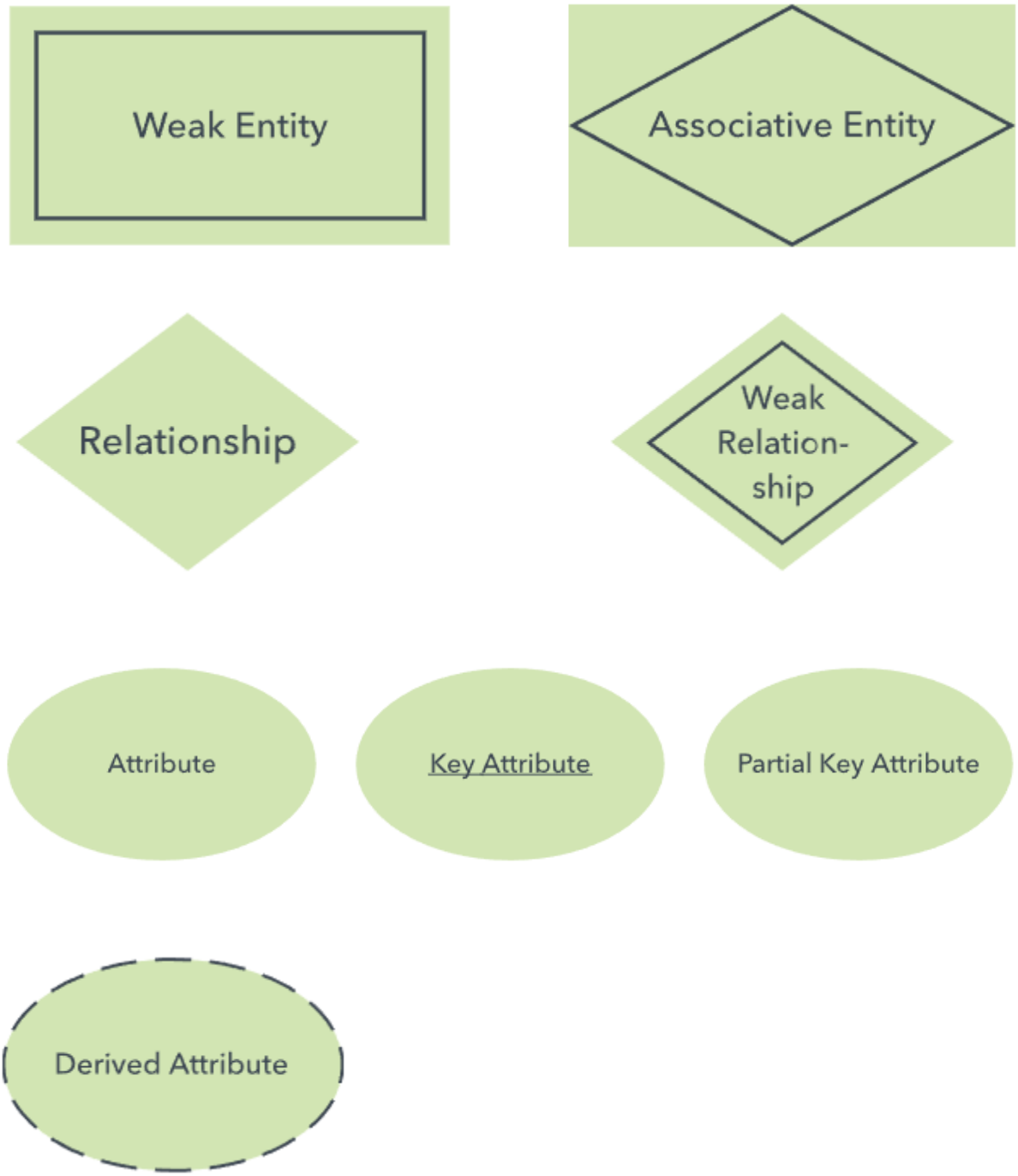
- Entities:
    - `Book` (attributes: `BookID`, `Title`, `Author`)
    - `Member` (attributes: `MemberID`, `Name`, `Email`)
    - `Loan` (attributes: `LoanID`, `IssueDate`, `ReturnDate`)
- **Relationships:**
    - A `Member` can borrow many `Books`, and a `Book` can be borrowed by many `Members` (many-to-many relationship), managed through the `Loan` entity.

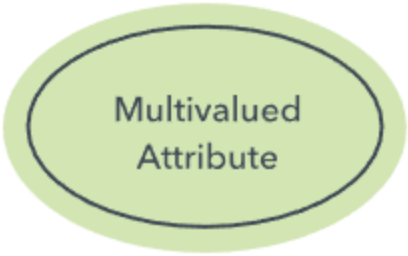
The ER diagram for this system would show:

- `Member` and `Book` entities with their respective attributes.
- A `Loan` entity acting as a bridge table connecting `Member` and `Book`.

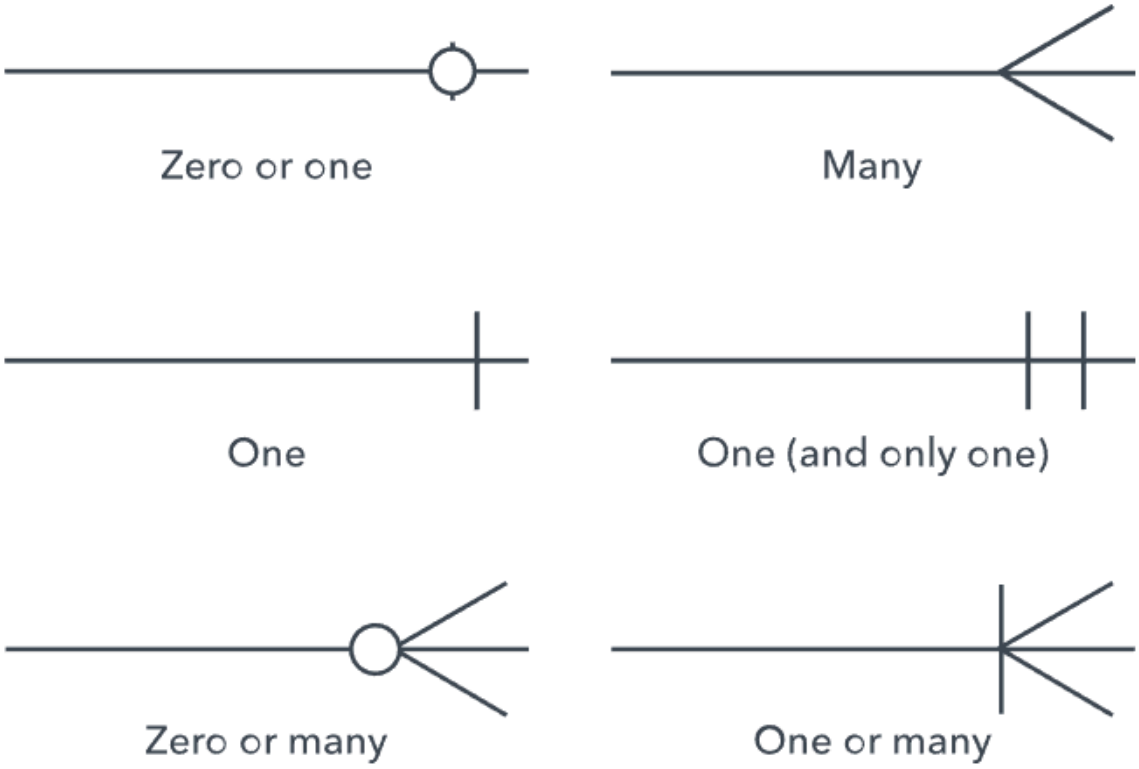
ER Diagram



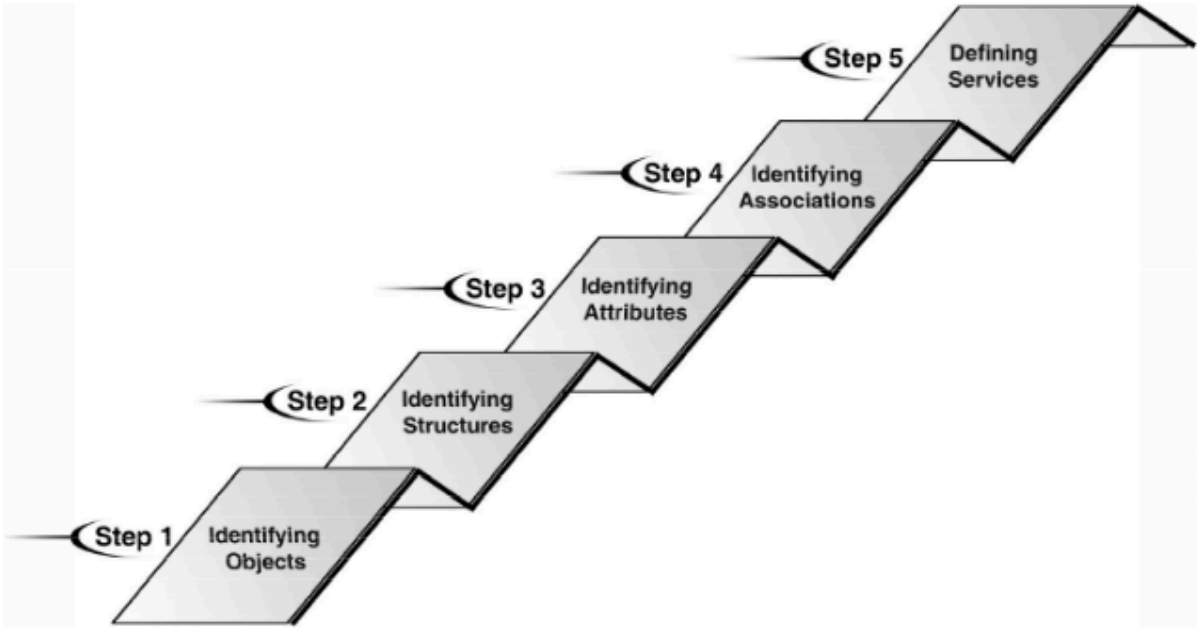

Weak Entity
Associative Entity
Relationship
Weak Relation-ship
Attribute
Key Attribute
Partial Key Attribute
Derived Attribute

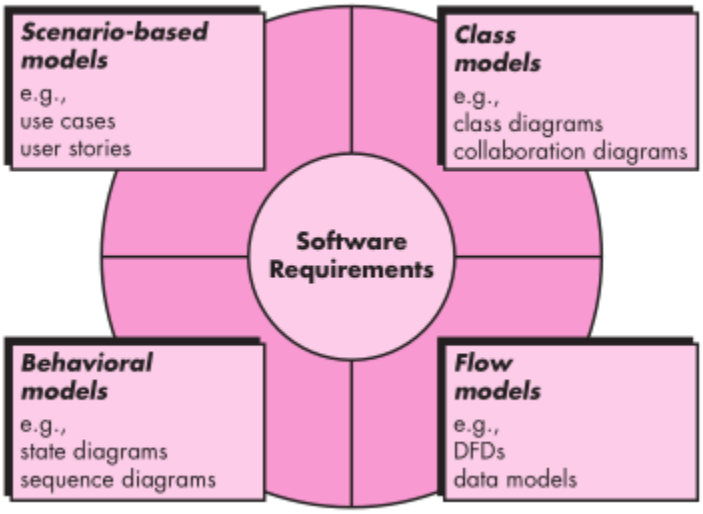


Cardinality



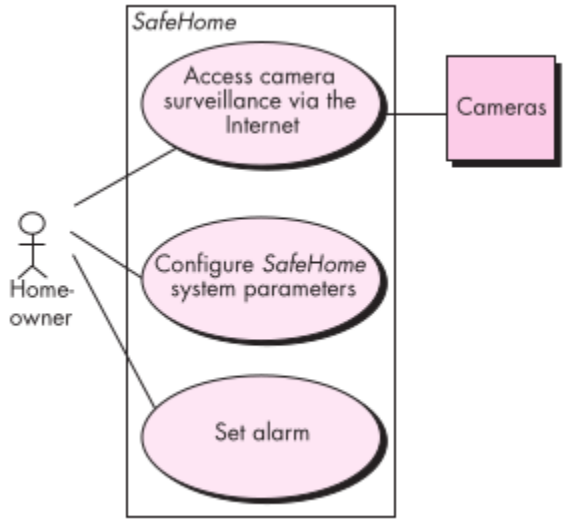
OOP analysis



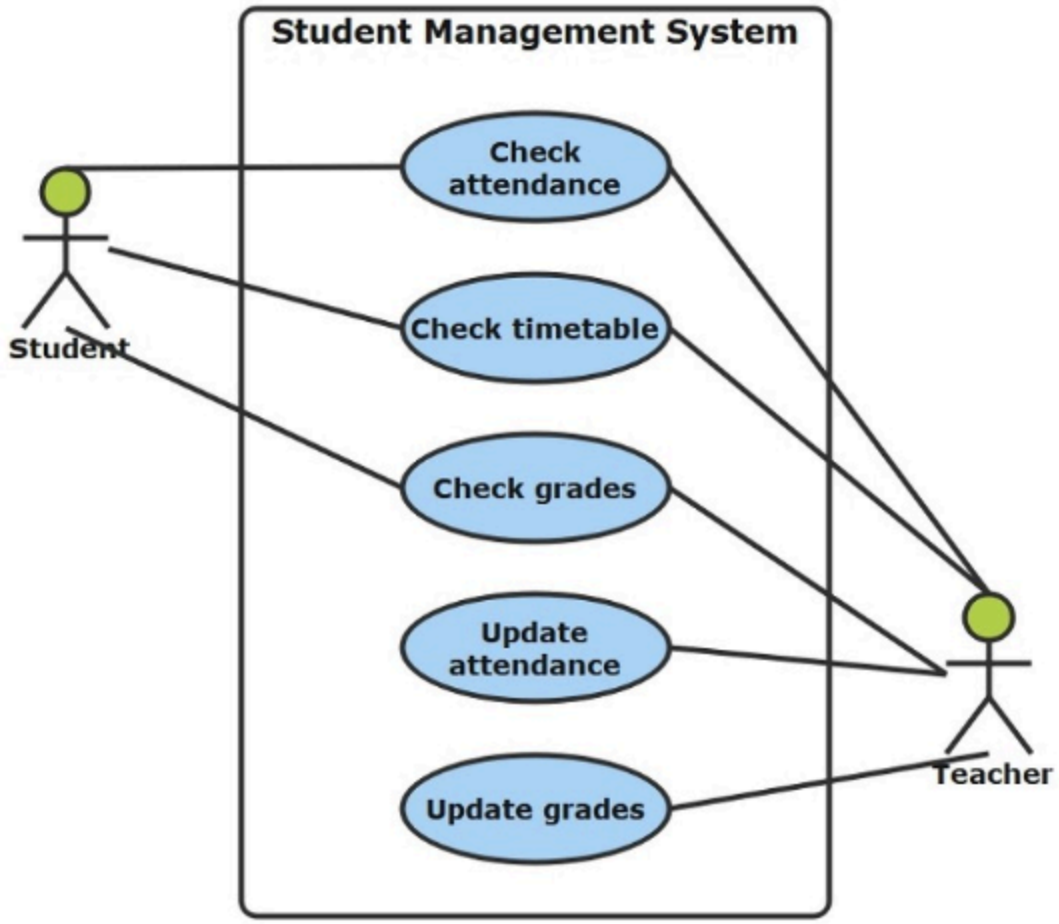


Use case diagrams

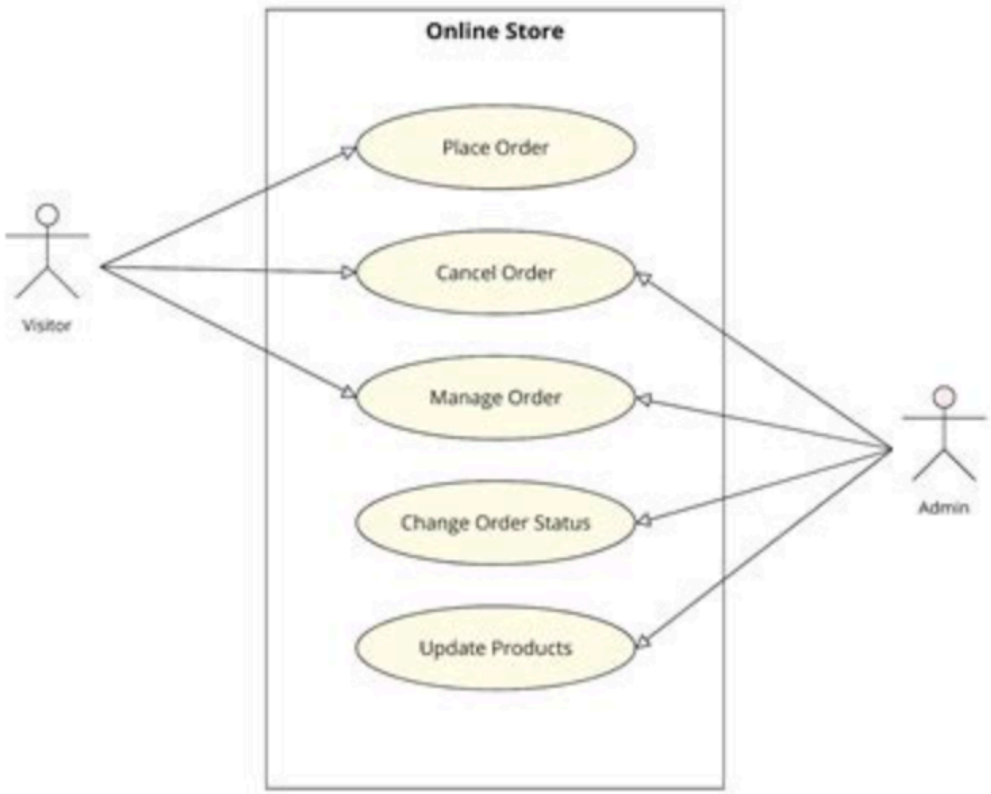
Safe home systems



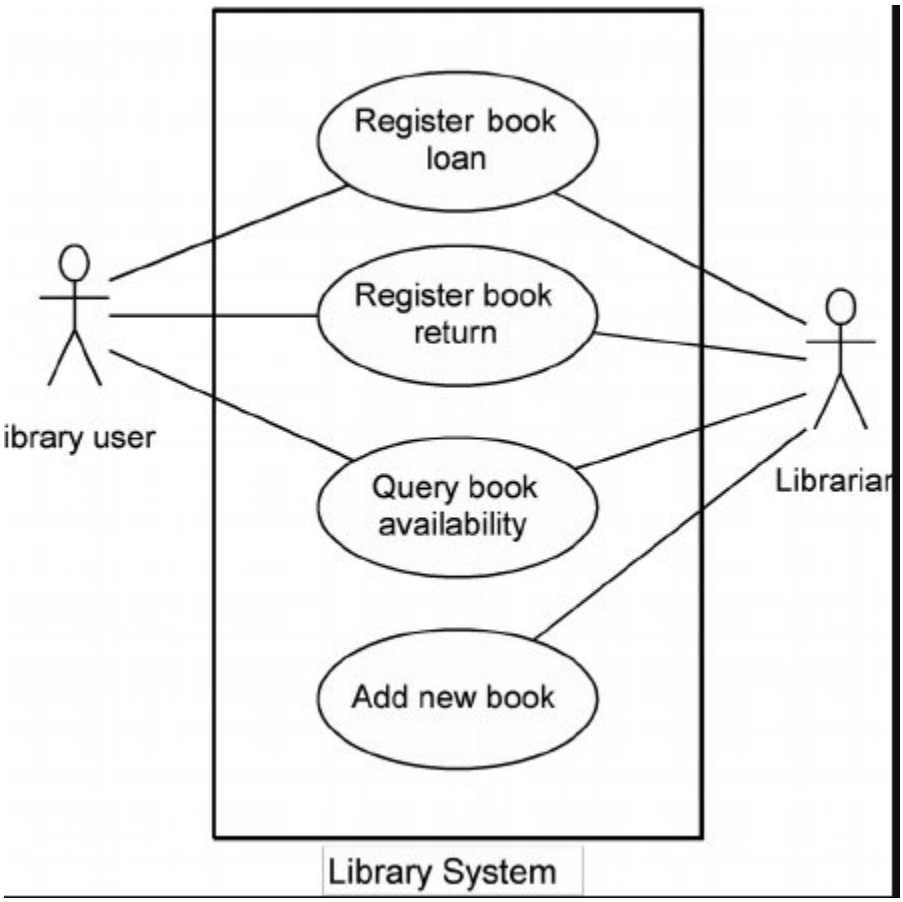
Student Management System

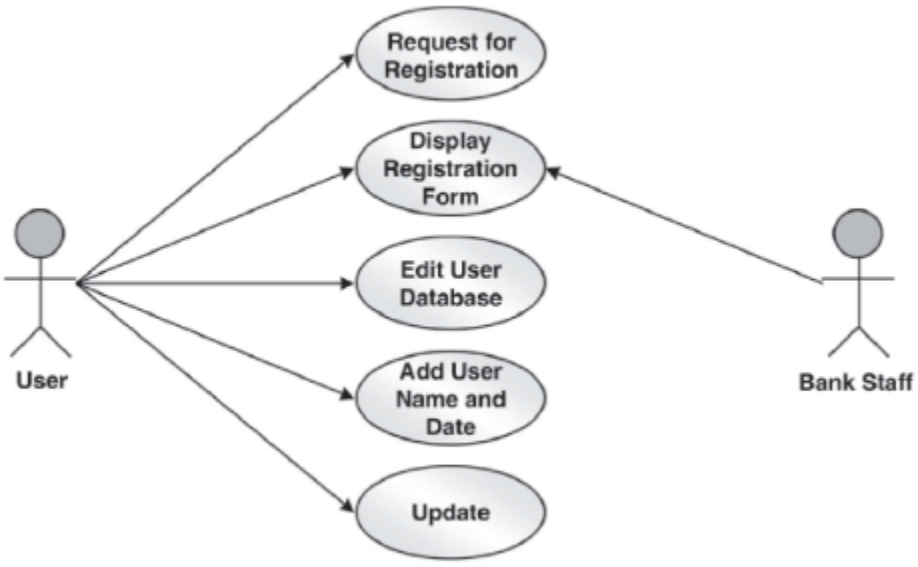

Student Management System
Check attendance
Check timetable
Check grades
Update attendance
Update grades
Student
Teacher

Online store

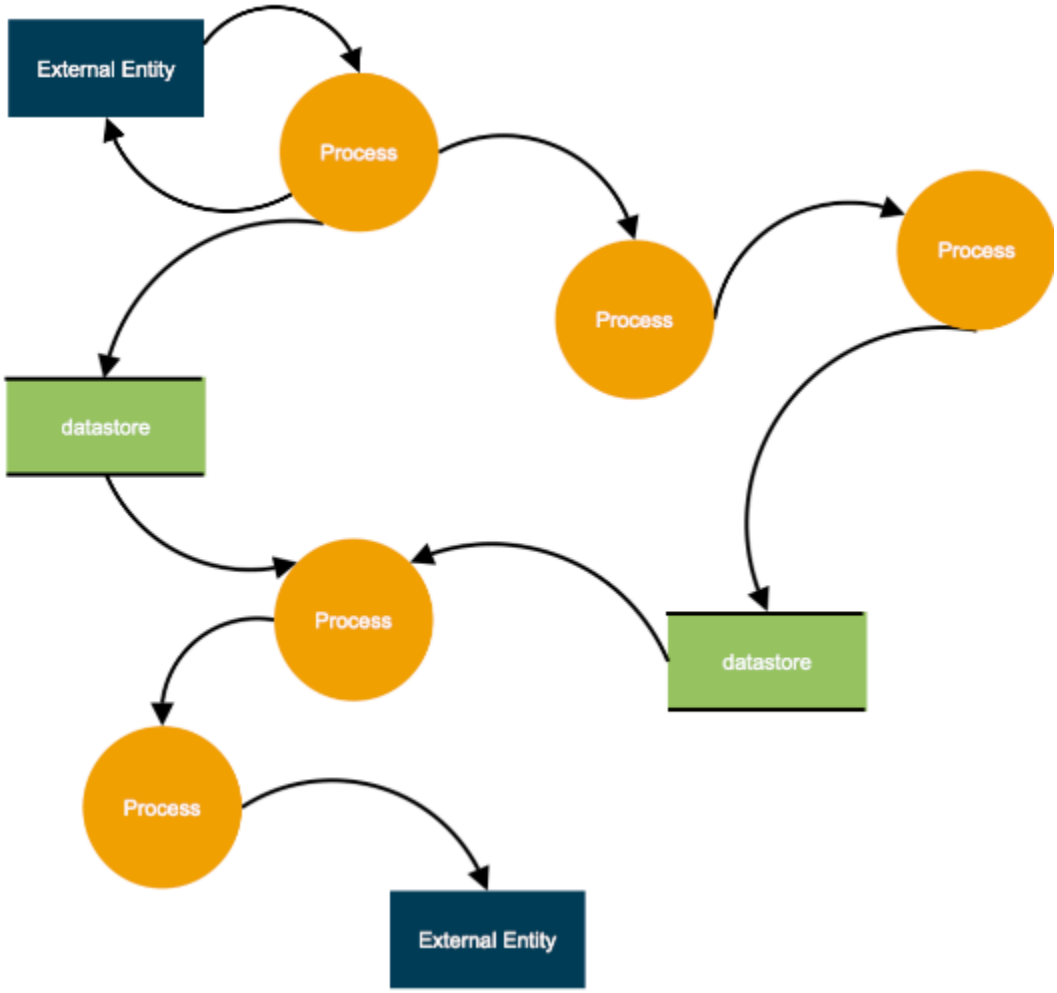


Library System





DFD Diagram


External Entity
Process
Process
Process
datastore
Process
datastore
Process
External Entity

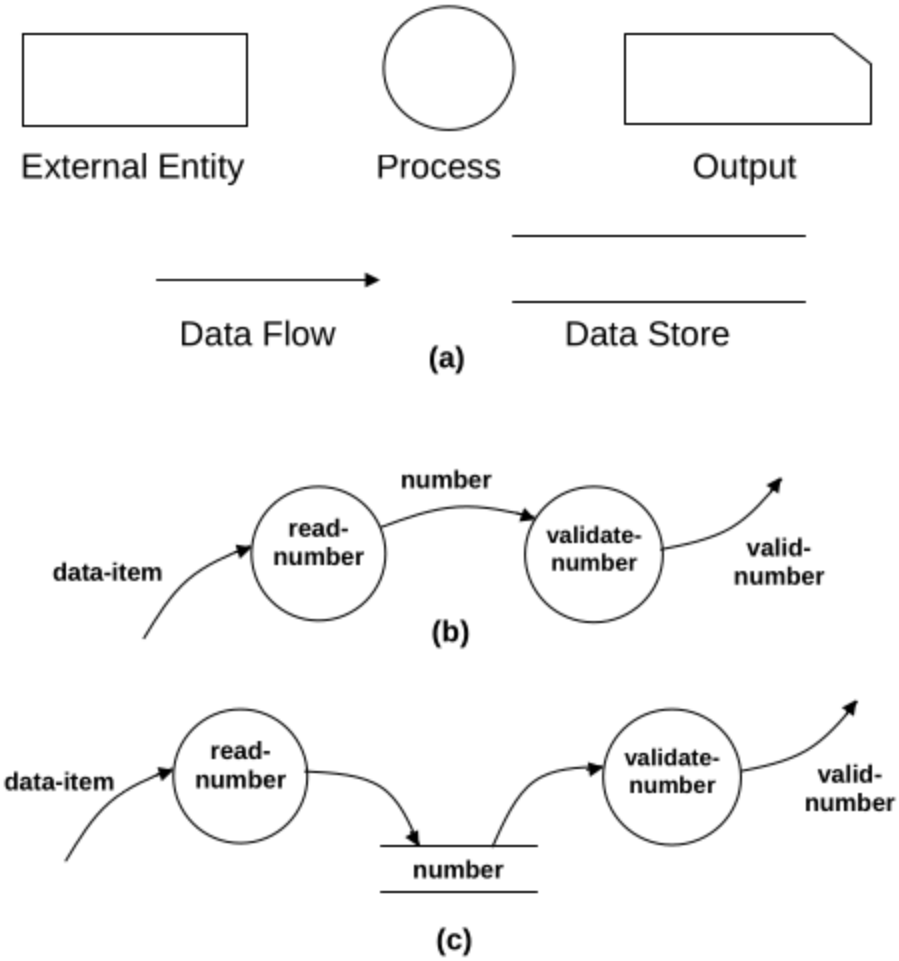


Fig. 5.1 (a) Symbols used for designing DFDs
(b), (c) Synchronous and asynchronous data flow

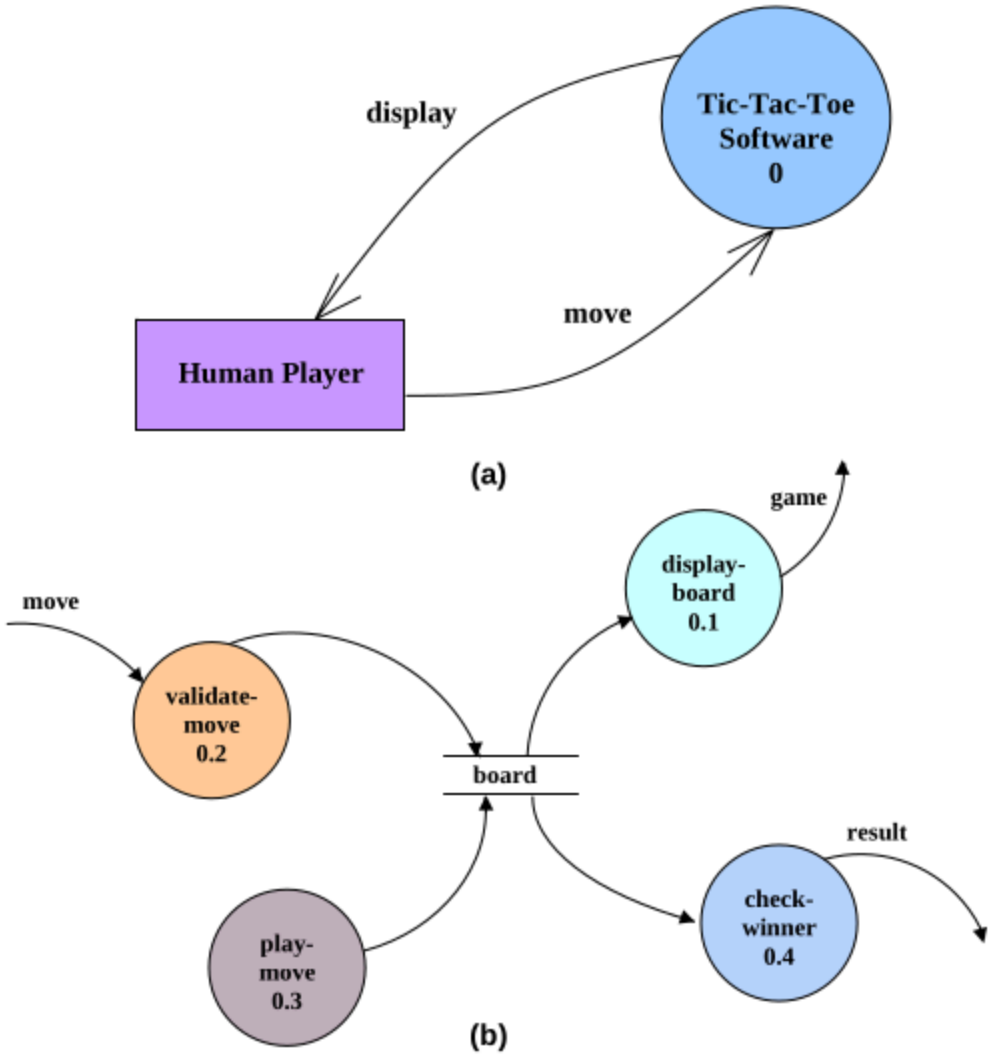
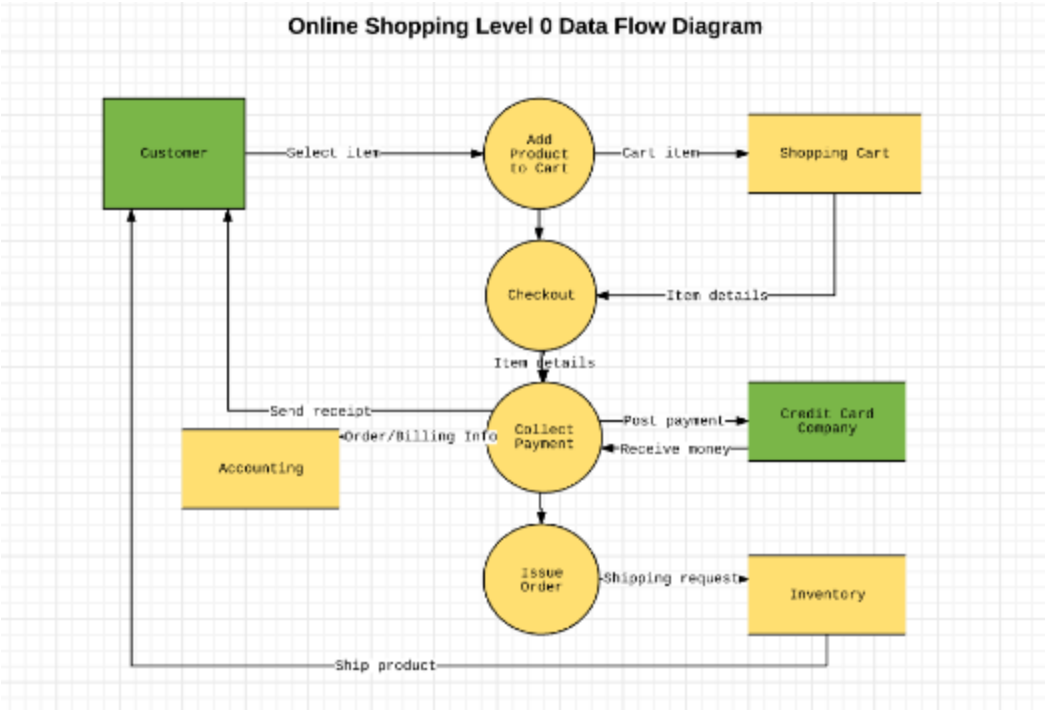


Fig 5.2 (a), (b) Level 0 and Level 1 DFD for Tic-Tac-Toe game described in Example 1



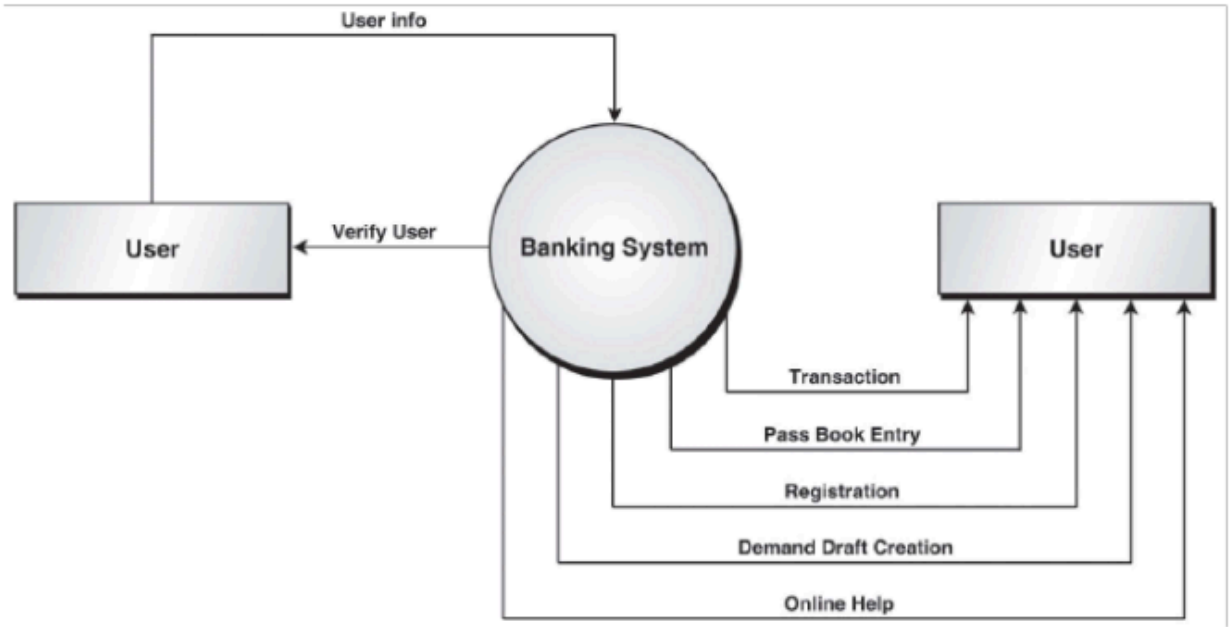


***Fig. 6.2** Level 0 DFD of Banking System*

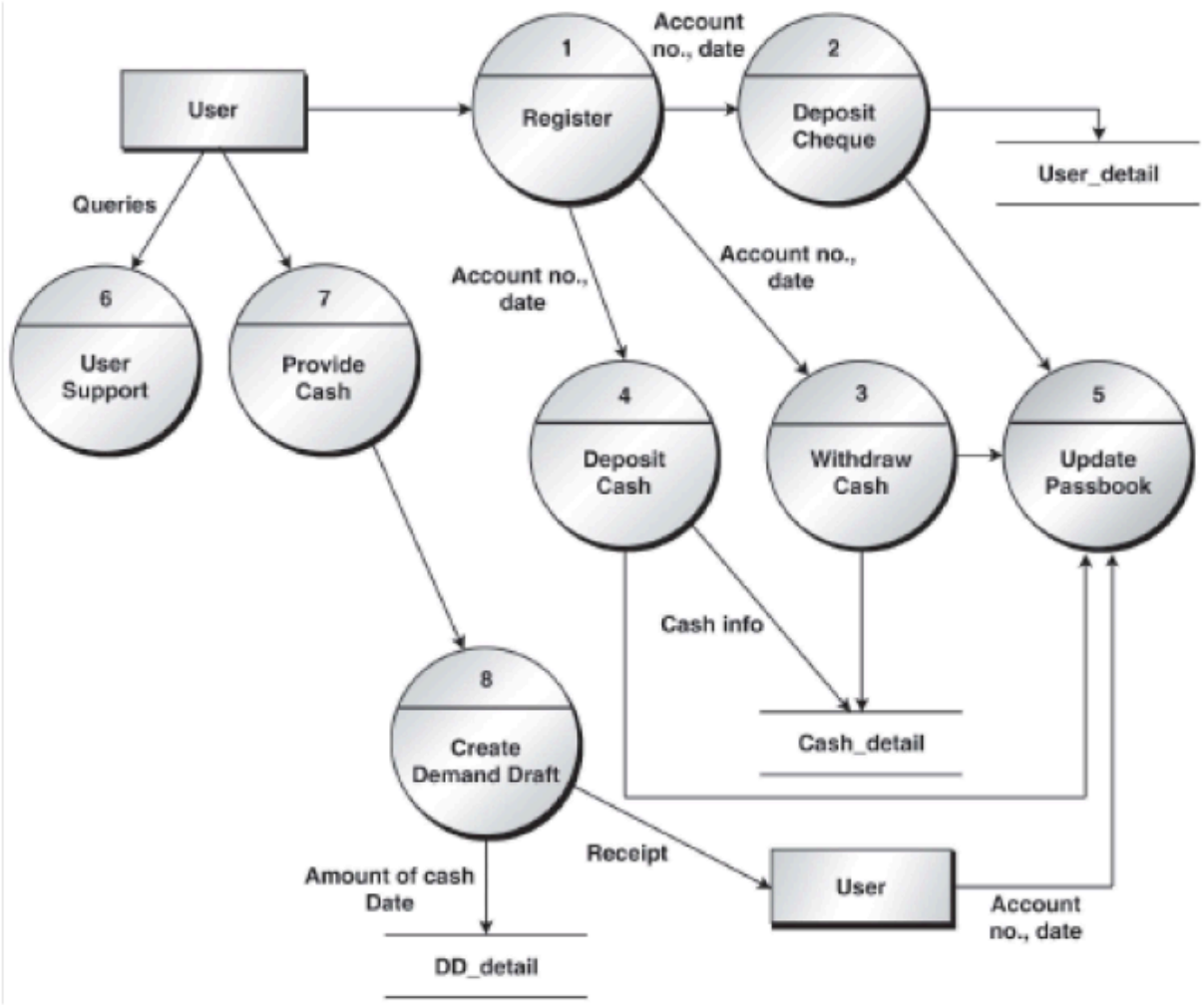


Fig. 6.3 *Level 1 DFD to Perform Transaction*

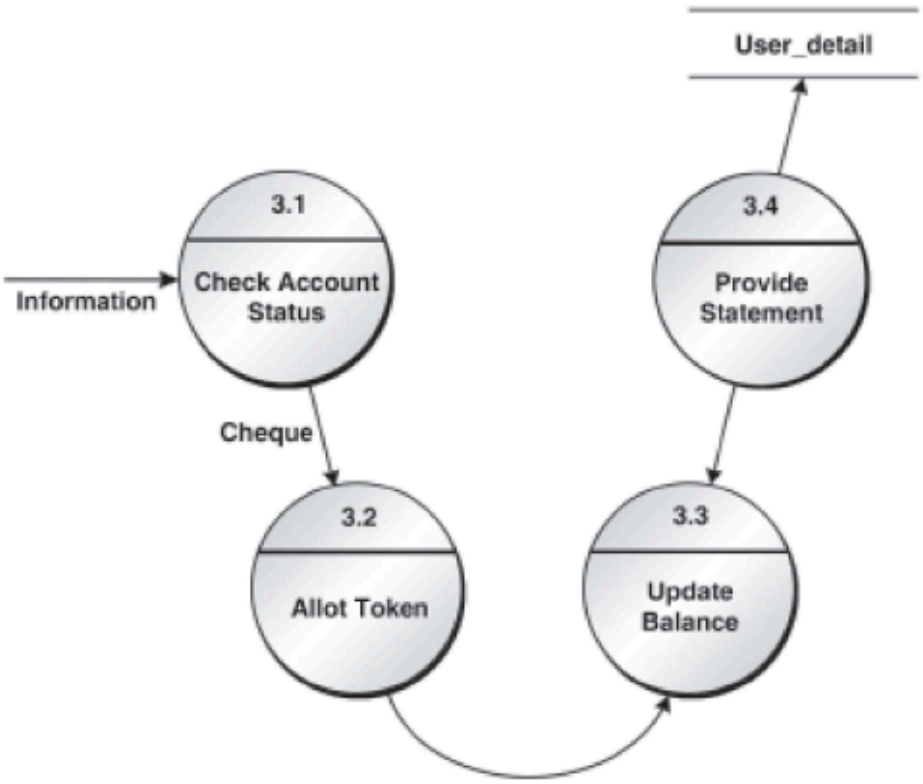


Fig. 6.4 *Level 2 DFD to Withdraw Cash*

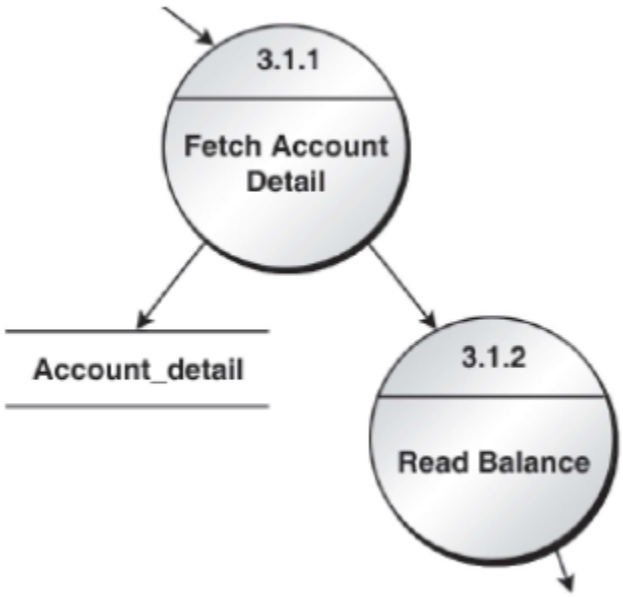
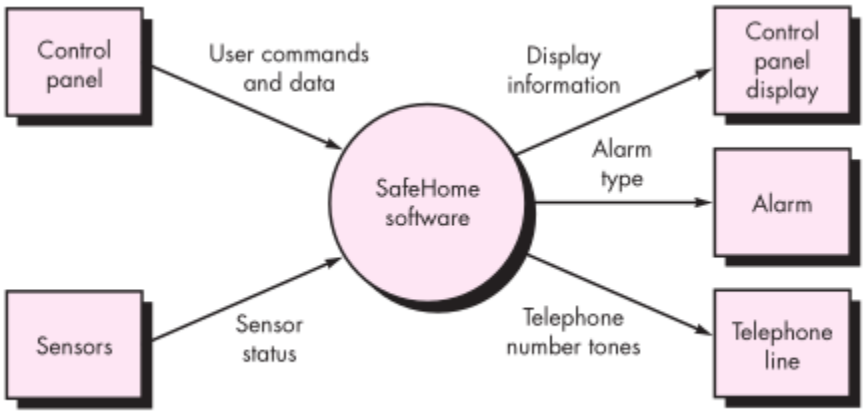
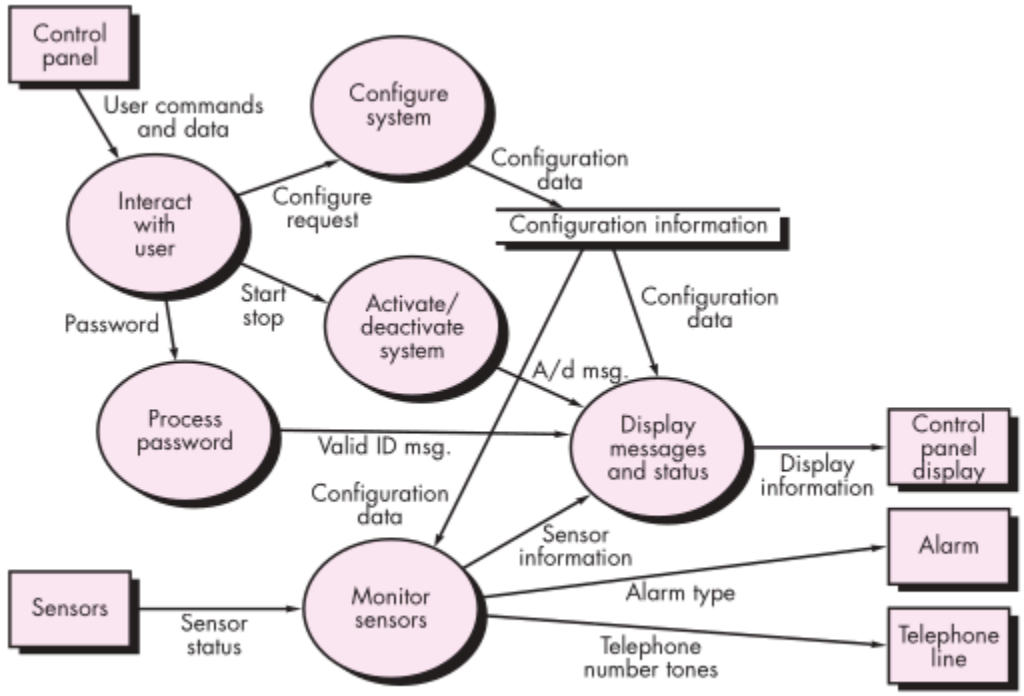


Fig. 6.5 *Level 3 DFD to Check Account Status*

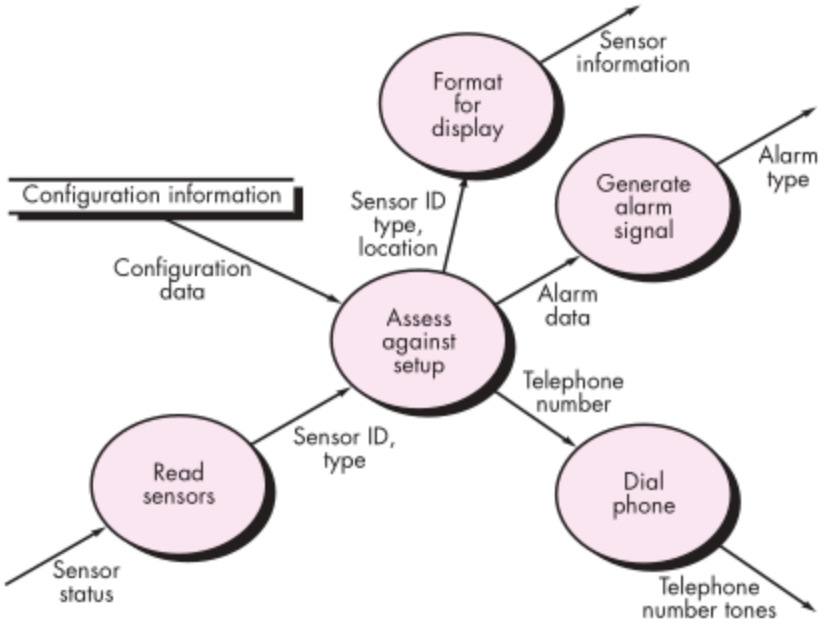
DFD diagram for safehome security function



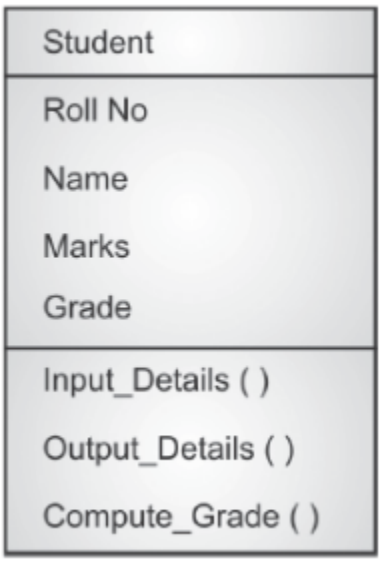
Level1 DFD - safehome

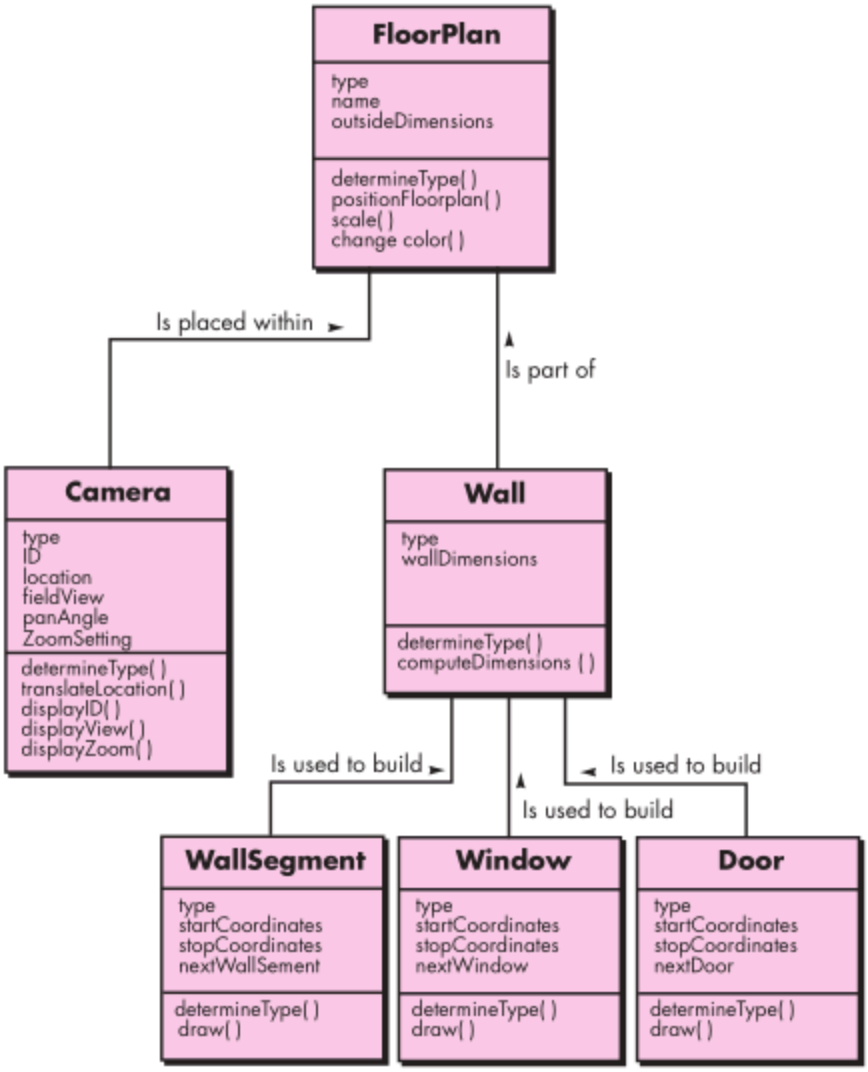


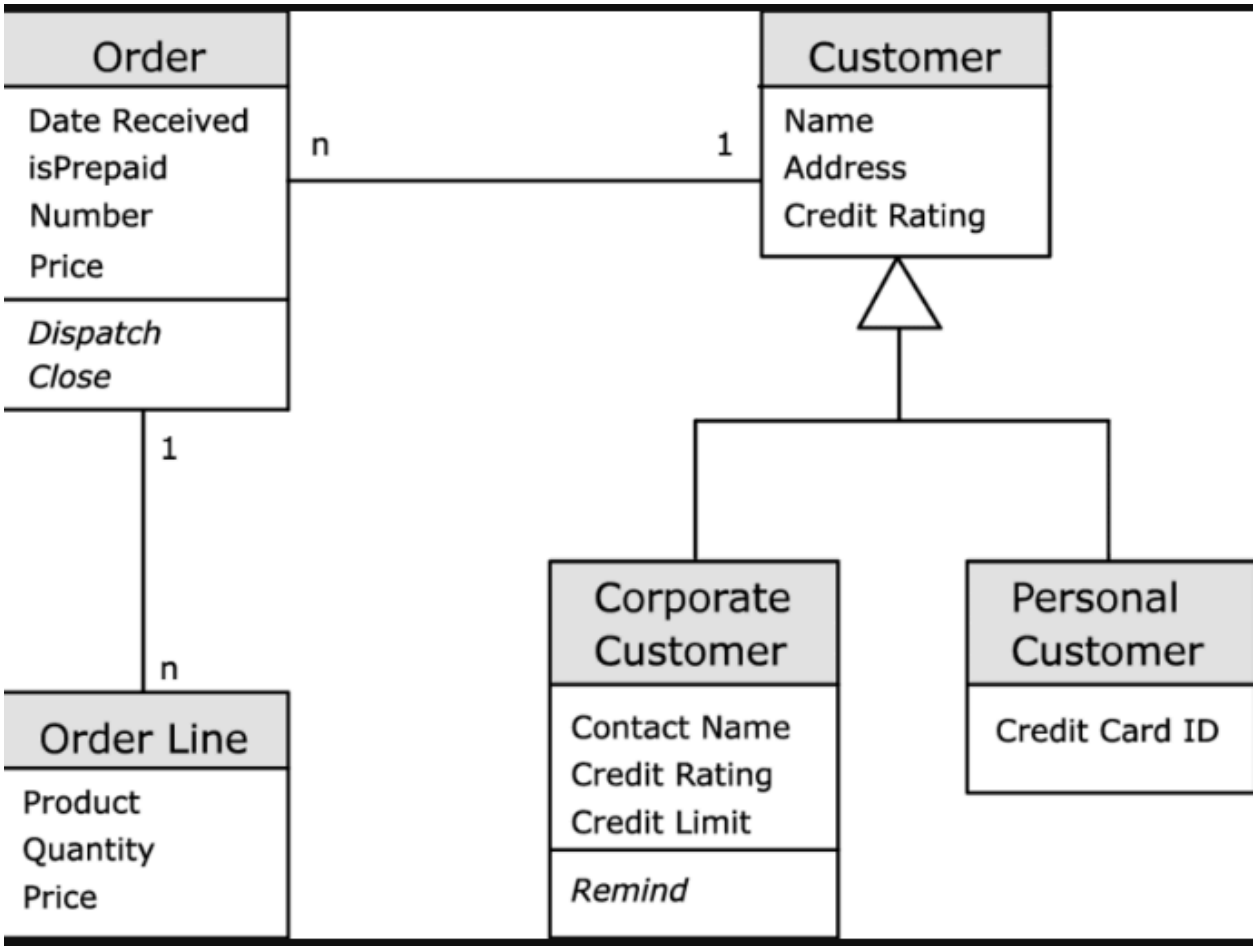
Level2 DFD - monitor sensors process



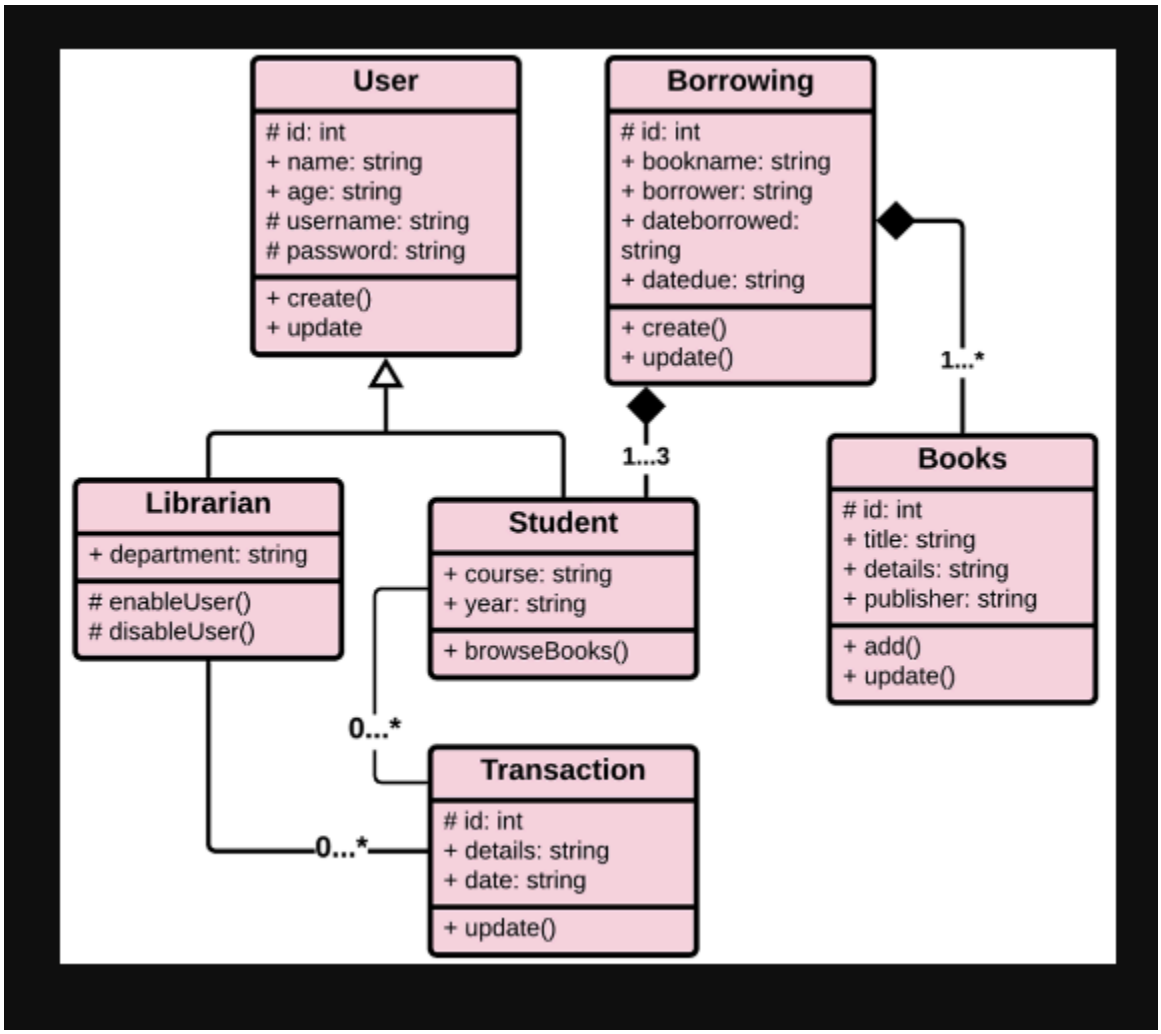
Class diagram

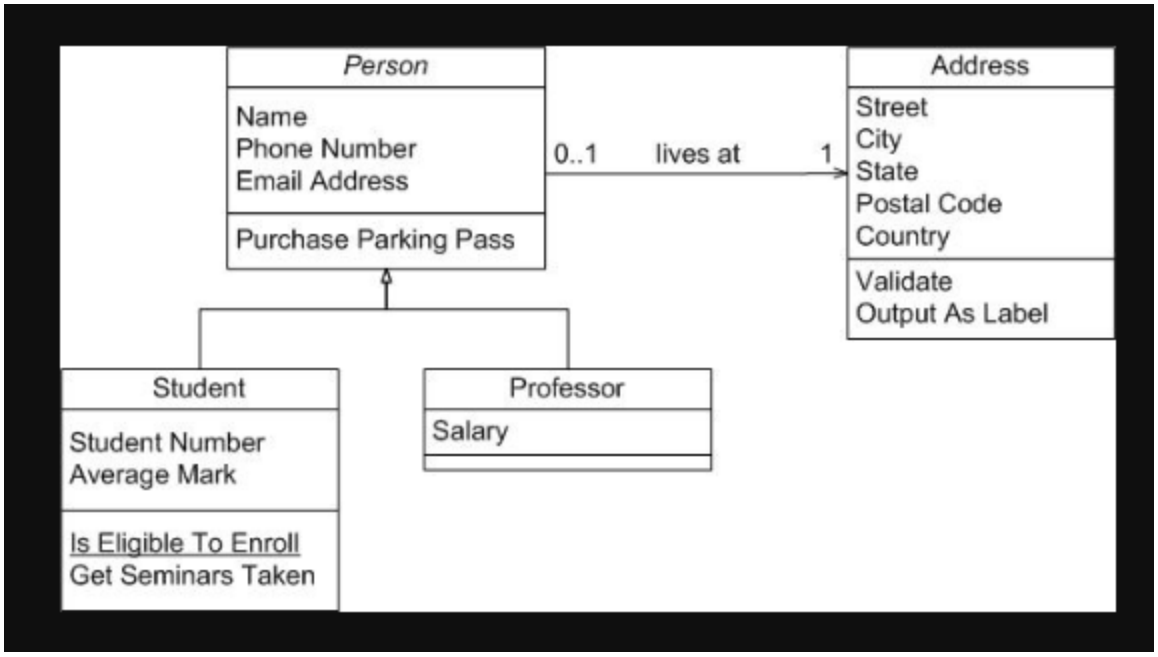



FloorPlan
type
name
outsideDimensions
determineType( )
positionFloorplan( )
scale( )
change color( )
Is placed within
Is part of
Camera
type
ID
location
fieldView
panAngle
ZoomSetting
determineType( )
translateLocation( )
displayID( )
displayView( )
displayZoom( )
Wall
type
wallDimensions
determineType( )
computeDimensions ( )
Is used to build
Is used to build
Is used to build
WallSegment
type
startCoordinates
stopCoordinates
nextWallSement
determineType( )
draw( )
Window
type
startCoordinates
stopCoordinates
nextWindow
determineType( )
draw( )
Door
type
startCoordinates
stopCoordinates
nextDoor
determineType( )
draw( )


Order
Date Received
isPrepaid
Number
Price
Dispatch
Close
n
1
Customer
Name
Address
Credit Rating
1
n
Order Line
Product
Quantity
Price
Corporate Customer
Contact Name
Credit Rating
Credit Limit
Remind
Personal Customer
Credit Card ID

Library Management System





Behavioral Modeling

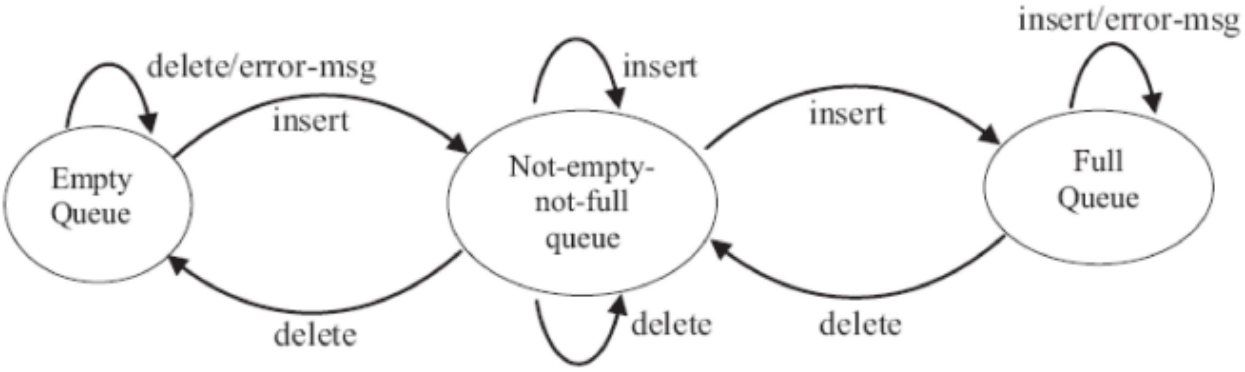
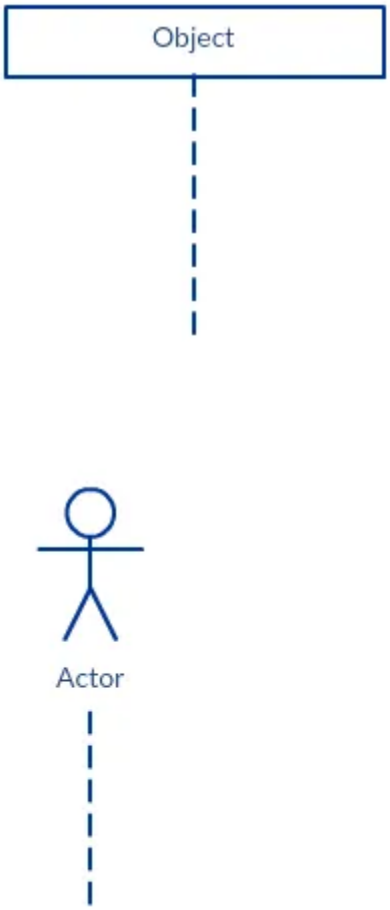
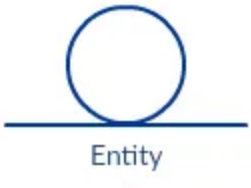


***Fig. 6.10** State Transition Diagram for Queue Object*

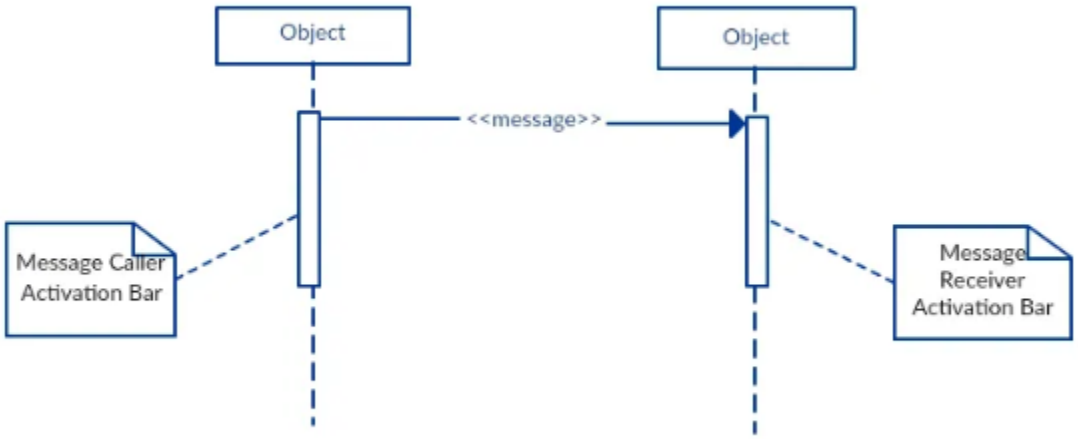
Sequence Diagrams


Object
Actor

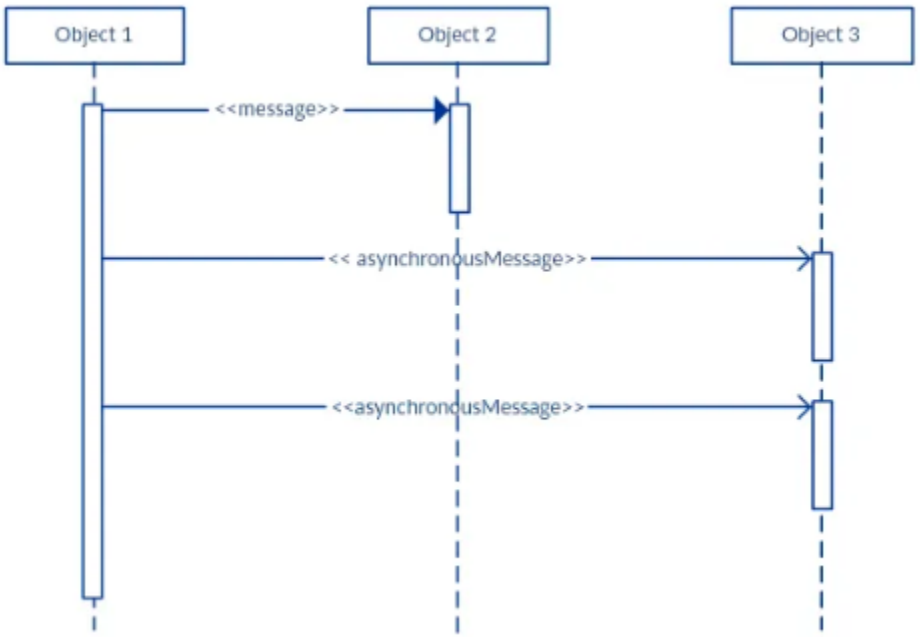

Entity


Boundary

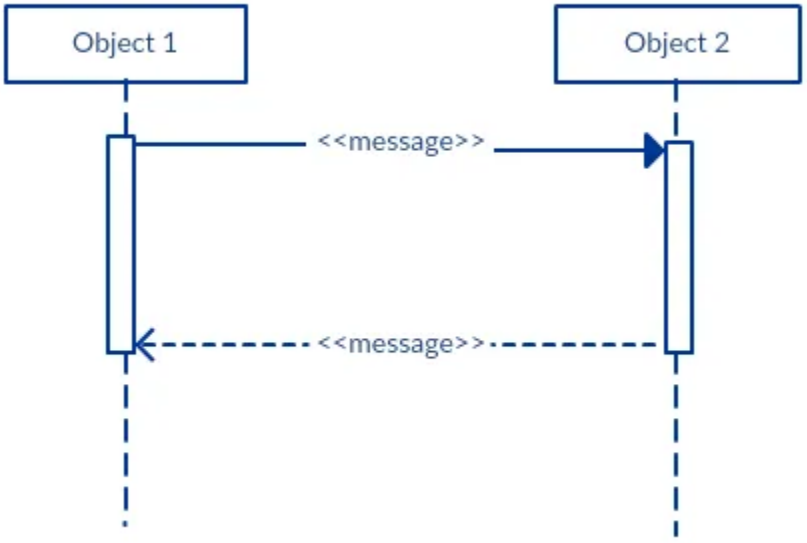

Control


Object
Object
<<message>>
Message Caller
Activation Bar
Message
Receiver
Activation Bar

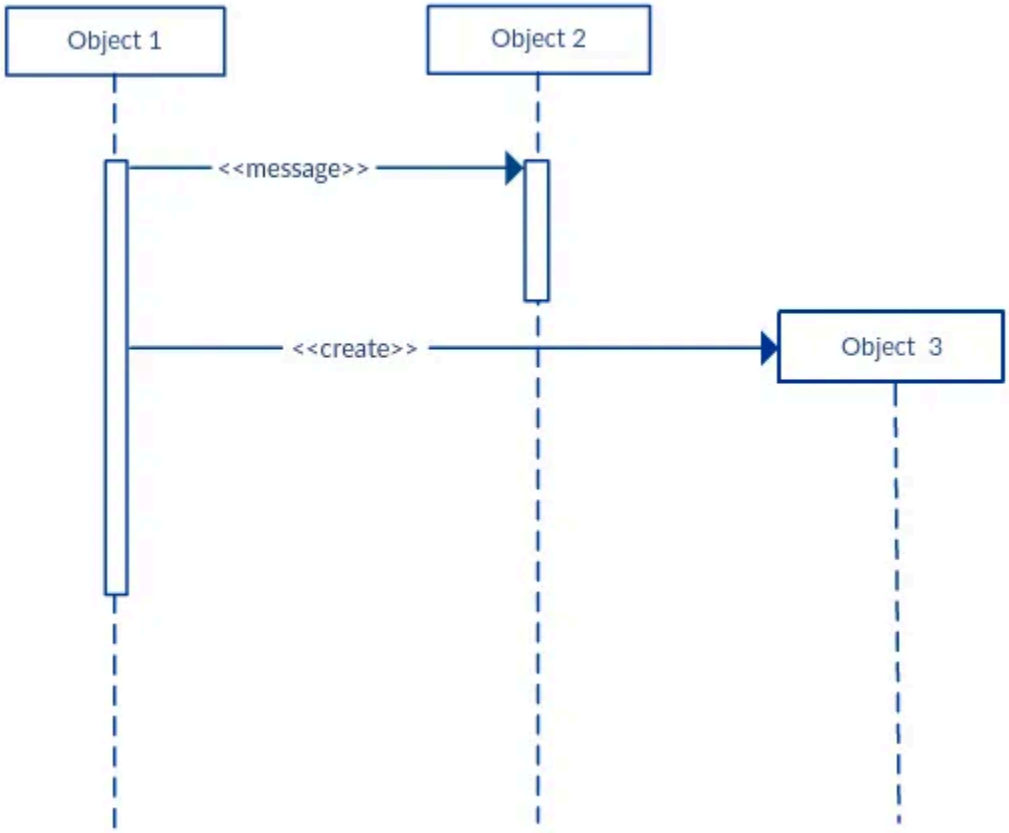
Message


Object 1
Object 2
Object 3
<<message>>
<< asynchronousMessage>>
<<asynchronousMessage>>

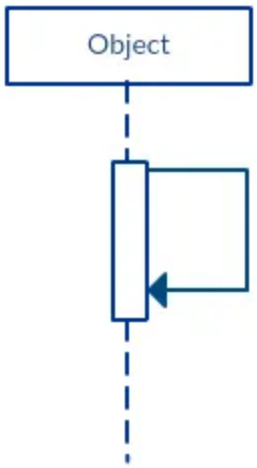
Return Message



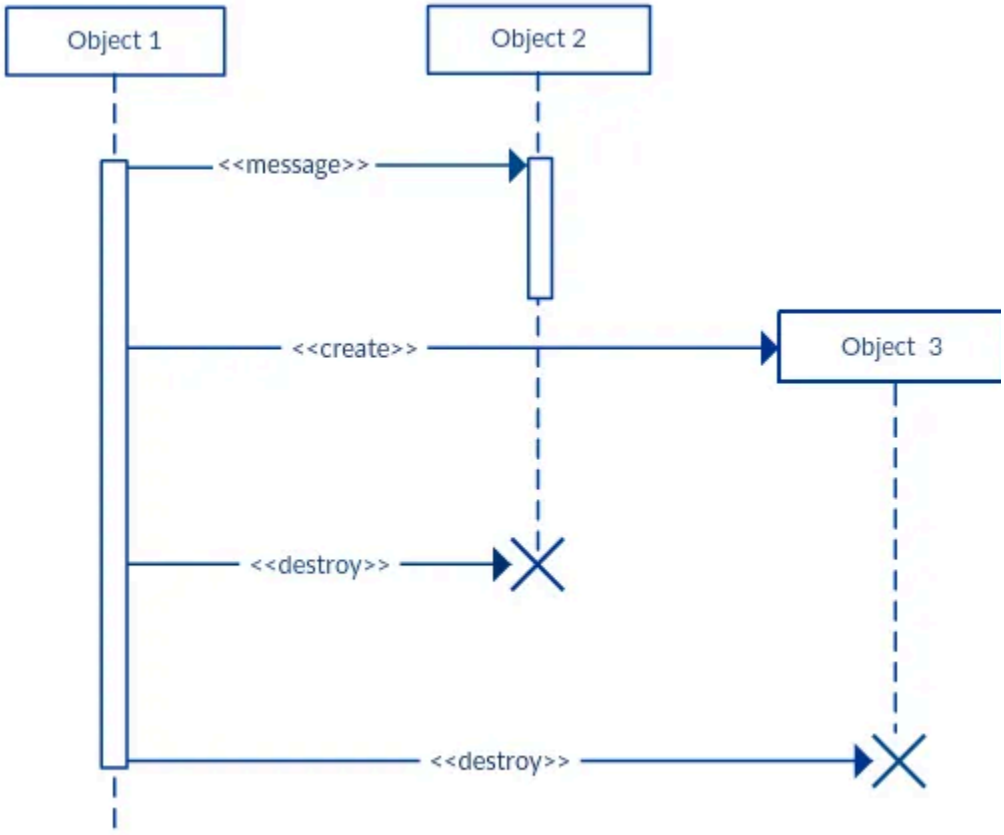
Create message



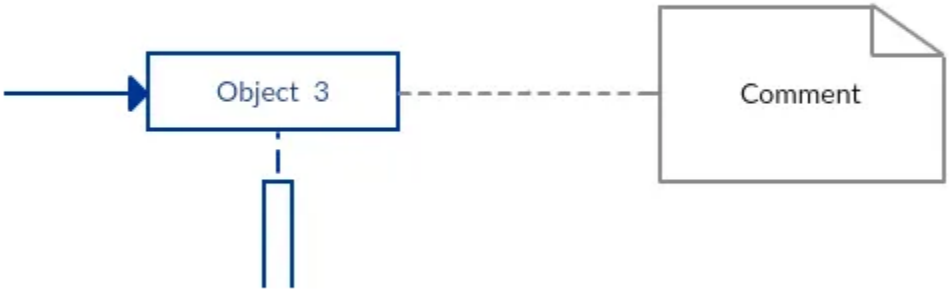
Message Destruction

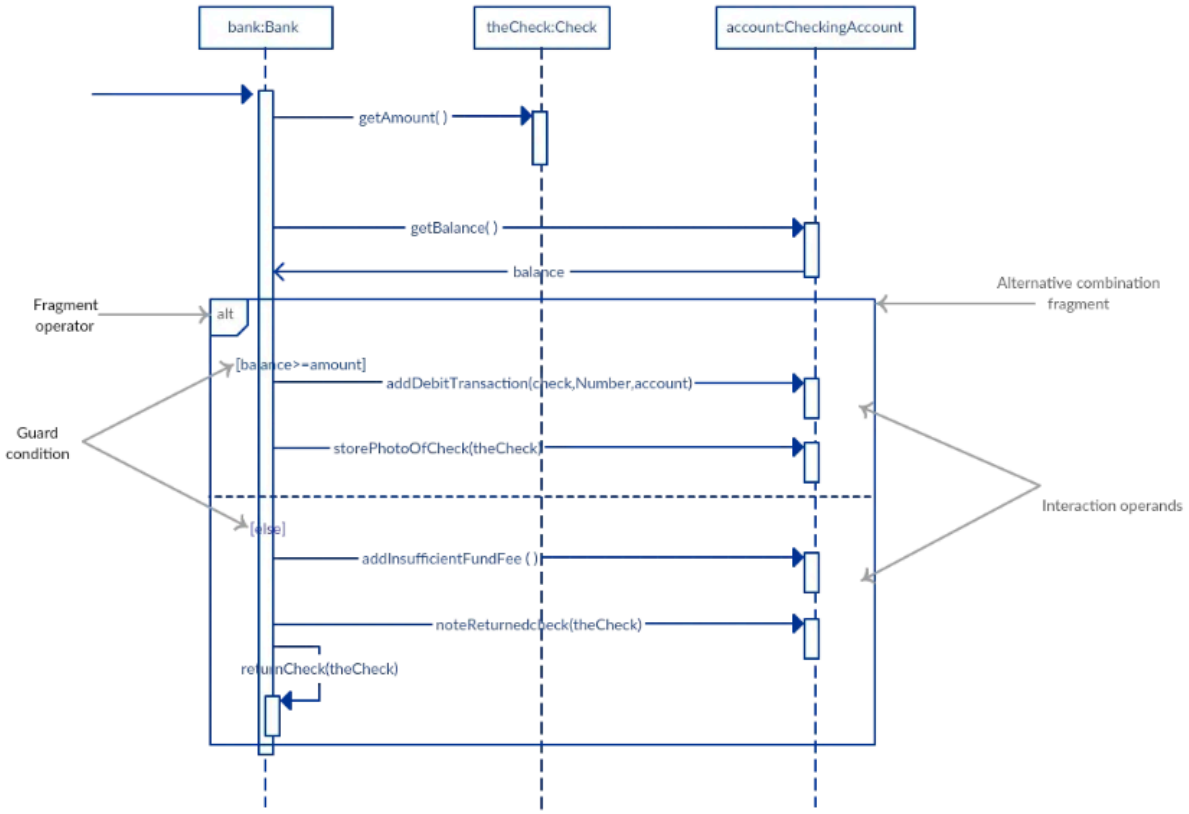


Reflexive message : when object sends message to itself

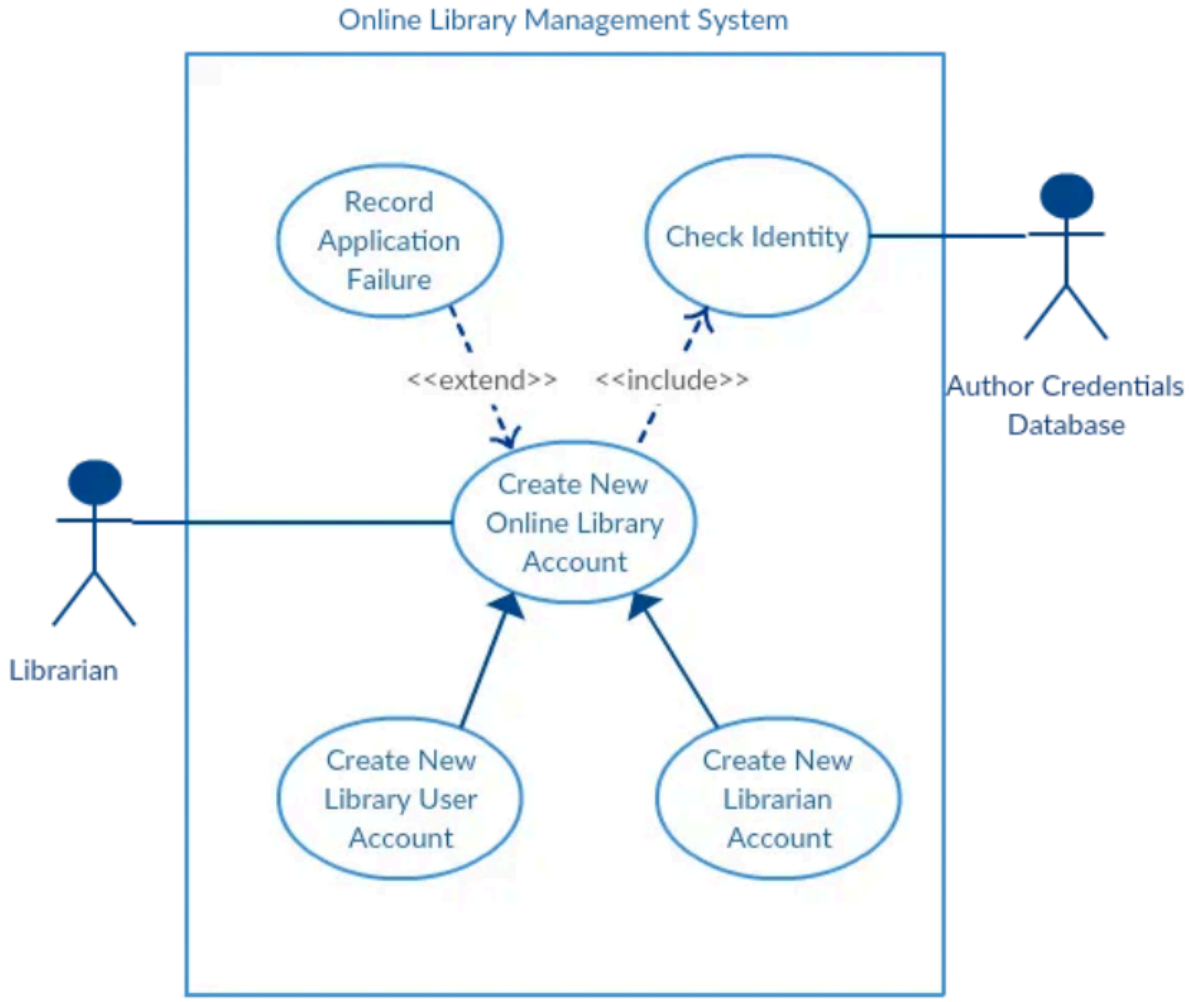


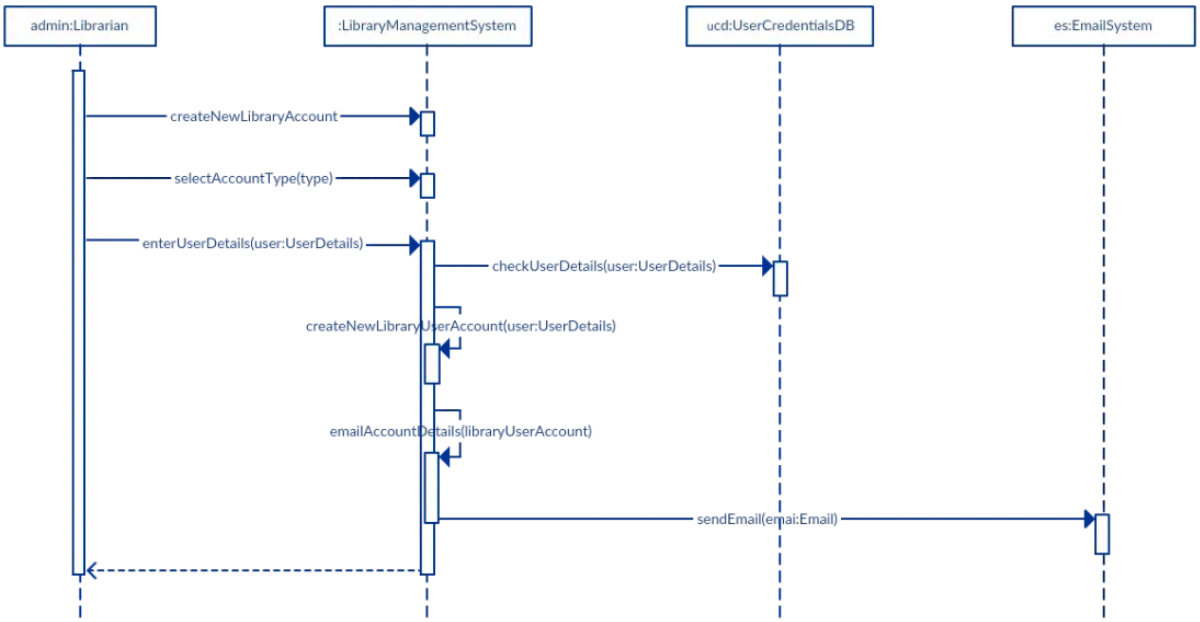
Comments


Object 3
Comment

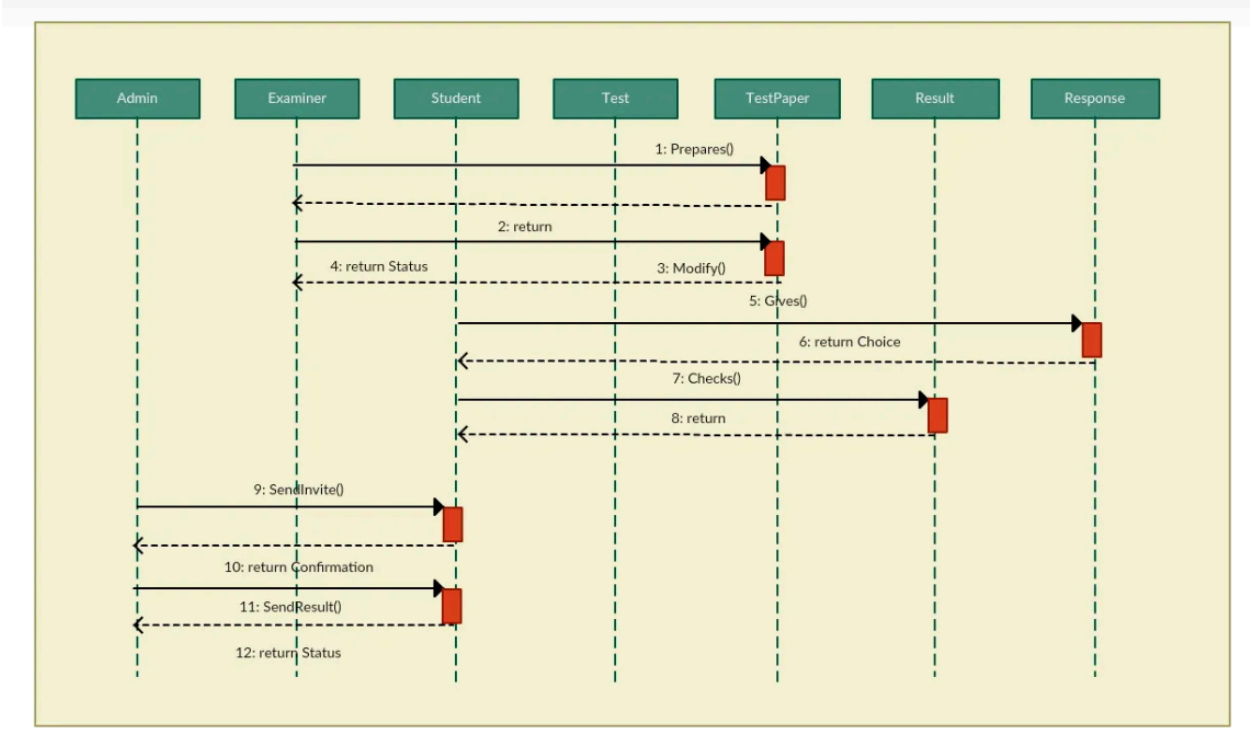

bank:Bank
theCheck:Check
account:CheckingAccount
getAmount( )
getBalance( )
balance
Fragment operator
alt
Alternative combination fragment
[balance>=amount]
addDebitTransaction(check,Number,account)
Guard condition
storePhotoOfCheck(theCheck)
Interaction operands
[else]
addInsufficientFundFee ( )
noteReturnedCheck(theCheck)
returnCheck(theCheck)

Creating a Sequence diagram from use case diagram

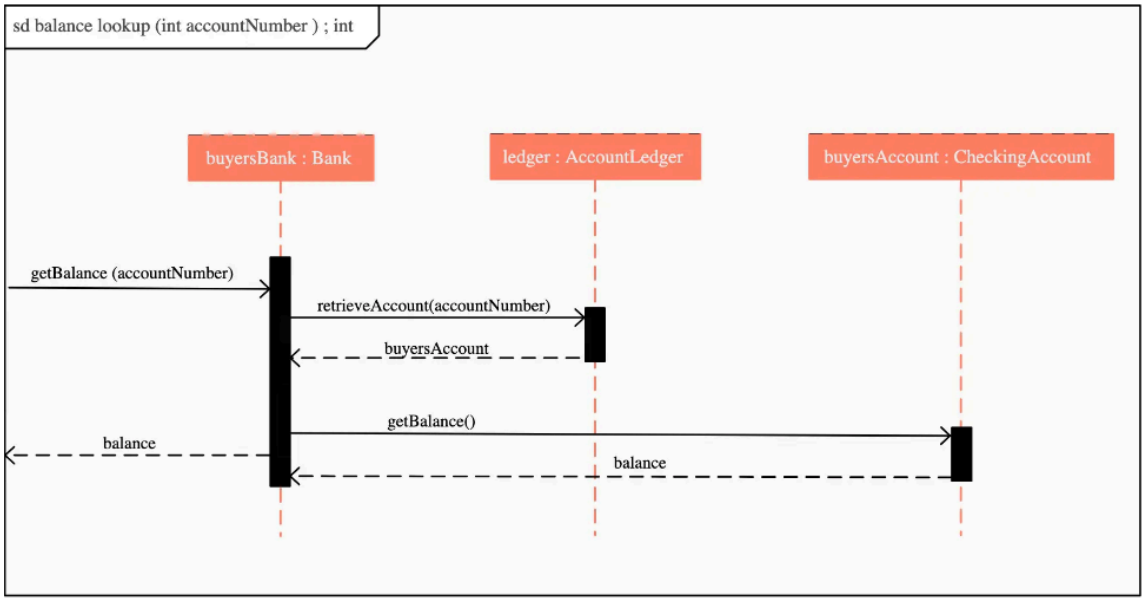




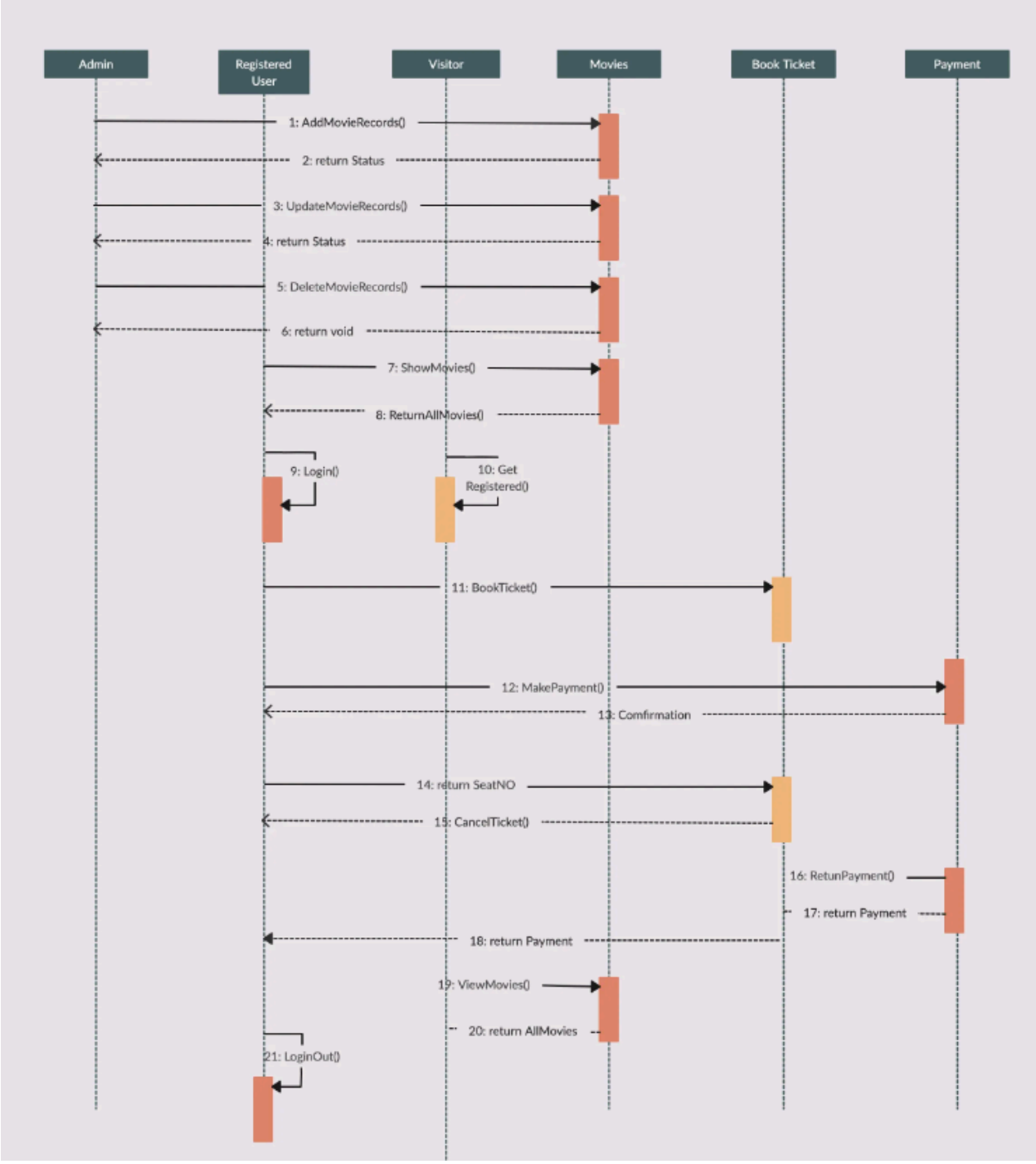
Sequence Diagram for Online Exam



Balance Lookup



Movie Ticket Booking System



END