1.)*Requirements engineering*

1. Requirements engineering is the term for the broad spectrum of tasks and techniques that lead to an understanding of requirements.
2. The process of finding out, analyzing, documenting and checking these services and constraints is called requirements engineering (RE)

- The requirements for a system are the descriptions of the services that a system should provide and the constraints on its operation.
- These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information.
- From a software process perspective, requirements engineering is a major software engineering action that begins during the communication activity and continues into the modeling activity.
- Requirements engineering establishes a solid base for design and construction.
- Without it, the resulting software has a high probability of not meeting customer's needs.
- It must be adapted to the needs of :
    - the process,
    - the project,
    - the product,
    - the people doing the work.
- It is important to realize that each of these tasks is done iteratively as the project team and the stakeholders continue to share information about their respective concerns.
- Requirements engineering builds a bridge to design and construction.
- But where does the bridge originate? One could argue that it begins with the project stakeholders (e.g., managers, customers, and end users), where business needs are defined, user scenarios are described, functions and features are delineated, and project constraints are identified. Others might suggest that it begins with a broader system definition, where software is but one component of the larger system domain. But regardless of the starting point, the journey across the bridge takes you high above the project, allowing you to examine the context of the software work to be performed; the specific needs that design and construction must address; the priorities that guide the order in which work is to be completed; and the information, functions, and behaviors that will have a profound impact on the resulting design.
- Requirements engineering encompasses seven tasks with sometimes muddy boundaries:
    1. Inception
    2. Elicitation
    3. Elaboration
    4. Negotiation
    5. Specification
    6. Validation
    7. Management
- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.
- Expect to do a bit of design during requirements work and a bit of requirements work during design.

2.)*Inception*

- Most projects begin with an identified business need or when a potential new market or service is discovered.
- At project inception, you establish a basic understanding of:
  - the problem,
  - the people who want a solution,
  - the nature of the solution that is desired.
- Communication between all stakeholders and the software team needs to be established during this task to begin an effective collaboration

3.)*Elicitation*

- It certainly seems simple enough—ask the customer, the users, and others what the objectives for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day-to-day basis. But it isn't simple—it's very hard.
- An important part of elicitation is to understand the business goals
- A goal is a long-term aim that a system or product must achieve.
- Goals may deal with either functional or nonfunctional (e.g., reliability, security, usability) concerns
- Goals are often a good way to explain requirements to stakeholders and, once established, can be used to manage conflicts among stakeholders.
- Goals should be specified precisely and serve as the basis for:
  - requirements elaboration,
  - verification and validation,
  - conflict management,
  - negotiation,
  - explanation,
  - evolution.
- Your job is to engage stakeholders and to encourage them to share their goals honestly.
- Once the goals are captured, you establish a prioritization mechanism and create a design rationale for a potential architecture (that meets stakeholder goals).
- Agility is an important aspect of requirements engineering.
- The intent of elicitation is to transfer ideas from stakeholders to the software team smoothly and without delay.
- It is highly likely that new requirements will continue to emerge as iterative product development occurs.

4.)*Elaboration*

- The elaboration task focuses on developing a refined requirements model that identifies various aspects of software function, behavior, and information
- Elaboration is driven by the creation and refinement of user scenarios obtained
- These scenarios describe how the end users (and other actors) will interact wi the system.
- Each user scenario is parsed to extract analysis classes.
- The attributes of each analysis class are defined, and the services that are required by each class are identified.
- The relationships and collaboration between classes are identified.
- Elaboration is a good thing, but you need to know when to stop. The key is to describe the problem in a way that establishes a firm base for design and then move on. Do not obsess over unnecessary details.

analysis classes—business domain entities that are visible to the end user

5.)Negotiation

- It isn't unusual for customers and users to ask for more than can be achieved, given limited business resources.
- It's also relatively common for different customers or users to propose conflicting requirements, arguing that their version is "essential for our special needs."
- These conflicts need to be reconciled through the process of negotiation.
- Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority.
- There should be no winner and no loser in an effective negotiation.
- Both sides win, because a "deal" that both can live with is solidified.
- You should use an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts.
- In this way, requirements are eliminated, combined, and/or modified so that each party achieves some measure of satisfaction.

6.)Specification

- In the context of computer-based systems (and software), the term specification means different things to different people.
- A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.
- Some suggest that a "standard template" should be developed and used for a specification, arguing that this leads to requirements that are presented in a consistent and therefore more understandable manner.
- However, it is sometimes necessary to remain flexible when a specification is to be developed.
- The formality and format of a specification varies with the size and the complexity of the software to be built.
- For large systems, a written document, combining natural language descriptions and graphical models, may be the best approach.
- However, usage scenarios may be all that are required for smaller products or systems that reside within well-understood technical environments.

7.)Validation

- The work products produced during requirements engineering are assessed for quality during a validation step.
- A key concern during requirements validation is consistency.
- Use the analysis model to ensure that requirements have been consistently stated.
- Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product.
- The primary requirements validation mechanism is the technical review.
- The review team that validates requirements includes software engineers, customers, users, and other stakeholders who examine the specification looking for errors in content or interpretation, areas where clarification may be required, missing information, inconsistencies (a major problem when large products or systems are engineered), conflicting requirements, or unrealistic (unachievable) requirements.

- To illustrate some of the problems that occur during requirements validation, consider two seemingly innocuous requirements: · The software should be user friendly. The probability of a successful unauthorized database intrusion should be less than 0.0001. The first requirement is too vague for developers to test or assess. What exactly does “user friendly” mean? To validate it, it must be quantified or qualified in some manner. The second requirement has a quantitative element (“less than 0.0001”), but intrusion testing will be difficult and time consuming. Is this level of security even warranted for the application? Can other complementary requirements associated with security (e.g., password protection, specialized handshaking) replace the quantitative requirement noted?

Requirements Management

- Requirements for computer-based systems change, and the desire to change requirements persists throughout the life of the system.
- Requirements management is a set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds. Many of these activities are identical to the software configuration management (SCM) techniques.

1.Establishing the Groundwork

- In an ideal setting, stakeholders and software engineers work together on the same team.
- In such cases, requirements engineering is simply a matter of conducting meaningful conversations with colleagues who are well-known members of the team. But reality is often quite different.
- Customer(s) or end users may reside in different cities or countries, may have only a vague idea of what is required, may have conflicting opinions about the system to be built, may have limited technical knowledge, and may have limited time to interact with the requirements engineer.
- None of these things are desirable, but all are common, and you are often forced to work within the constraints imposed by this situation. In the sections that follow, we discuss the steps required to establish the groundwork for an understanding of software requirements—to get the project started in a way that will keep it moving forward toward a successful solution

2. Identifying Stakeholders

- Sommerville and Sawyer [Som97] define a stakeholder as “anyone who benefits in a direct or indirect way from the system which is being developed.”
- We have already identified the usual suspects: business operations managers, product managers, marketing people, internal and external customers, end users, consultants, product engineers, software engineers, support and maintenance engineers, and others.
- Each stakeholder has a different view of the system, achieves different benefits when the system is successfully developed, and is open to different risks if the development effort should fail.
- At inception, you should create a list of people who will contribute input as requirements are elicited.
- The initial list will grow as stakeholders are contacted because every stakeholder will be asked: “Whom else do you think I should talk to?”

8.)Recognizing Multiple Viewpoints

- Because many different stakeholders exist, the requirements of the system will be explored from many different points of view.
- For example, the marketing group is interested in features that will excite the potential market, making the new system easy to sell.

- Business managers are interested in a feature set that can be built within budget and that will be ready to meet defined market windows.
- End users may want features that are familiar to them and that are easy to learn and use.
- Software engineers may be concerned with functions that are invisible to nontechnical stakeholders but that enable an infrastructure that supports more marketable functions and features.
- Support engineers may focus on the maintainability of the software.
- Each of these constituencies (and others) will contribute information to the requirements engineering process.
- As information from multiple viewpoints is collected, emerging requirements may be inconsistent or may conflict with one another.
- You should categorize all stakeholder information (including inconsistent and conflicting requirements) in a way that will allow decision makers to choose an internally consistent set of requirements for the system.
- Several things can make it hard to elicit requirements for software that satisfies its users: project goals are unclear, stakeholders' priorities differ, people have unspoken assumptions, stakeholders interpret meanings differently, and requirements are stated in a way that makes them difficult to verify.
- The goal of effective requirements engineering is to eliminate or at least reduce these problems

Working Toward Collaboration

- The job of a requirements engineer is to identify areas of commonality (i.e., requirements on which all stakeholders agree) and areas of conflict or inconsistency (i.e., requirements that are desired by one stakeholder but conflict with the needs of another stakeholder).
- In many cases, stakeholders collaborate by providing their view of requirements, but a strong "project champion" (e.g., a business manager or a senior technologist) may make the final decision about which requirements make the cut

Nonfunctional Requirements

- A nonfunctional requirement (NFR) can be described as a quality attribute, a performance attribute, a security attribute, or a general constraint on a system.
- It is possible to define a two-phase approach that can assist a software team and other stakeholders in identifying nonfunctional requirements.
- During the first phase, a set of software engineering guidelines is established for the system to be built.
- These include guidelines for best practice, but also address architectural style and the use of design patterns.
- A list of NFRs (e.g., requirements that address usability, testability, security, or maintainability) is then developed.
- A simple table lists NFRs as column labels and software engineering guidelines as row labels. A relationship matrix compares each guideline to all others, helping the team to assess whether each pair of guidelines is complementary, overlapping, conflicting, or independent. In the second phase, the team prioritizes each nonfunctional requirement by creating a homogeneous set of nonfunctional requirements using a set of decision rules that establish which guidelines to implement and which to reject.

Traceability:

- Traceability is a software engineering term that refers to documented links between software engineering work products (e.g., requirements and test cases).
- A traceability matrix allows a requirements engineer to represent the relationship between requirements and other software engineering work products.
- Rows of the traceability matrix are labeled using requirement names, and columns can be labeled with the name of a software engineering work product (e.g., a design element or a test case).
- A matrix cell is marked to indicate the presence of a link between the two.
- The traceability matrices can support a variety of engineering development activities.
- They can provide continuity for developers as a project moves from one project phase to another, regardless of the process model being used.
- Traceability matrices often can be used to ensure the engineering work products have taken all requirements into account.
- As the number of requirements and the number of work products grows, it becomes increasingly difficult to keep the traceability matrix up to date.
- Nonetheless, it is important to create some means for tracking the impact and evolution of the product requirements.

Slides

Requirements:

Standish (1995) asked the survey respondents to explain the causes of the failed projects. The top factors were reported to be

1. Incomplete requirements (13.1%)
2. Lack of user involvement (12.4%)
3. Lack of resources (10.6%)
4. Unrealistic expectations (9.9%)
5. Lack of executive support (9.3%)
6. Changing requirements and specifications (8.7%)
7. Lack of planning (8.1%)
8. System no longer needed (7.5%)

Requirements Engineering:

- The requirements for a system are the descriptions of the services that a system should provide and the constraints on its operation.
- These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information.

The process of finding out, analyzing, documenting and checking these services and constraints is called requirements engineering (RE)

User Requirements	System Requirements
User requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.	System requirements are more detailed descriptions of the software system's functions, services, and operational constraints.
The user requirements may vary from broad statements of the system features required to detailed, precise descriptions of the system functionality.	The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented.

User requirements definition

1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

User Stories:

As a <type of user>, I want <some goal>, so that <some reason>

1. As a forgetful user, I want a straightforward way to reset my password in case I forget it.
2. As an online gamer, I want to have a multiplayer option so that I can play online with friends.
3. As a design team lead, I want to organize assets, so I can keep track of multiple creative projects.
4. As an e-commerce shopper, I want to filter my searches so I can find products quickly.
5. As a trainer, I want my profile to list my upcoming classes and include a link to a detailed page about each so that prospective attendees can find my courses.
6. As a site visitor, I want to access old news that is no longer on the home page, so I can access things I remember from the past

Requirements Engineering Tasks:

1. Inception: understand problem, people, nature of solution, effectiveness of communication.
2. Elicitation: Ask questions about objectives, targets, detailed requirements
3. Elaboration: Identify software function, behavior, information. Develop Requirements Model!!! Analysis???
4. Negotiation: Resolve conflicts. Prioritize Requirements!!!
5. Specification: Write document containing requirements models, scenarios etc.
6. Validation: Ensure that all requirements have been stated, unambiguously!
7. Management: Identify, control, track requirements and changes

Inception:

Identify: all stakeholders, measurable benefits of successful implementation, possible alternatives

Ask questions stepwise, as early as possible, first meeting/encounter

Possible questions at 1st step (stakeholders, overall goals and benefits):

1. Who is behind the request for this work?
2. Who will use this solution?
3. What will be the economic benefit of a successful solution?

Possible questions at 2nd step (detailed understanding and customer perception about the solution):

1. What problems(s) will this solution address?
2. Can you show me (or describe) the business environment in which the solution will be used?
3. How do you characterize the 'good' output?

Possible questions at 3rd step (effectiveness of communication):

1. Are you the right person to answer these questions?
2. Are my questions relevant to the problem that you have?
3. Can anyone else provide additional information?

Elicitation:

- Get more detailed requirements.
- Customers do not always understand what their needs and problems are.
- It is important to discuss the requirements with everyone who has a stake in the system.
- Come up with agreement on what the requirements are
- If we cannot agree on what the requirements are, then the project is doomed to fail

Different Stakeholders are:

1. Clients: pay for the software to be developed
2. Customers: buy the software after it is developed
3. Users: use the system
4. Domain experts: familiar with the problem that the software must automate
5. Market Researchers: conduct surveys to determine future trends and potential customers
6. Lawyers or auditors: familiar with government, safety, or legal requirements
7. Software engineers or other technology experts

Elaboration:

• Analyze, model, specify…

Some Analysis Techniques:

1. Data Flow Diagrams (DFD)
2. Use case Diagram
3. Object Models (ER Diagrams)
4. State Diagrams
5. Sequence Diagrams
6. Activity Diagrams

Negotiation:

Specification:

Validation:

Management:

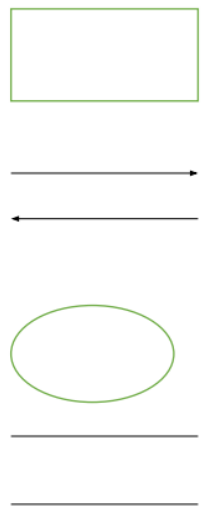
Requirements Modeling:

1. Scenario-based Models
    1. User Stories
    2. Use Case Diagram
2. Class-oriented Models
    3. Class Diagram
    4. CRC Cards
3. Behavioral Models
    5. State Diagram
    6. Sequence Diagram
4. Flow-oriented Models
    7. Data Flow Diagram

Data Flow Diagram:

- A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system.
- By drawing a Data Flow Diagram, you can tell the information provided by and delivered to someone who takes part in system processes, the information needed to complete the processes and the information needed to be stored and accessed.
- The DFD is presented in a hierarchical fashion. That is, the first data flow model (sometimes called a level 0 DFD or context diagram) represents the system as a whole.
- Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequent level.

- **Square Box:** A square box defines **external entities** (source or sink) of the system. Source supplies data to system and sink receives data from system.
- **Arrow or Line:** An arrow identifies the **data flow** i.e. it gives information to the data that is in motion. Connects processes, external entities and data stores.
- **Circle or bubble chart:** It represents a **process** that gives us information. It is also called processing box.
- **Open Rectangle:** An open rectangle is a **data store** where data is stored. Data can be read from or written to the data store



Level 0 DFD:

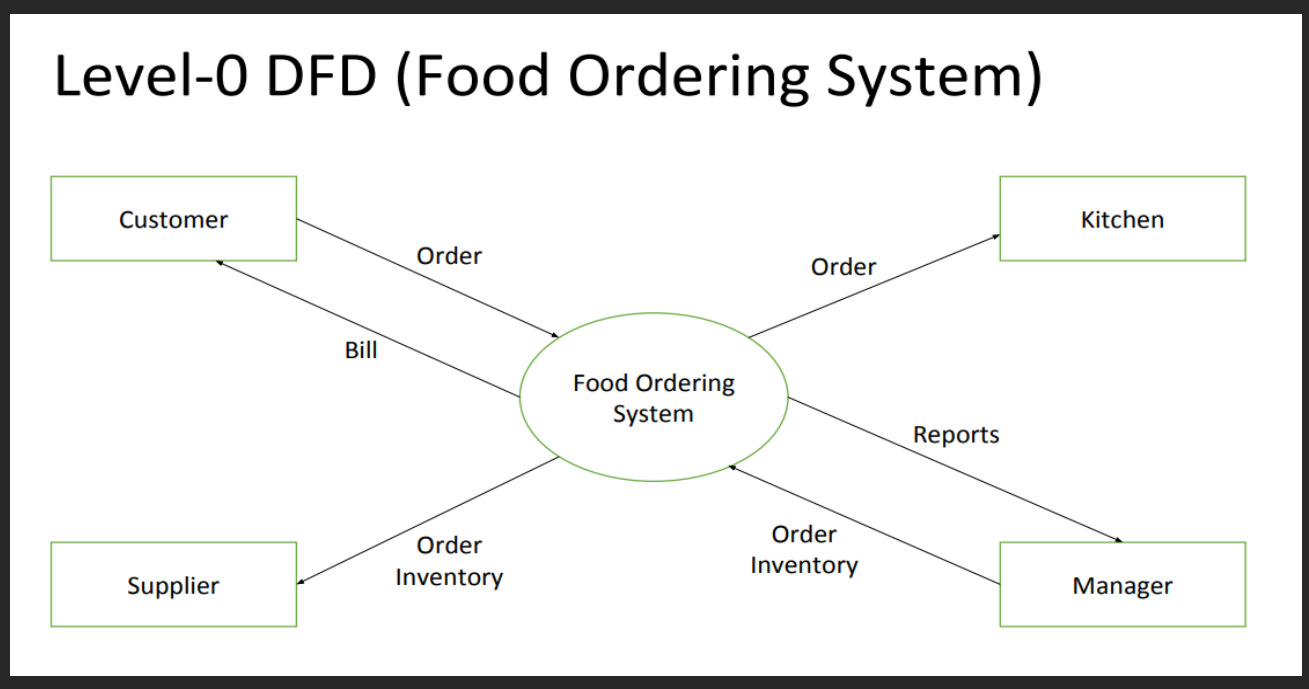
A context diagram is a data flow diagram that only shows the top level, otherwise known as Level 0.

At this level, there is only one visible process node that represents the functions of a complete system regarding how it interacts with external entities.

Context DFD is the entrance of a data flow model. It contains one and only one process and does not show any data store.

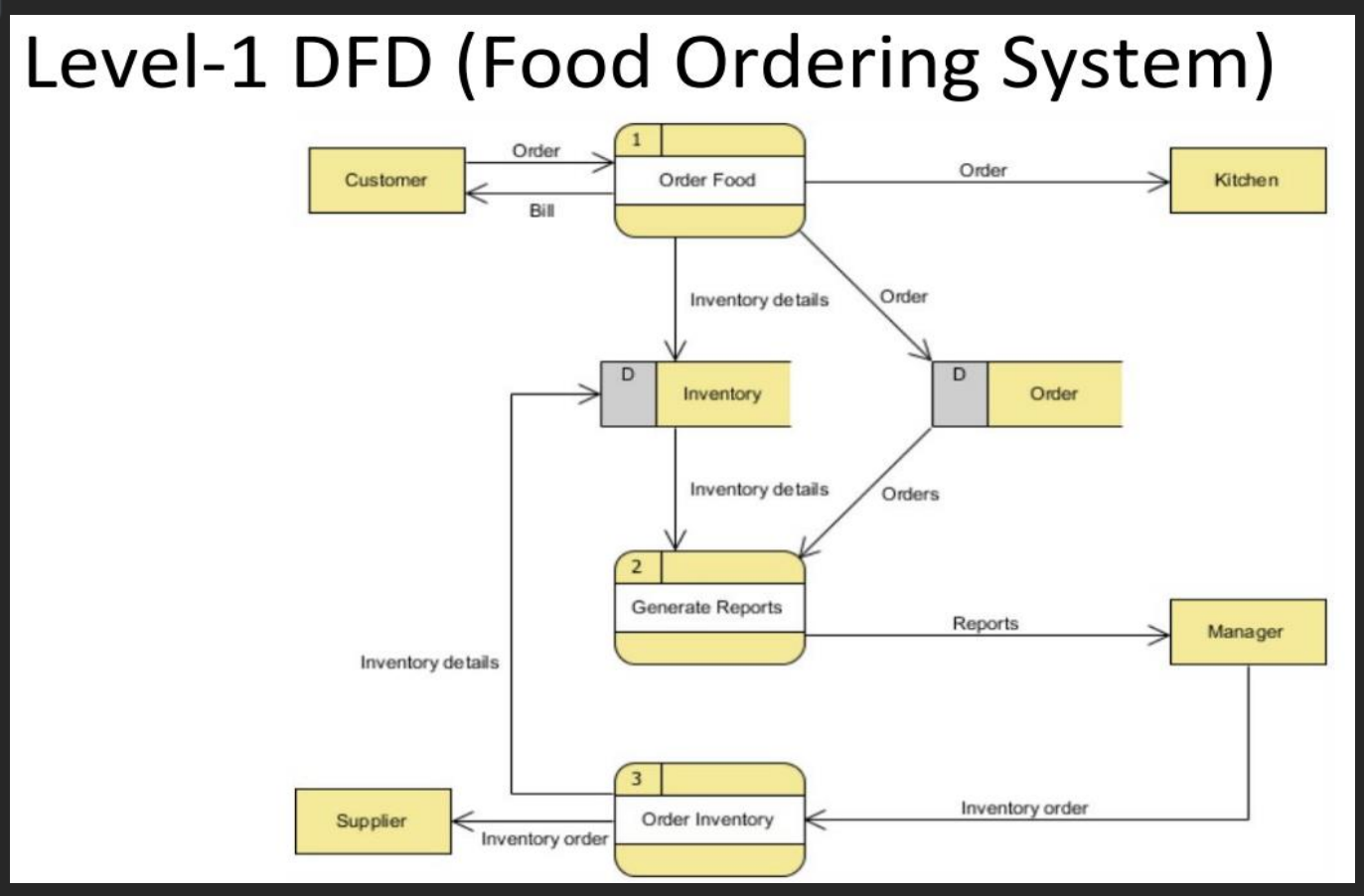
A verb in requirements narrative is potentially a process, a noun is either data, or external entity, or data store.

It contains a process that represents the system to model, in this case, the **"Food Ordering System"**.

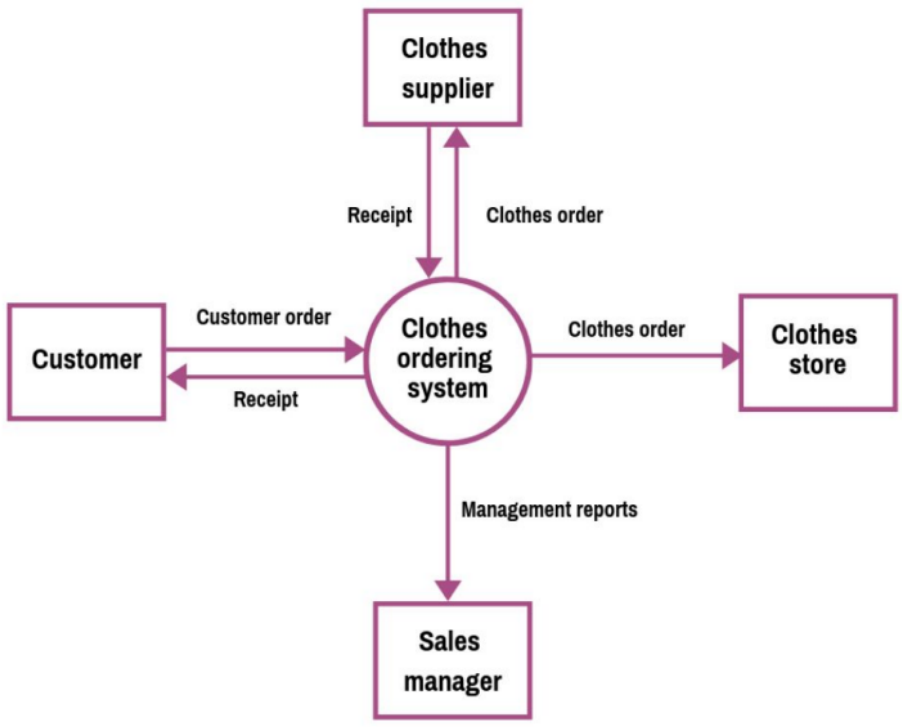


It also shows the participants who will interact with the system, called the **external entities**. In this example, the Supplier, Kitchen, Manager, and Customer are the entities who will interact with the system.

In between the process and the external entities, there is data flow (connectors) that indicate the existence of information exchange between the entities and the system.



Level-0 DFD (Clothes Ordering System)



Step 1: Define the processes.

The three processes are: Order Clothes, Generate Reports, Order Inventory.

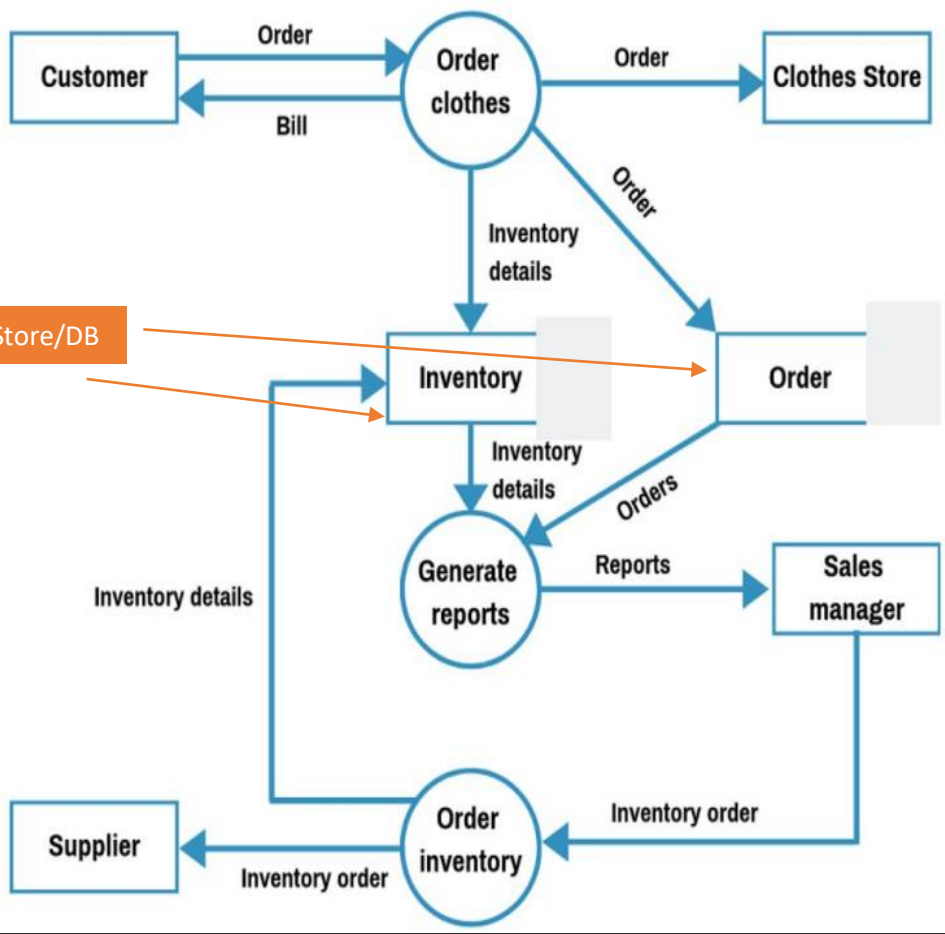
Step 2: Create the list of all external entities.

The external entities are Customer, Clothes Store, Sales Manager, and Supplier

• Step 3: Create the list of the data stores.

These are Order and Inventory

Level-1 DFD (Clothes Ordering System)



Step 4: Create the list of the data flows

Data flows are: Order, Bill, Order, Order, Inventory details, Inventory details, Orders, Reports, Inventory Order, Inventory Order, Inventory details.

Now, connect the rectangles with arrows signifying the data flows.

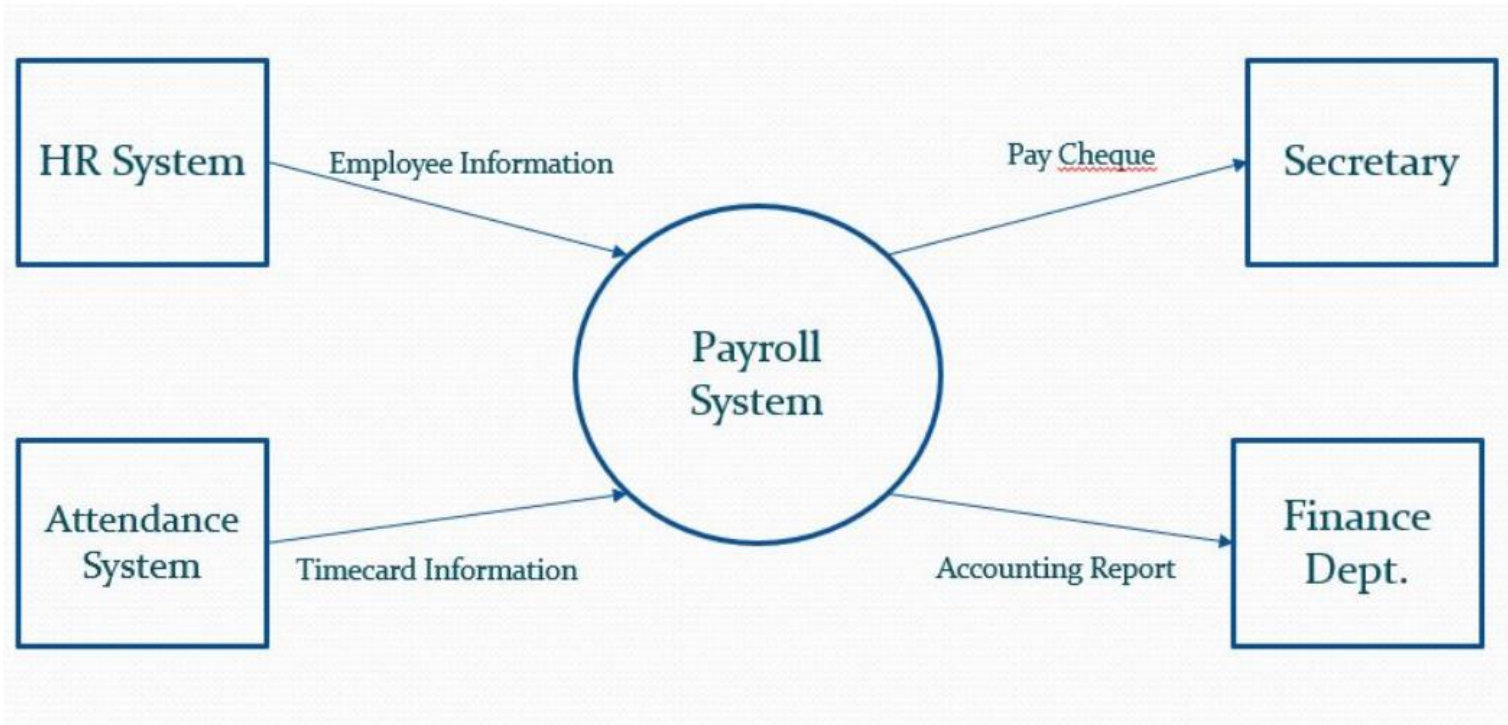
If data flows both ways between any two rectangles, create two individual arrows.

Step 5: Create the diagram.

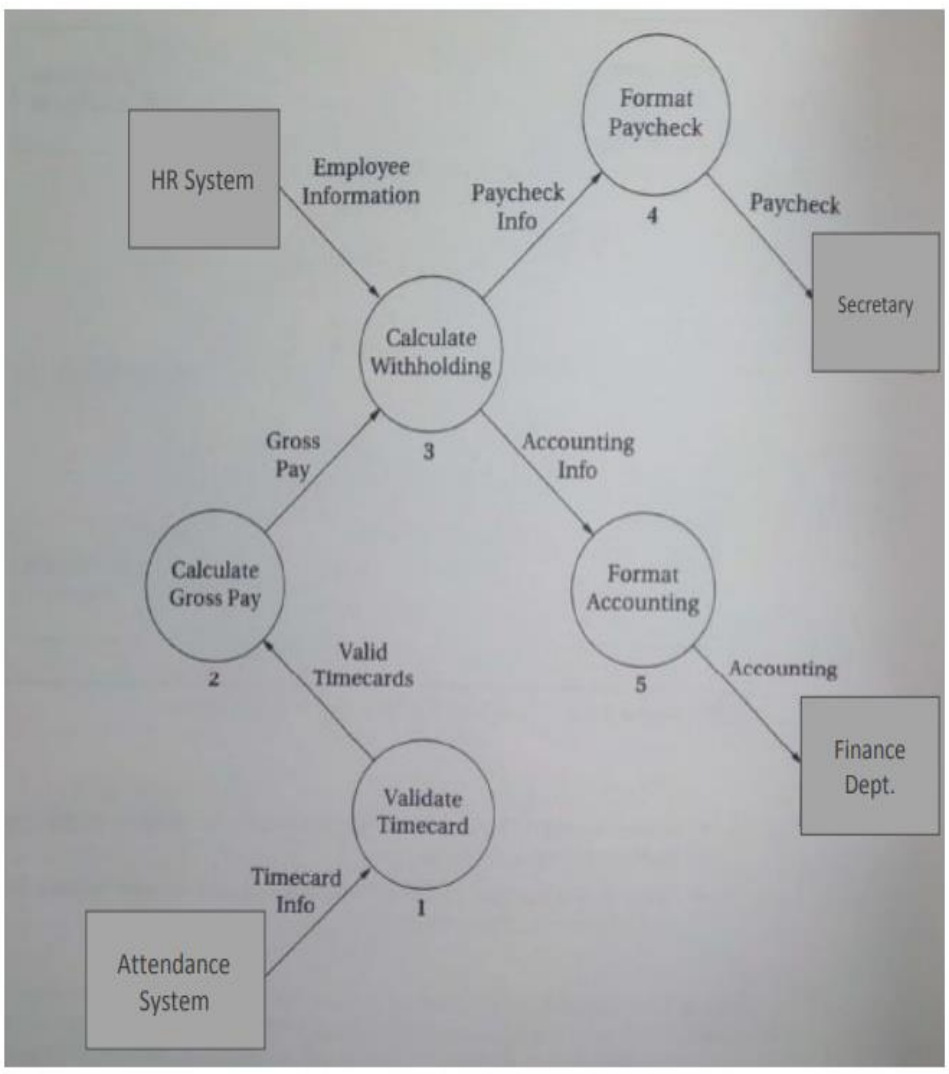
Level-0 DFD contains only one process and does not illustrate any data store. This is the main difference with level 1 DFD.

Level 1 DFD breaks down the main process into sub-processes that can then be seen on a deeper level. Also, level 1 DFD contains data stores that are used by the main process.

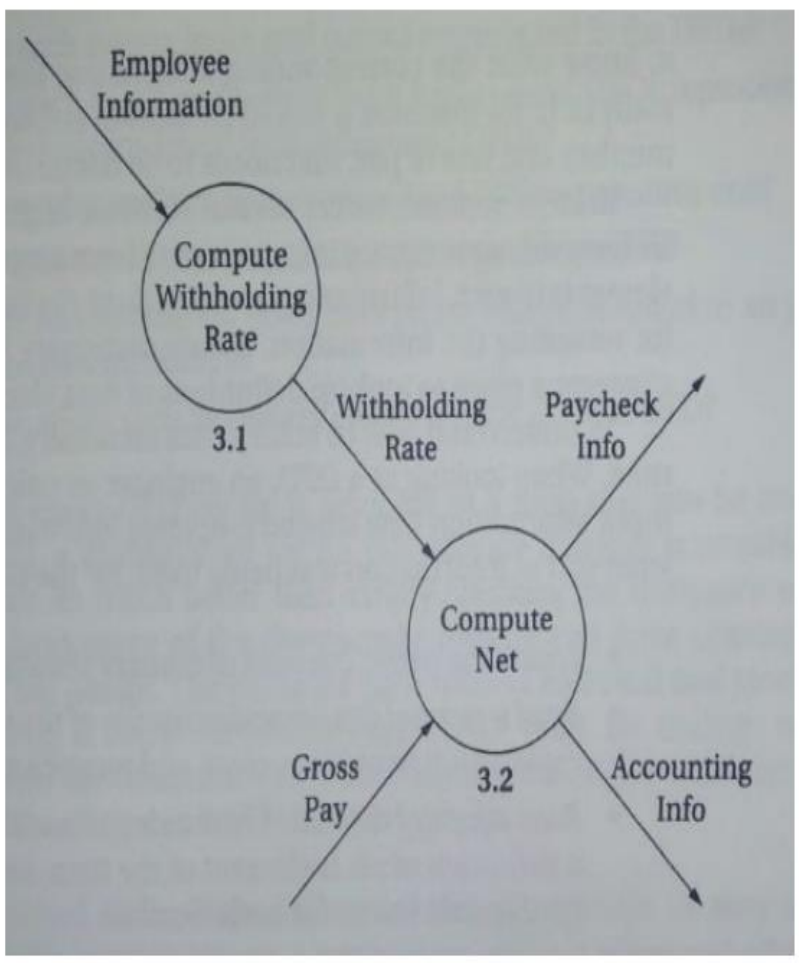
Level-0 DFD (Payroll System)



Level-1, Level-2 DFD (Payroll System)



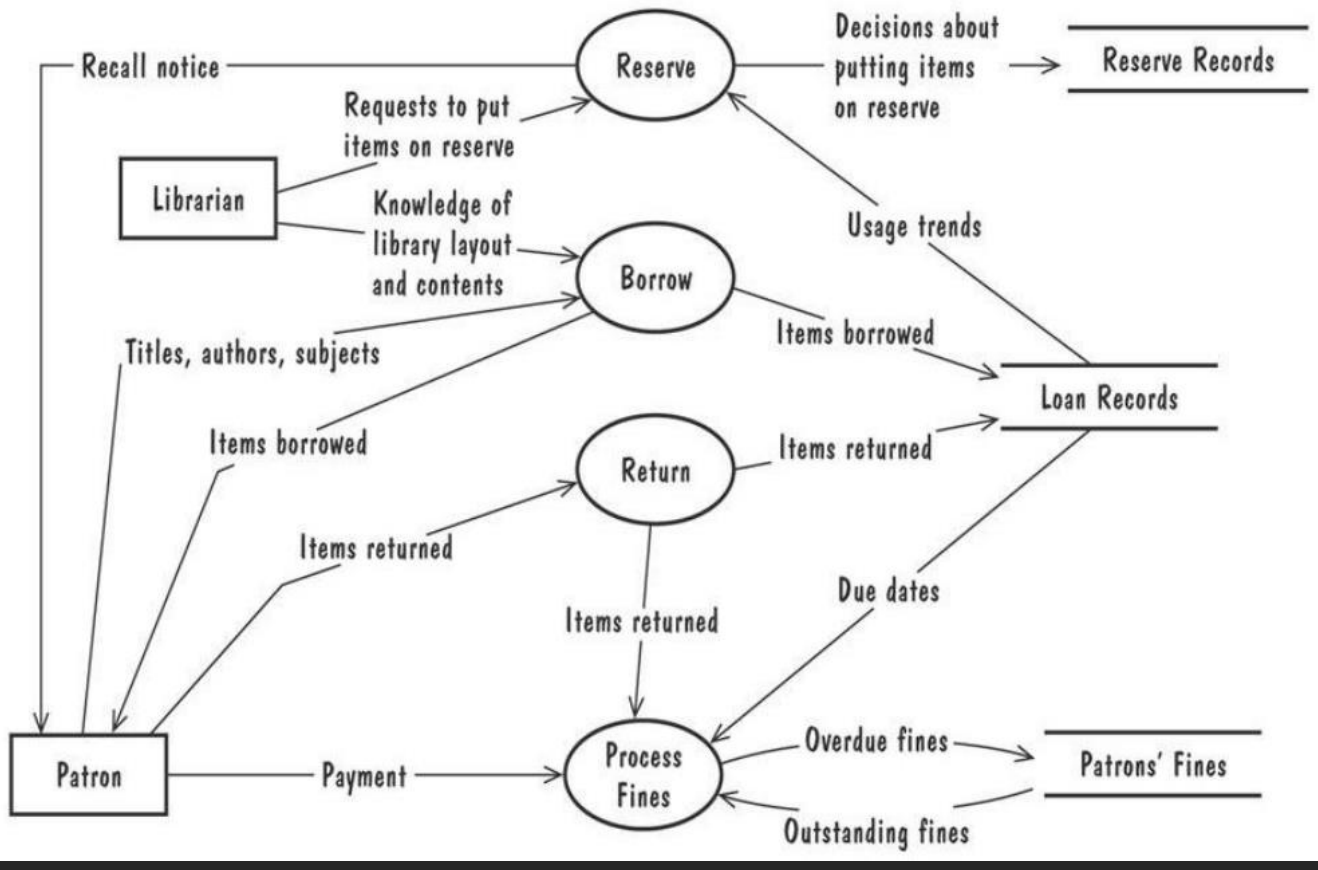
Level-1



Level-2

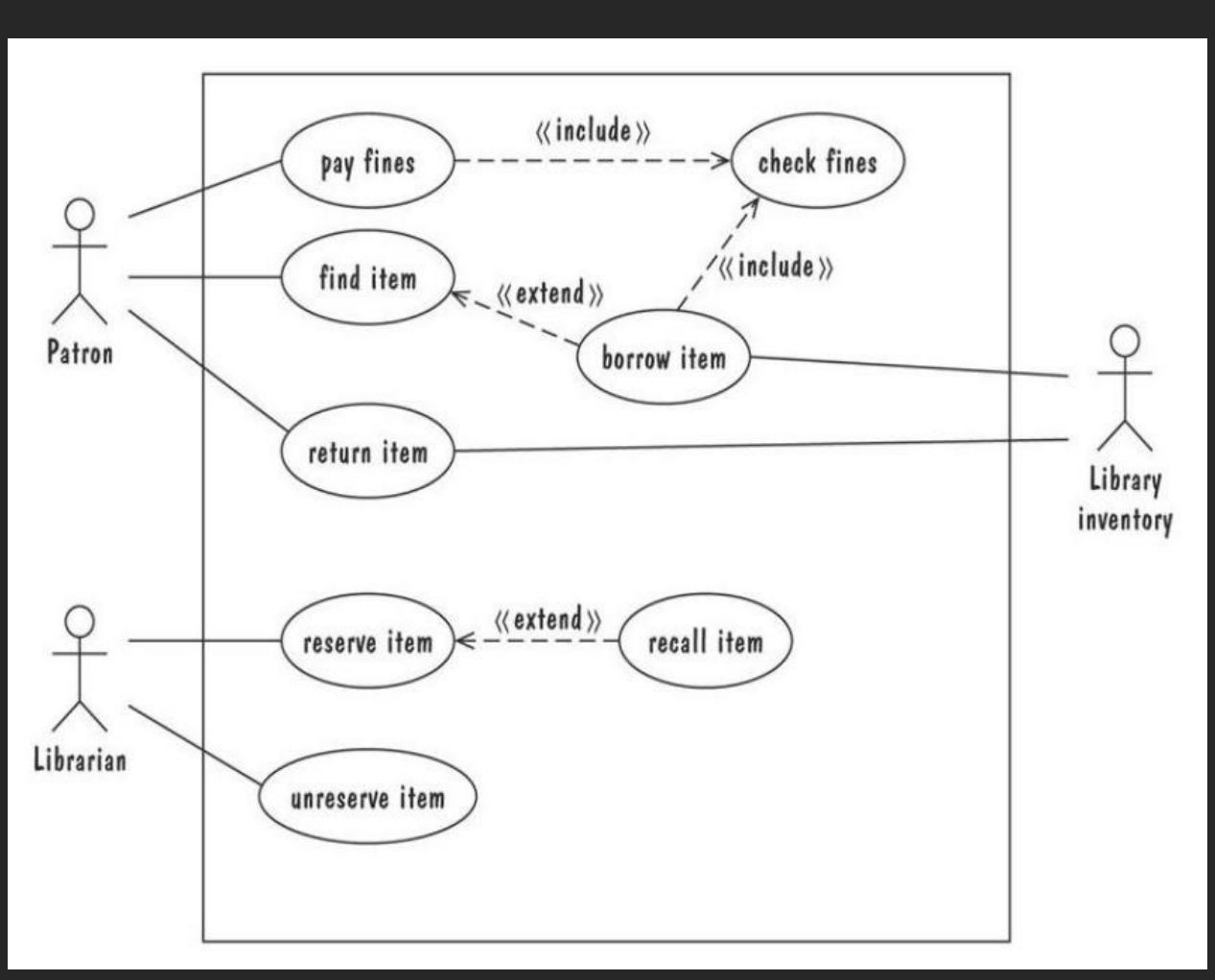
- Consider a library management system where circulation services allow the Patron to borrow or return a book/item/publication without involvement of Librarian, so the system needs to process these requests. The Librarian controls which items can be borrowed and how can the items be shown to the patrons. The system processes fines also when a borrowed item is returned after the due date. The librarian is able to reserve an item if required and a recall notice is sent to the patron as a consequence. The record of reserved items is kept so that the librarian can view which items have been reserved already.

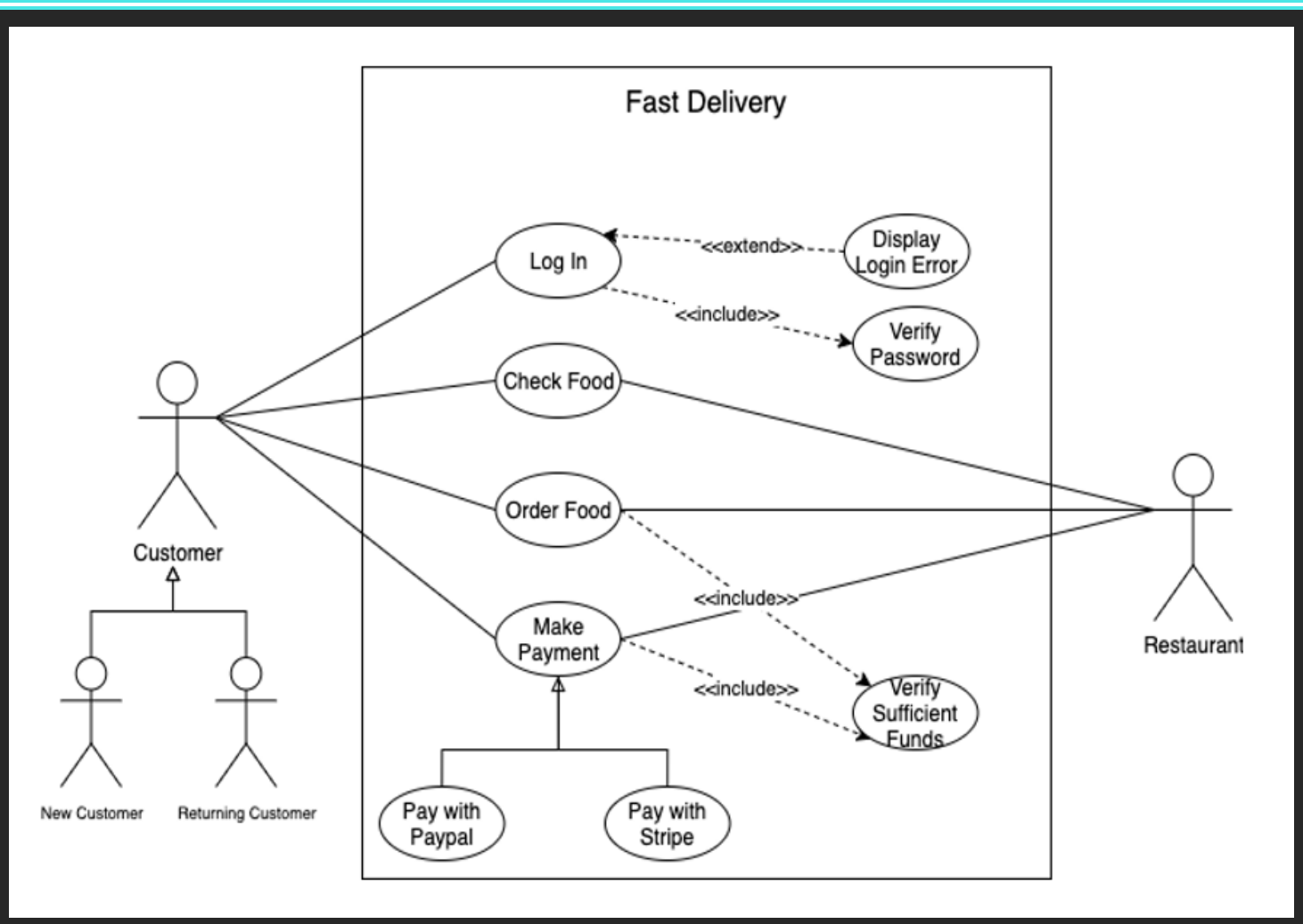
Library Management System (Level-1 DFD)

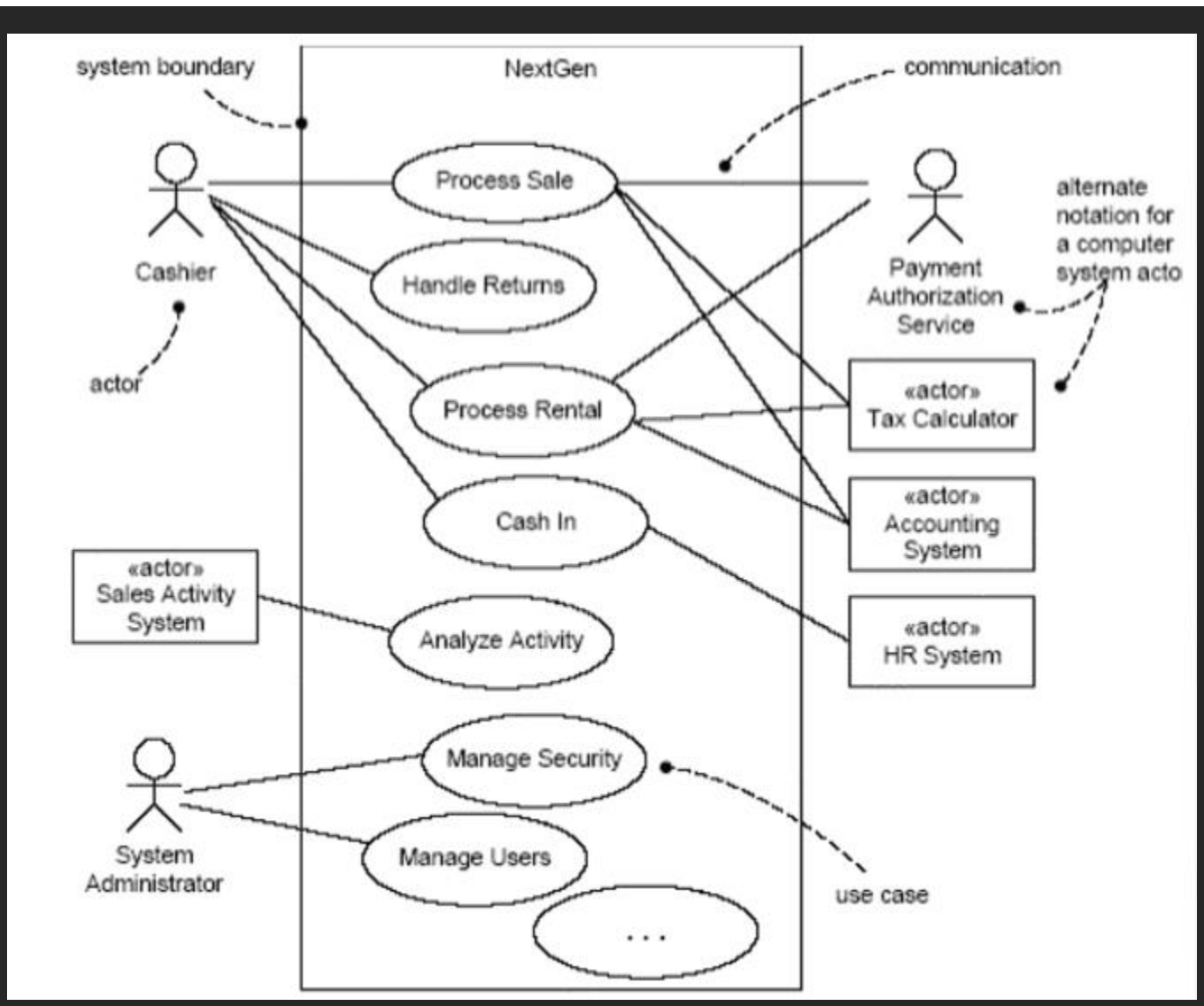


Use Case Diagrams:

Use cases do not necessarily model all the tasks, instead they are used to specify user views of essential system behavior.







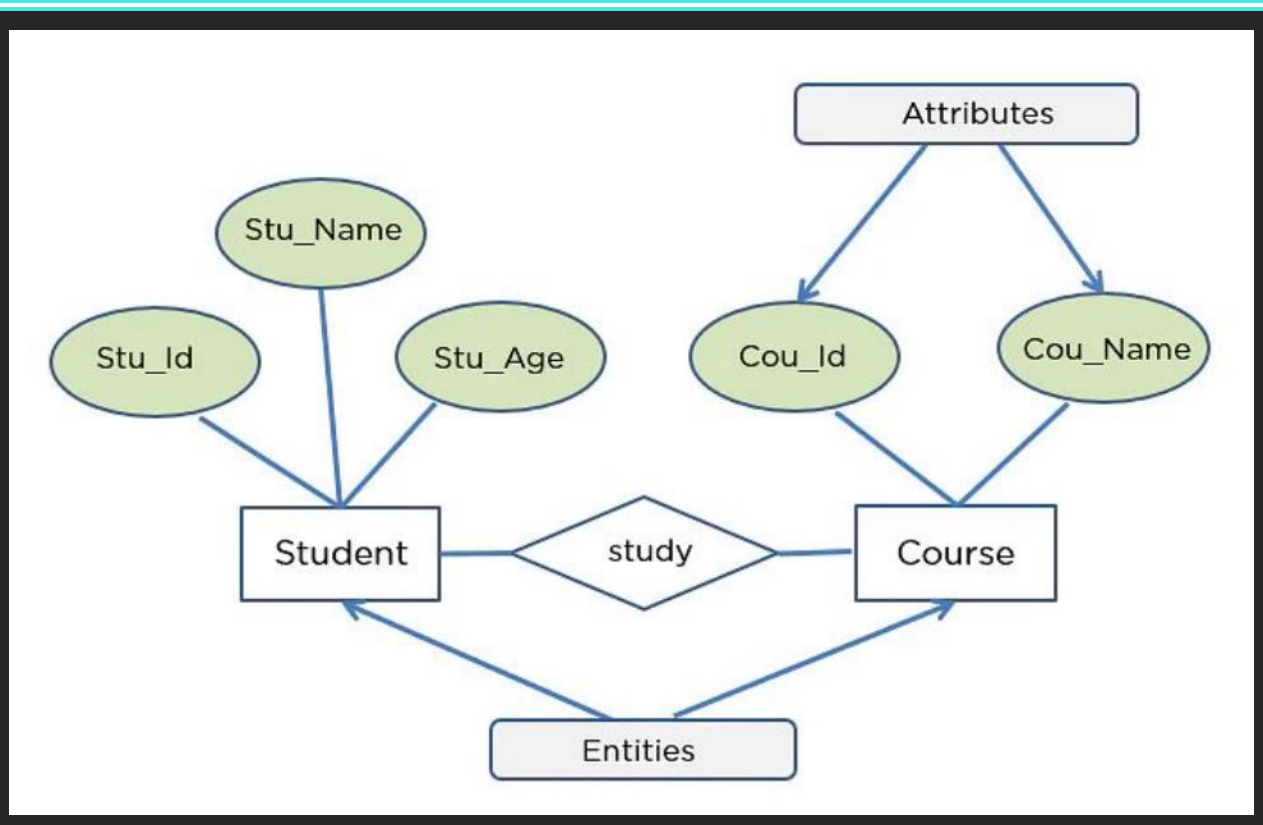
Entity Relationship Diagram:

ER Model is used to model the logical view of the system from a data perspective.

- An entity: depicted as a rectangle, represents a collection of real-world objects that have common properties and behaviors.
- A relationship: depicted as an edge between two entities, with diamond in the middle of the edge specifying the type of relationship.
- An attribute: represented as an oval, describes data or properties associated with the entity.

ER diagrams are popular because

1. they provide an overview of the problem to be addressed.
2. the view is relatively stable when changes are made to the problem's requirements.
3. ER diagram is likely to be used to model a problem early in the requirements process



Class Diagram:

Class Diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Components:

1. Objects: akin to entities, organized in classes.
2. Attributes: object's variables or characteristics.
3. Behaviours: actions on the object's variables.

Association

- Association is a semantically weak relationship (a semantic dependency) between otherwise unrelated objects.
- An association is a “using” relationship between two or more objects in which the objects have their own lifetime and there is no owner.
- As an example, imagine the relationship between a doctor and a patient. A doctor can be associated with multiple patients. At the same time, one patient can visit multiple doctors for treatment or consultation
- Each of these objects has its own life cycle and there is no “owner” or parent. The objects that are part of the association relationship can be created and destroyed independently.
- Association can be one-to-one, one-to-many, many-to-one, many-to-many.

Aggregation

- It represents a "part of" relationship.
- Aggregation is a specialized form of association between two or more objects in which each object has its own life cycle but there exists an ownership as well.
- Aggregation is a typical whole/part or parent/child relationship but it may or may not denote physical containment.

- An essential property of an aggregation relationship is that the whole or parent (i.e. the owner) can exist without the part or child and vice versa.
- As an example, an employee may belong to one or more departments in an organization. However, if an employee's department is deleted, the employee object would not be destroyed but would live on. Note that the relationships between objects participating in an aggregation cannot be reciprocal—i.e., a department may "own" an employee, but the employee does not own the department

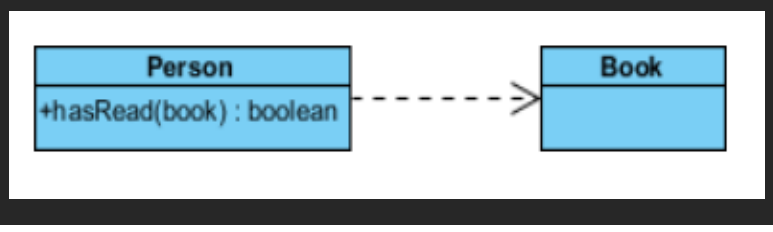
---

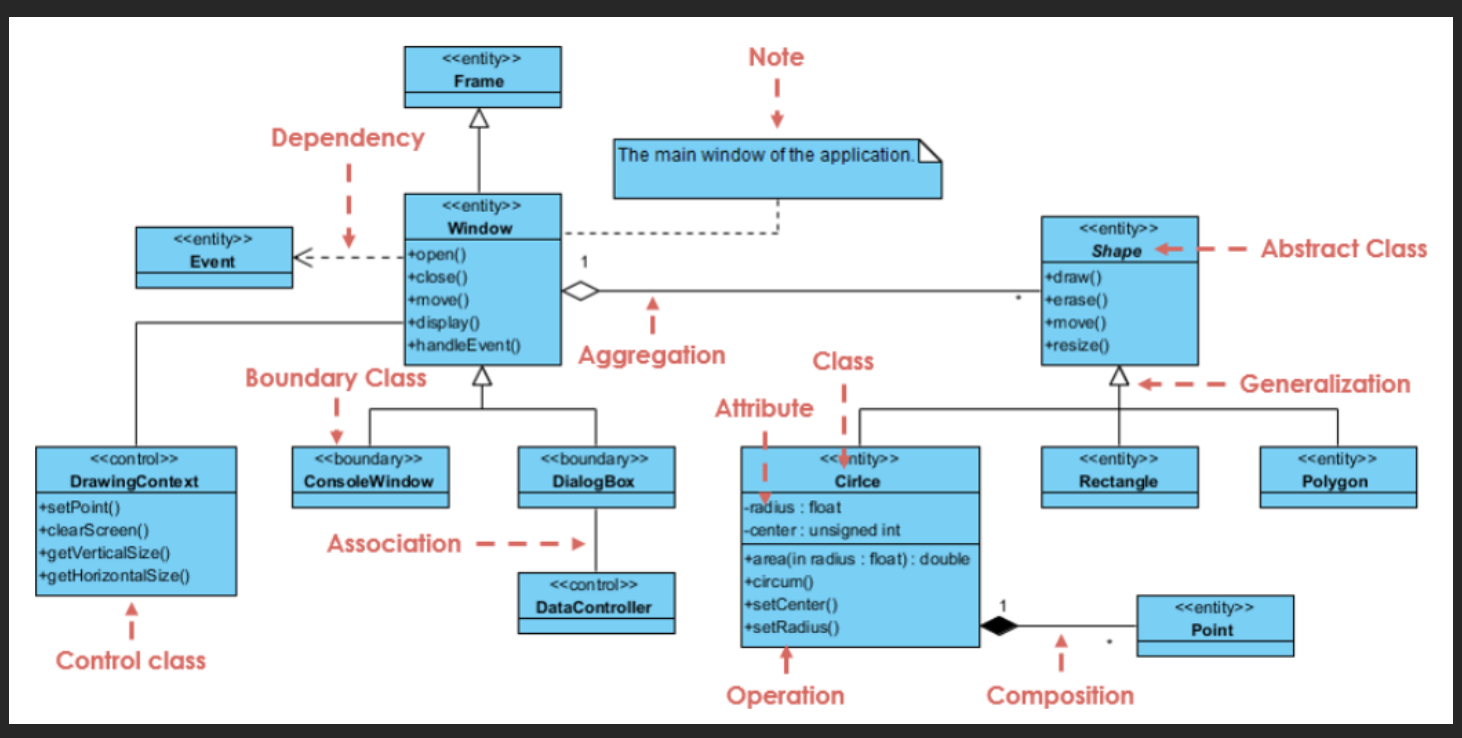
Composition

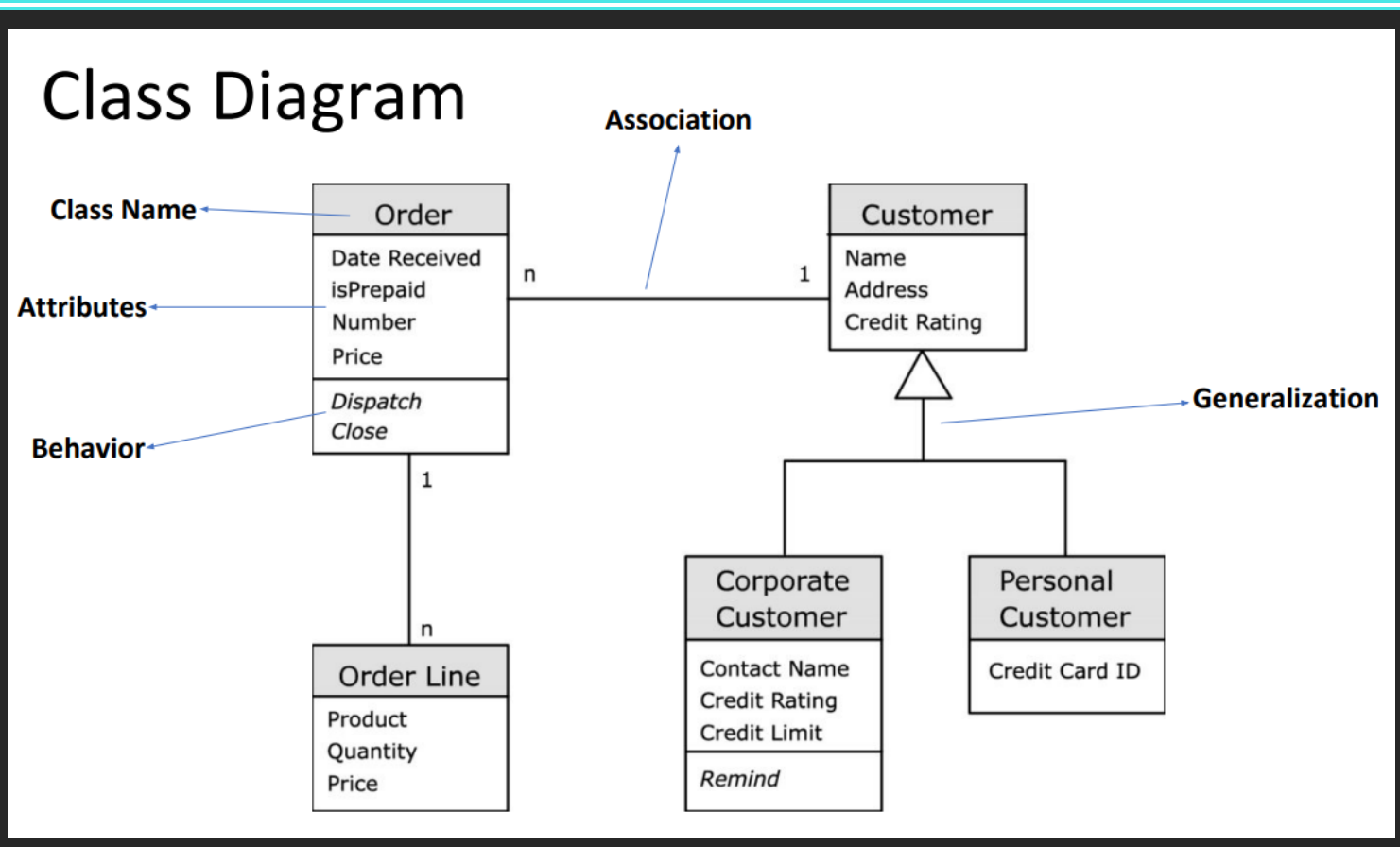
- Composition is a specialized form of aggregation.
- In composition, if the parent object is destroyed, then the child objects also cease to exist.
- Composition is actually a strong type of aggregation and is sometimes referred to as a "death" relationship.
- As an example, a house may be composed of one or more rooms. If the house is destroyed, then all of the rooms that are part of the house are also destroyed.
- Like aggregation, composition is also a whole/part or parent/child relationship.
- However, in composition the life cycle of the part or child is controlled by the whole or parent that owns it.
- It should be noted that this control can either be direct or transitive. That is, the parent may be directly responsible for the creation or destruction of the child or the parent may use a child that has been already created.
- Similarly, a parent object might delegate the control to some other parent to destroy the child object.

Dependency

- An object of one class might use an object of another class in the code of a method. If the object is not stored in any field, then this is modeled as a dependency relationship.
- The Person class might have a hasRead method with a Book parameter that returns true if the person has readthe book (perhaps by checking some database).







Class-Responsibility-Collaborator Modeling

- Class-responsibility-collaborator (CRC) modeling provides a simple means for identifying and organizing the classes that are relevant to system or product requirements.
- A CRC model can be viewed as a collection of index cards.
- Each index card contains a list of responsibilities on the left side and the corresponding collaborations that enable the responsibilities to be fulfilled on the right side.
- Responsibilities are the attributes and operations that are relevant for the class.
- Collaborators are those classes that provide a class with the information needed or action required to complete a responsibility

Collaborations:

Classes fulfill their responsibilities in one of two ways:

1. A class can use its own operations to manipulate its own attributes, thereby fulfilling a responsibility itself
2. A class can collaborate with other classes.

Collaborations are identified by determining whether a class can fulfill each responsibility itself. If it cannot, then it needs to interact with another class

Once a complete CRC model has been developed, the representatives from the stakeholders can review the model using the following approach:

1. All participants in the review (of the CRC model) are given a subset of the CRC model index cards. No reviewer should have two cards that collaborate.
2. The review leader reads the use case deliberately. As the review leader comes to a named object, she passes a token to the person holding the corresponding class index card.
3. When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card. The group determines whether one (or more) of the responsibilities satisfies the use case requirement.
4. If an error is found, modifications are made to the cards. This may include the definition of new classes (and corresponding CRC index cards) or revising lists of responsibilities or collaborations on existing cards.

State Diagrams:

A graphical description of all dialog between the system and its environment.

Node (state) represents a stable set of conditions that exists between event occurrences

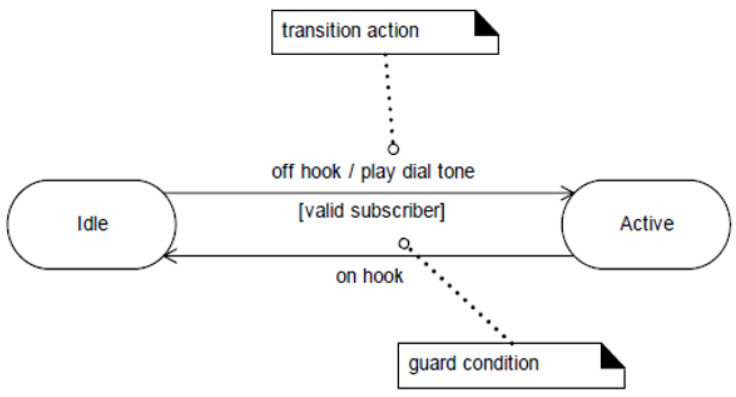
Edge (transition) represents a change in behavior or condition due to the occurrence of an event

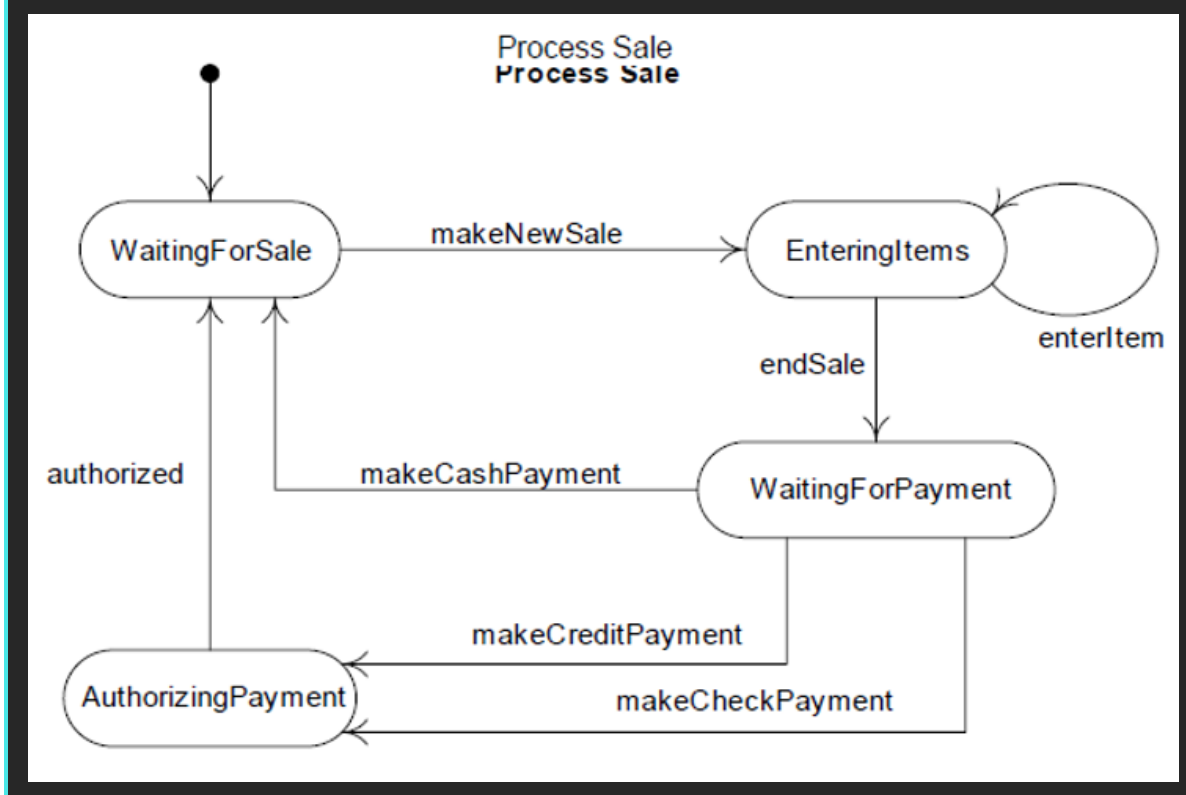
- Useful both for specifying dynamic behavior and for describing how behavior should change in response to the history of events that have already occurred.
- State chart diagrams provide us an efficient way to model the interactions or communication that occur within the external entities and a system.
- These diagrams are used to model the event-based system.
- A UML model is a collection of concurrently executing state charts.
- UML state chart diagram have a rich syntax, including state hierarchy, concurrency, and inter-machine communication.

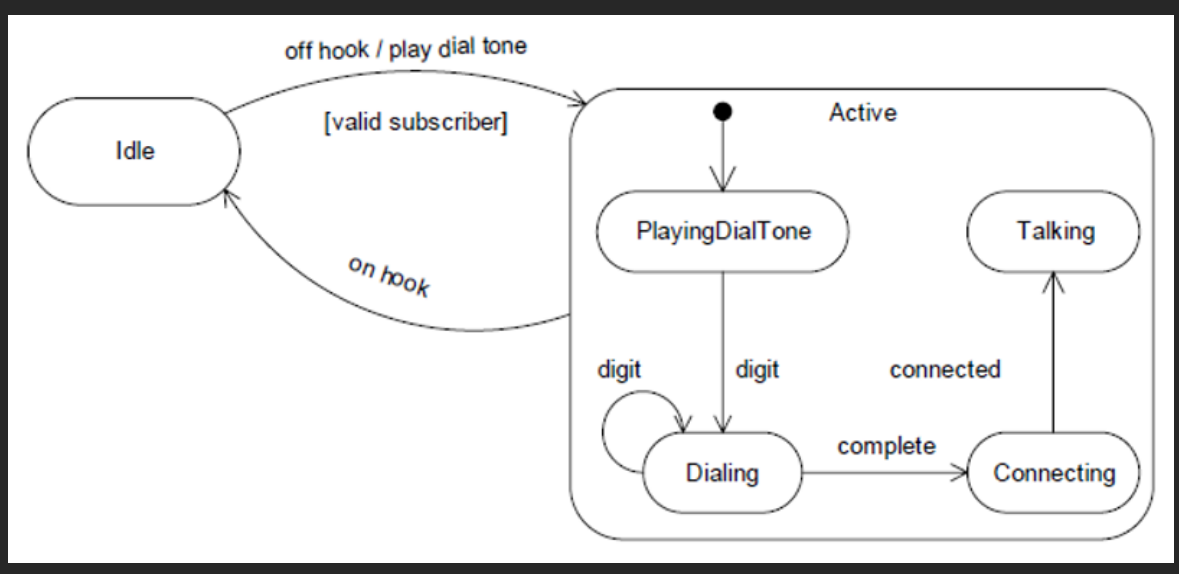
a do-activity **can only occur within a state** and cannot be attached to a transition.

A transition can cause an action to fire.

A transition may also have a conditional guard—or boolean test. The transition only occurs if the test passes.



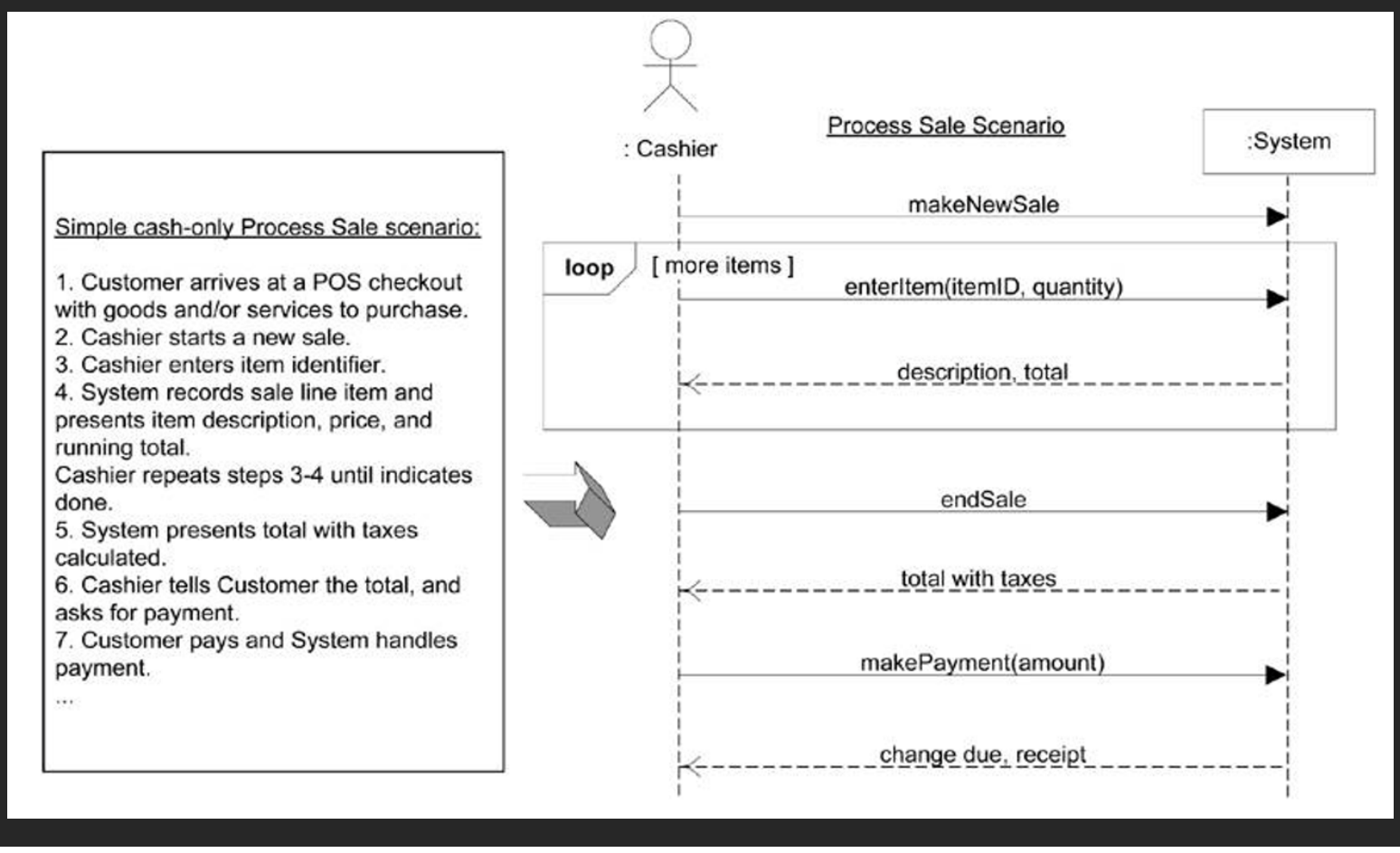


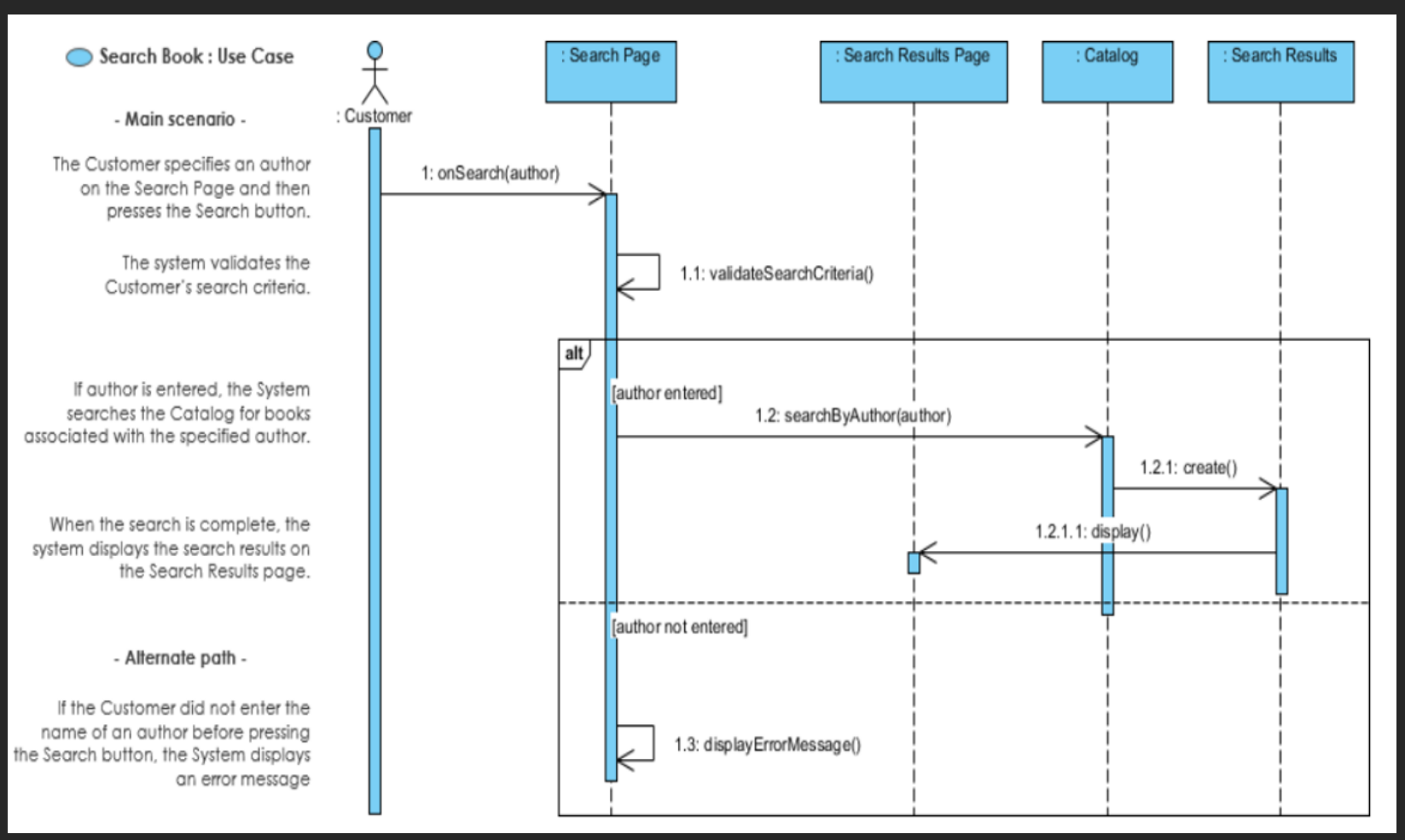


Sequence Diagrams:

A sequence diagram simply depicts interaction between objects in a sequential order, the order in which these interactions take place.

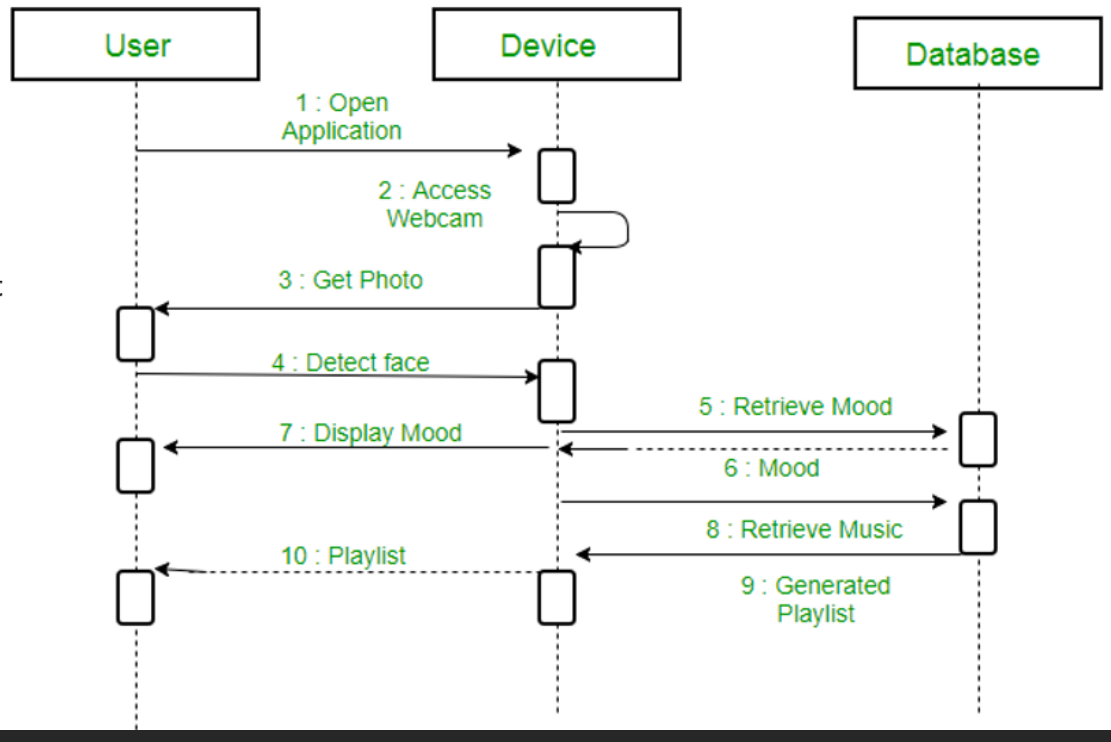
Sequence diagrams describe how and in what order the objects in a system function

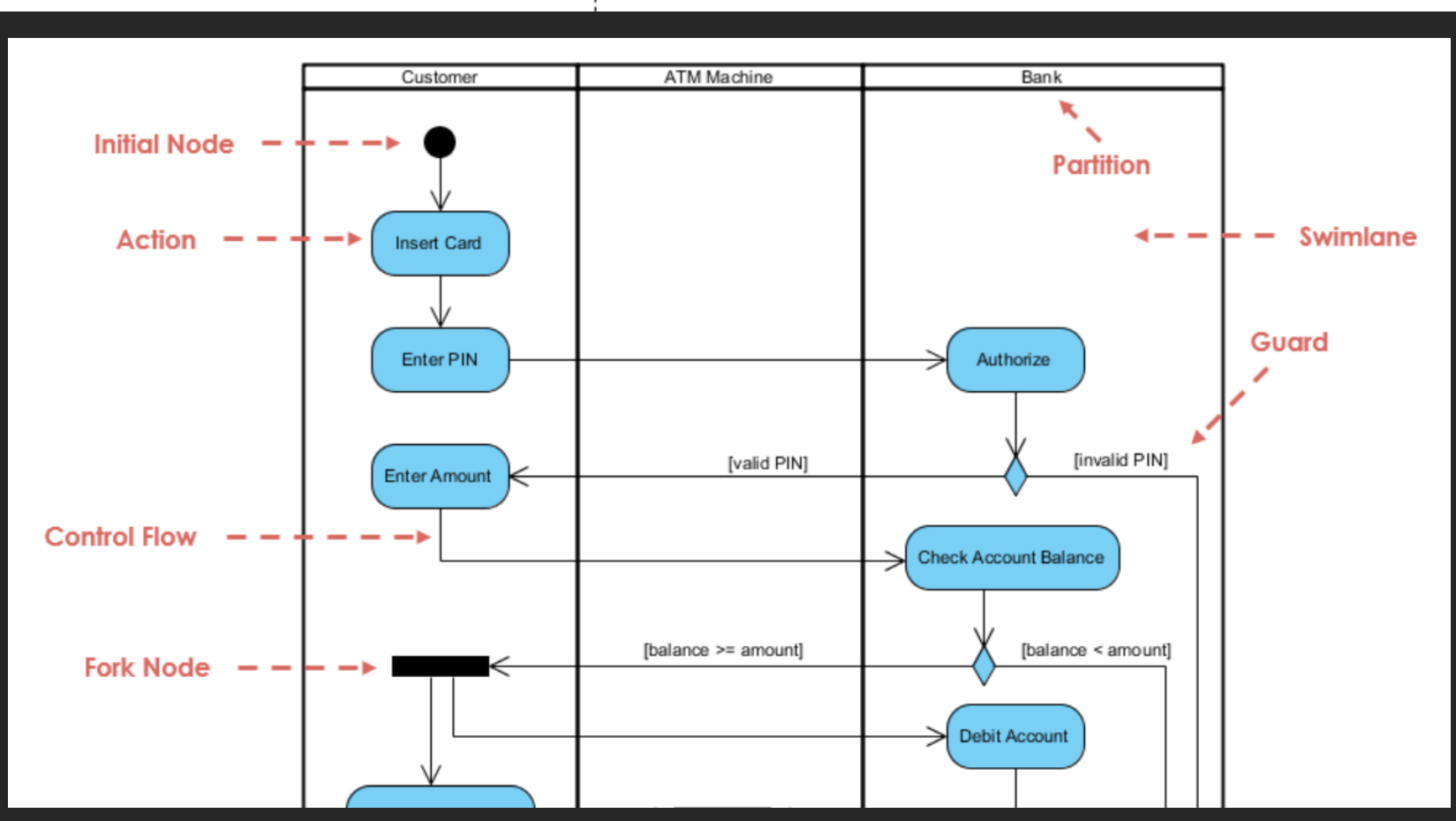




Sequence diagram for an emotion-based music player

- Firstly, the application is opened by the user.
- The device then gets access to the web cam.
- The webcam captures the image of the user.
- The device uses algorithms to detect the face and predict the mood.
- It then requests database for dictionary of possible moods.
- The mood is retrieved from the database.
- The mood is displayed to the user.
- The music is requested from the database.
- The playlist is generated and finally shown to the user.





Requirements:

Different stakeholders have different set of requirements: potentially conflicting ideas

Need to prioritize requirements to resolve conflicts

Prioritization might separate requirements into three categories

1. essential: absolutely must be met
2. desirable: highly desirable but not necessary
3. optional: possible but could be eliminated

Characteristics of Requirements:

The requirements must be:

1. Correct
2. Consistent
3. Unambiguous
4. Complete
5. Feasible
6. Relevant
7. Testable
8. Traceable

F/NF Requirements:

Functional requirement

Describes:

1. interaction between the system and its environment,
2. how should the system behave given certain stimuli,
3. required behaviour in terms of required activities

Example: For a system of printing pay cheques, the functional requirements must answer the following:

1. When are pay cheques issued?
2. What input is necessary for a pay cheque to be printed?

Quality requirement or non-functional requirement

Describes:

- some quality characteristic that the software must possess,
- a restriction on the system that limits our choices for constructing a solution

Constraints could be:

1. Design constraint: a design decision such as choice of platform or interface components
2. Process constraint: a restriction on the techniques or resources that can be used to build the system

Example: queries to the system must be answered within 3 seconds

- To put it simply, functional requirements describe what the product should do, while non-functional requirements place constraints on how the product should do it.

They can be expressed in the following form:

1. Functional requirement: "The system must do [requirement]."
2. Non-functional requirement: "The system shall be [requirement]."

Example:

1. Functional requirement: "The system must allow the user to submit feedback through a contact form in the app."
2. Non-functional requirement: "When the submit button is pressed, the confirmation screen shall load within 2 seconds."

Functional Requirements

Functionality

- What will the system do?
- When will the system do it?
- Are there several modes of operation?
- What kinds of computations or data transformations must be performed?
- What are the appropriate reactions to possible stimuli?

Data

- For both input and output, what should be the format of the data?
- Must any data be retained for any period of time?

User story: As an existing user, I want to be able to log into my account.

Functional requirement:

Log in:

1. The system must allow users to log into their account by entering their email and password.
2. The system must allow users to log in with their Google accounts.
3. The system must allow users to reset their password by clicking on "I forgot my password" and receiving a link to their verified email address

Non-functional Requirements

- Non-functional or Quality requirements, as the name suggests, are requirements that are not directly concerned with the specific services delivered by the system to its users.
- These non-functional requirements usually specify or constrain characteristics of the system as a whole.
- A nonfunctional requirement (NFR) can be described as a quality attribute, a performance attribute, a security attribute, or a general constraint on a system.

Example: Security

- Users shall be forced to change their password the next time they log in if they have not changed it within the length of time established as “password expiration duration”.
- Passwords shall never be visible at the point of entry or at any other time.
- Any new login shall be allowed after two step verification.

Testable Requirements:

Testable/Measurable Requirement:

- A requirement which is unambiguous and clearly specifies the behavior.
- Objective description of the requirement’s meanings.
- All possible entities and activities can be examined and classified as Meet Requirements and Do Not Meet Requirements.
- Testable requirements are helpful in making good design.
- Requirements that are not testable are likely to be ambiguous, incomplete and incorrect

3 ways to help make requirements testable:

1. Specify a quantitative description for each adverb and adjective
2. Replace pronouns with specific names of entities
3. Make sure that every noun is defined in exactly one place in the requirements document

Some examples:

1. Not Testable: Water quality information must be accessible immediately
2. Testable: Water quality information must be retrieved within five seconds of request

- Not Testable: The system should handle a large number of users at a time
- Testable: The system should handle 5000 users at a time

1. Not Testable: User should press the Save button when writing text in the system. This prevents it from being lost.
2. Testable: User should press the Save button when writing a note in the system. Pressing the Save button prevents the text from being lost.

Requirements Documentation:

- No matter what method we choose for defining requirements, we must keep a set of documents recording the result.
- We and our customers will refer to these documents throughout development and maintenance.
- Clear and precise illustrations and diagrams accompanying the documentation should be consistent with the text.

Requirements Definition

- The requirements definition is a record of the requirements expressed in the customer's terms.
- Working with the customer, we document what the customer can expect of the delivered system:
- Outline the general purpose and scope of the system, including relevant benefits, objectives, and goals
- Describe the background and the rationale behind proposal for new system
- Describe the essential characteristics of an acceptable solution
- Describe the environment in which the system will operate
- Outline a description of the proposal, if the customer has a proposal for solving the problem
- List any assumptions we make about how the environment behaves

Requirements Specification

- The requirements specification covers exactly the same ground as the requirements definition, but from the perspective of the developers.
- Where the requirements definition is written in terms of the customer's vocabulary, referring to objects, states, events, and activities in the customer's world, the requirements specification is written in terms of the system's interface.
- We accomplish this by rewriting the requirements so that they refer only to those real-world objects (states, events, actions) that are sensed or actuated by the proposed system.

Requirements Specification

Describe all inputs and outputs in detail, including

1. the sources of inputs
2. the destinations of outputs
3. the value ranges
4. data format of inputs and output data
5. data protocols
6. window formats and organizations
7. timing constraint
8. Restate the required functionality in terms of the interfaces' inputs and outputs
9. Devise fit criteria for each of the customer's quality requirements

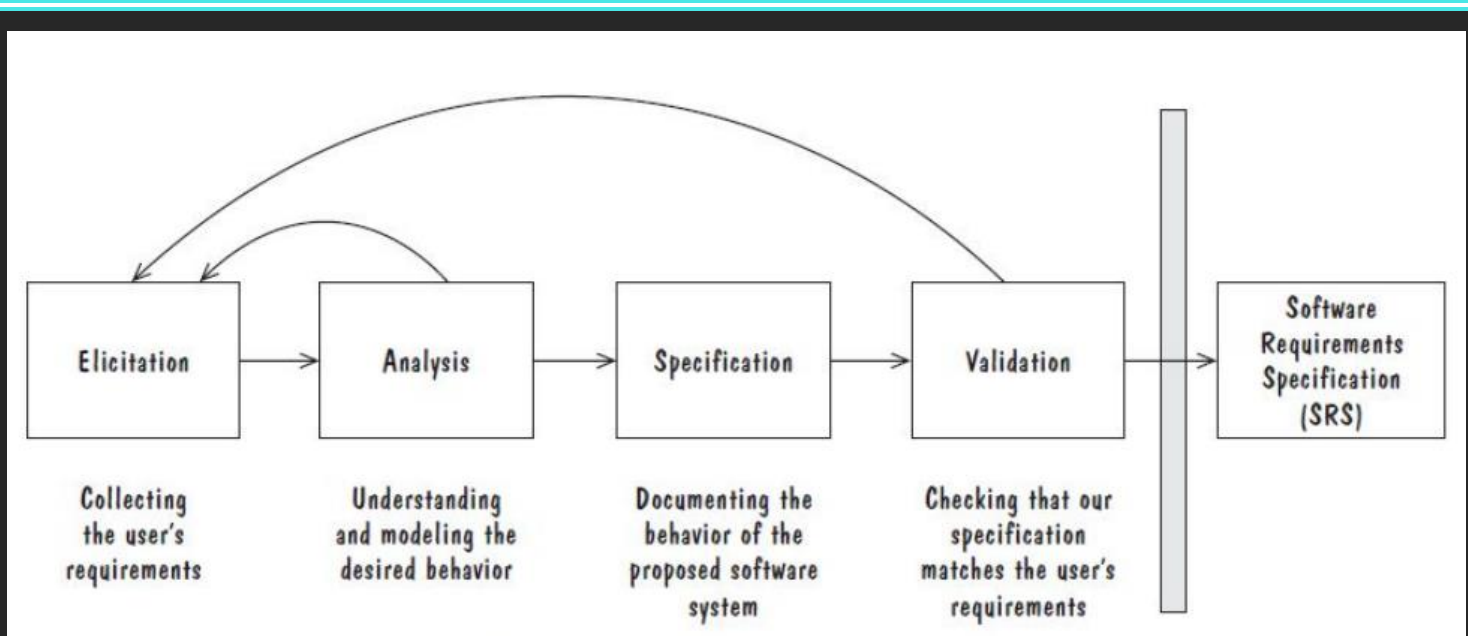


FIGURE 4.1 Process for capturing the requirements.

Process Management and Requirements Traceability:

- Process management is a set of procedures that track
- the requirements that define what the system should do
- the design modules that are generated from the requirement
- the program code that implements the design
- the tests that verify the functionality of the system
- the documents that describe the system
- It provides the threads that tie the system parts together

Validation and Verification

In Requirements Validation, we check that our requirements definition accurately reflects the customer's—actually, all of the stakeholders'—needs.

In Verification, we check that one document or artifact conforms to another. Thus, we verify that our code conforms to our design, and that our design conforms to our requirements specification; at the requirements level, we verify that our requirements specification conforms to the requirements definition.

To summarize, verification ensures that we build the system right, whereas validation ensures that we build the right system!

Requirements Review

Review the stated goals and objectives of the system

Compare the requirements with the goals and objectives

Review the environment in which the system is to operate

Review the information flow and proposed functions

Assess and document the risk, discuss and compare alternatives

Testing the system: how the requirements will be revalidated as the requirements grow and change

Measuring Requirements

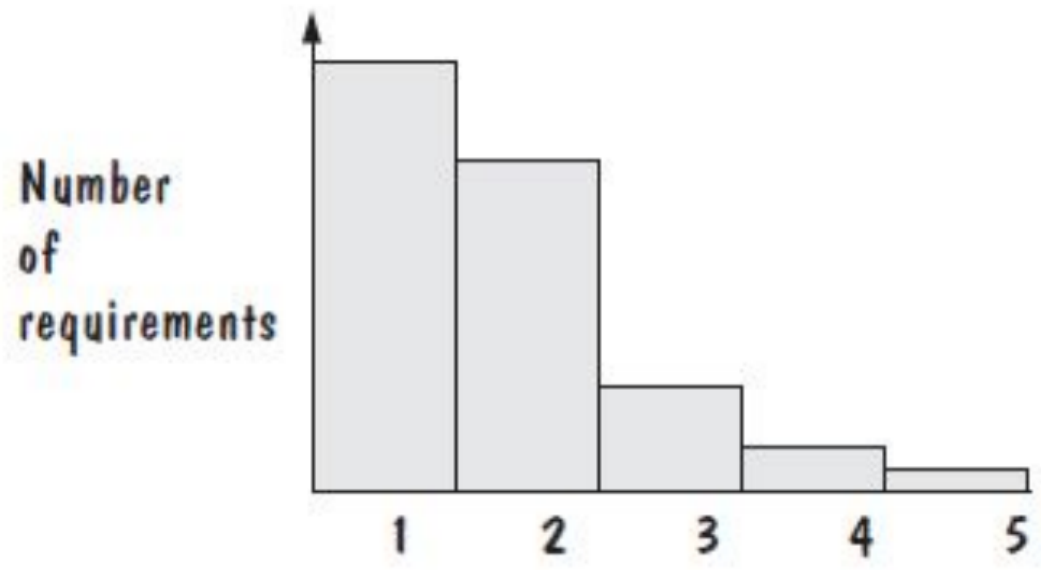
Measurements focus on three areas:

1. product
2. process
3. resources

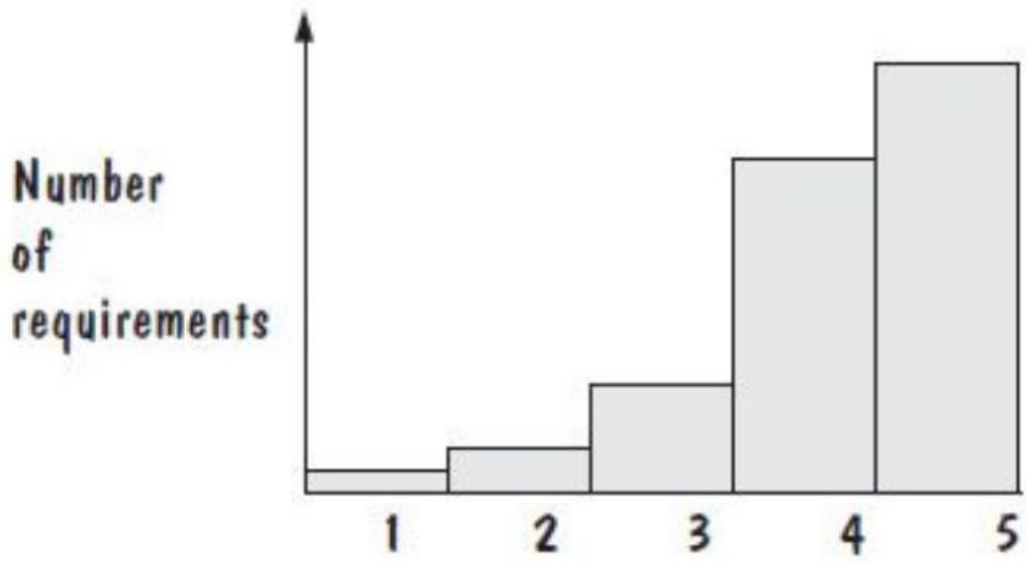
• Number of requirements can give us a sense of the size of the developed system • Number of changes to requirements • Many changes indicate some instability or uncertainty in our understanding of the system • Requirement-size and change measurements should be recorded by requirements type

Rating Scheme on Scale from 1 to 5 1. You understand this requirement completely, have designed systems from similar requirements, and have no trouble developing a design from this requirement 2. Some elements of this requirement are new, but they are not radically different from requirements that have been successfully designed in the past 3. Some elements of this requirement are very different from requirements in the past, but you understand the requirement and can develop a good design from it 4. You cannot understand some parts of this requirement, and are not sure that you can develop a good design 5. You do not understand this requirement at all, and can not develop a design

- If the designers and testers yield profiles with mostly 1s and 2s, as shown in Figure, then the requirements are in good shape and can be passed on to the design team.



- If there are many 4s and 5s, then the requirements should be revised, and the revisions reassessed to have better profiles, before we proceed to design.



Analysis• Focus on requirements • Each element should improve understanding of requirements • Delay consideration of infrastructure till design • Requirements model provides value to all stakeholders • Keep the models simple