Chapter 3:Analysis concept and principles

Compiled by:
Er. Shree Krishna Yadav

overview

- Understanding the requirements of a problem is among the most difficult tasks that a software engineer face.
- When you first think about it developing a clear understanding of requirements doesn't seem that hard.
- After all, doesn’t the customer know what is required?
- Shouldn't the end users have good understanding of the features and function that they require?
- Surprisingly in many instances the answer to the question is “no”.
- Even if customers and end users are explicit in their needs those needs will changes throughout the project

No Formal Customer Requirements

- Recipe for disaster:
- 1.The customer has only a vague idea(not clear) of what is required.
- 2.The developer is willing to proceed with the “vague idea” on the assumption that “we’ll fill in the details as we go” .
- 3.Repeat

a. Customer keeps changing requirements

b. Developer is “ratcheted” (engaged) by these changes, making errors in specifications and development.

- Until the project flops.

What is requirement?

- A requirement is **a features of the system or a description of something** the system is capable of doing in order to fulfill the systems purpose.
- A software requirement is a capability needed by the user to solve a problem or to achieve an objective. In other words, requirement is a software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documentation. Ultimately, what we want to achieve is to develop quality software that meets customers' real needs on time and within budget.
- Perhaps the greatest challenge being faced by software developers is to share the vision of the final product with the customer. All stakeholders in a project - developers, end users, software managers, customer managers - must achieve a common understanding of what the product will be and do, or someone will be surprised when it is delivered. Surprises in software are almost never good news.

Understanding requirements

- (what is it?)Before we begin any technical work,it is a good idea to create a set of requirements for **any engineering tasks**. These task lead to an understanding of what the business impact of the software will be,what the customer wants and how end users will interact with the software.
- (Who does it?) software engineers sometime referred to as **system engineer or analysts** and the other project stakeholder(managers, customer or end users)all participate in requirement engineering.
- (why is it important?) designing and building an elegant computer program **that solves the wrong problem** serves no ones need(that do not fulfill users requirement)that is why it is important to understand what the customer wants before we begin to design and build computer based system.

- (what are the steps?)requirements engineering begins with **inception**(a task that defines scope and nature of the problem to be solved)it moves onward to **elicitation** (a task that helps stakeholders define what is required),and then **elaboration**(where basic requirements are refined and modified)as stakeholders define the problem, **negotiation** occurs(what are the *priorities*, what is essential,when is it required)and then **reviewed** or validated to ensure that our understanding of the problem and the stakeholders understanding are same.
- (what is the work product?) the intent of requirements engineering is to provide all parties with a written understanding of the problem. This can be achieved through number of work products usage scenarios,functions and features lists,requirements models or a specification.

In simpler terms:
Goal: To agree on exactly what to build.

- **What they are**: Things you write down or draw (like user stories, diagrams, lists) that explain the problem and solution.
- **Why they matter**: They turn vague ideas into a shared understanding so developers and customers are on the same page, preventing mistakes and building the right thing.

Examples of these "work products":
Usage Scenarios: Stories of how users will interact with the system.
Functions/Features Lists: Bullet points of what the system must do.
Requirements Models: Diagrams (like UML) showing system structure or behavior.
Specifications: Detailed documents outlining everything required.

Functional and non-functional requirements

Functional requirements specify what a system **does** (e.g., user can log in),
while non-functional requirements define **how** the system does it, focusing on quality attributes like performance, security, and usability (e.g., the system must load in under 2 seconds)

Functional and non-functional requirements

Functional requirements

Definition: What the system must do; its specific features, tasks, and operations.

Purpose: To define the behavior and functions of the system.

Examples:

- Users can log in with a username and password.
- The system can calculate the total of items in a shopping cart.
- Users can search for products on the website.

Key characteristics:

- Often defined by the user.
- Straightforward to define.
- Tested using functional testing, such as unit or integration tests.

Non-functional requirements

Definition: How the system performs and the quality attributes it must possess.
Purpose: To define constraints and conditions for performance, security, and usability.
Examples:

- The website must load in under 2 seconds.
- The system should be able to handle 10,000 concurrent users.
- User passwords must be stored using a secure hashing algorithm.
- The application must be compatible with the latest versions of major web browsers.

Key characteristics:

- Often defined by technical experts.
- More difficult to define.
- Tested using performance, security, and usability testing.

What is requirement engineering

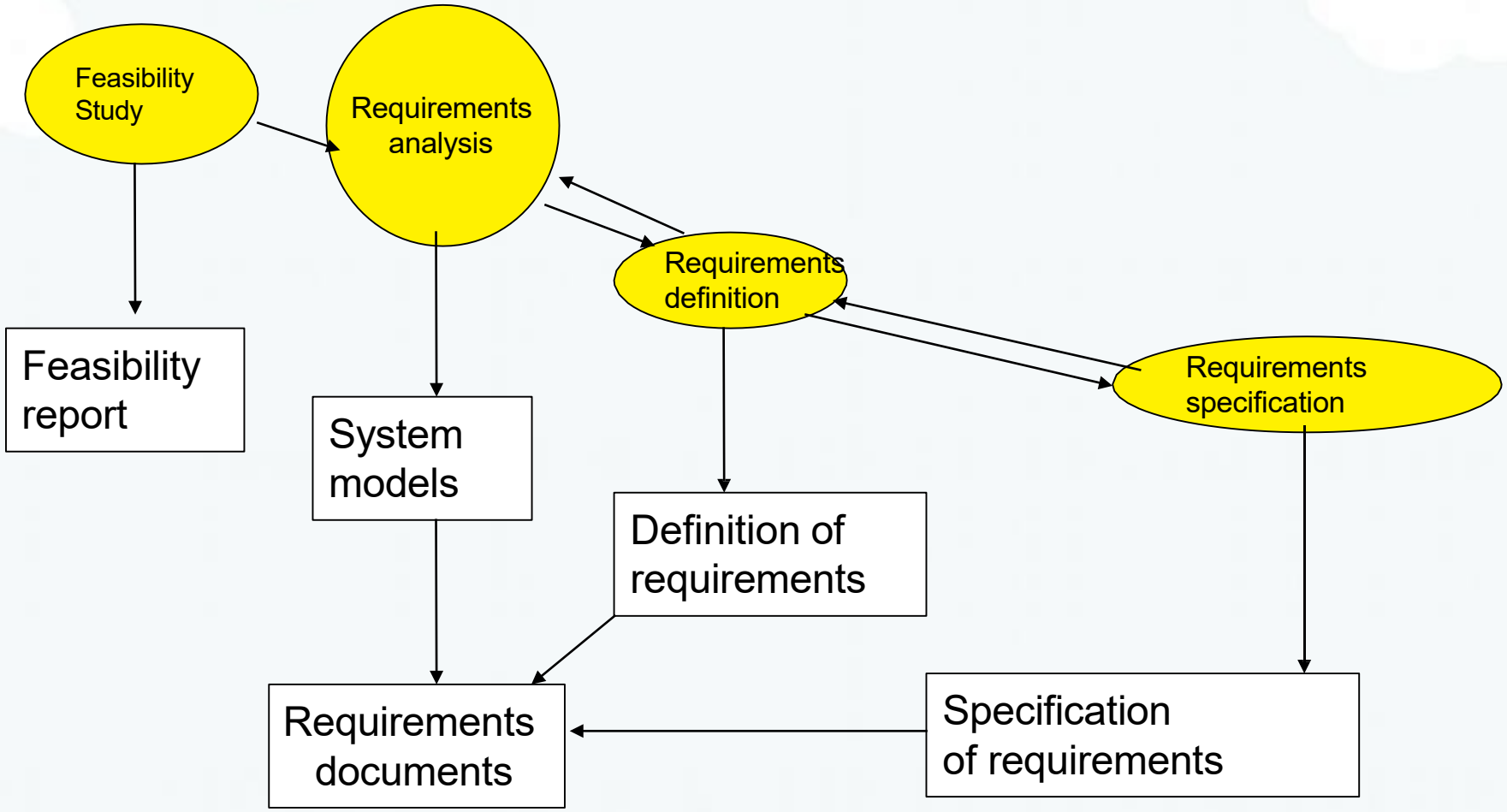
- The process to gather the software requirement from client,analyze and document them is known as requirement engineering.
- It is a process of gathering and defining service provided by the system. Requirements Engineering Process consists of the following main activities:
- Requirements elicitation
- Requirements specification
- Requirements verification and validation
- Requirements management

- Feasibility study:
  - Identify and estimate to see if user needs can be satisfied using current techniques and technologies.

- Requirements analysis:
  - The process of deriving the system requirements through observation of existing systems, discussion with users and customers, task analysis, and so on.

Requirements definition:
  - Translating the information into a REQ. document.

Requirements specification:
  - Define system requirements using a consistent precise, and complete way.
  - Using some requirements specification method


Feasibility Study
Requirements analysis
Requirements definition
Requirements specification
Feasibility report
System models
Definition of requirements
Requirements documents
Specification of requirements

Requirement analysis process

- **Domain understanding:**
  - Understand the business or application area (how the real-world system works).

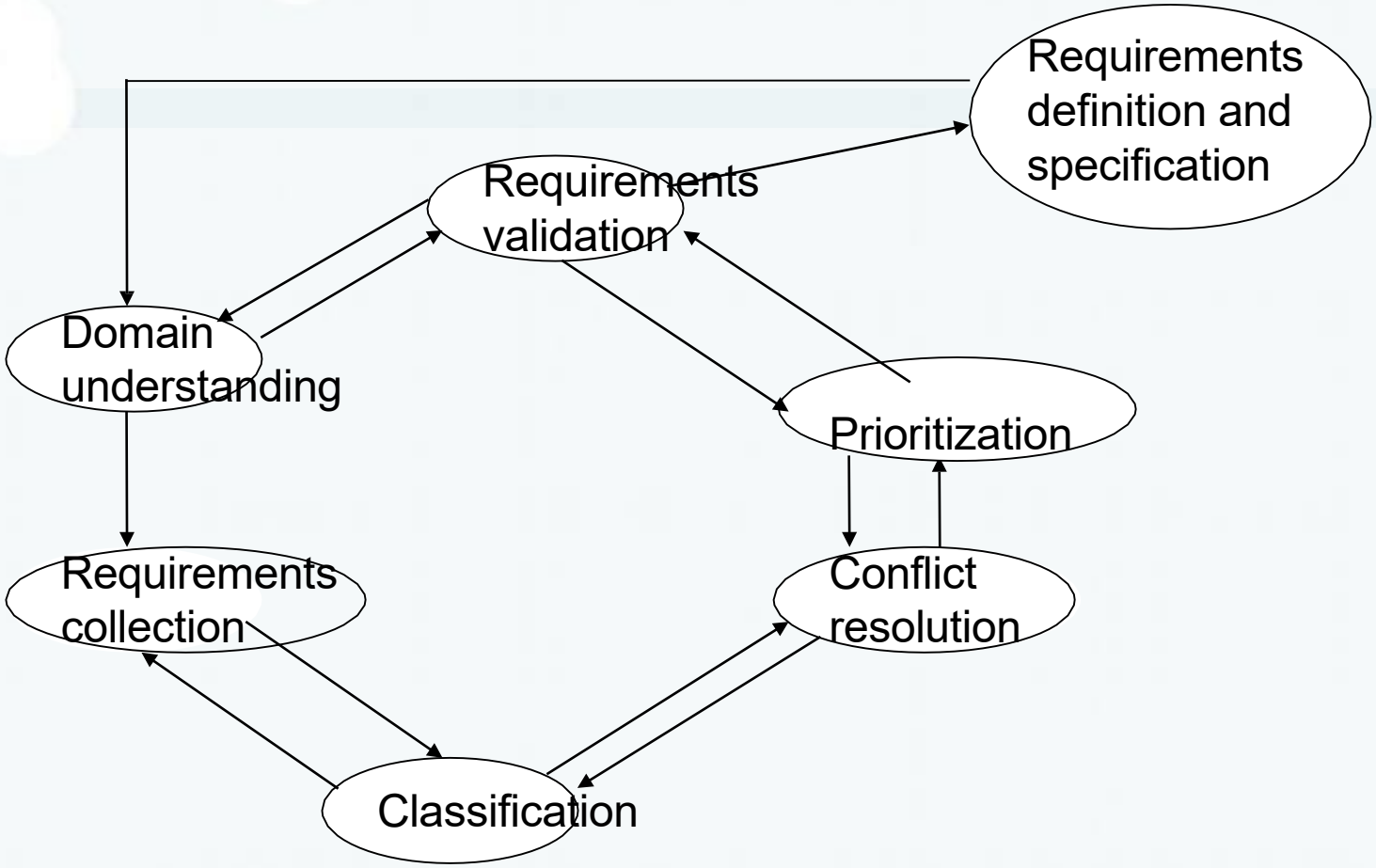
- **Requirements collections:**
  - The process of interacting with customers, users to discover the requirements for the system.

- **Requirements classification:**
  - Group and classify the gathered requirements. (e.g., functional, non-functional, UI, security)

- **Conflict resolution:**
  - When two requirements clash, discuss with stakeholders and agree on one solution.

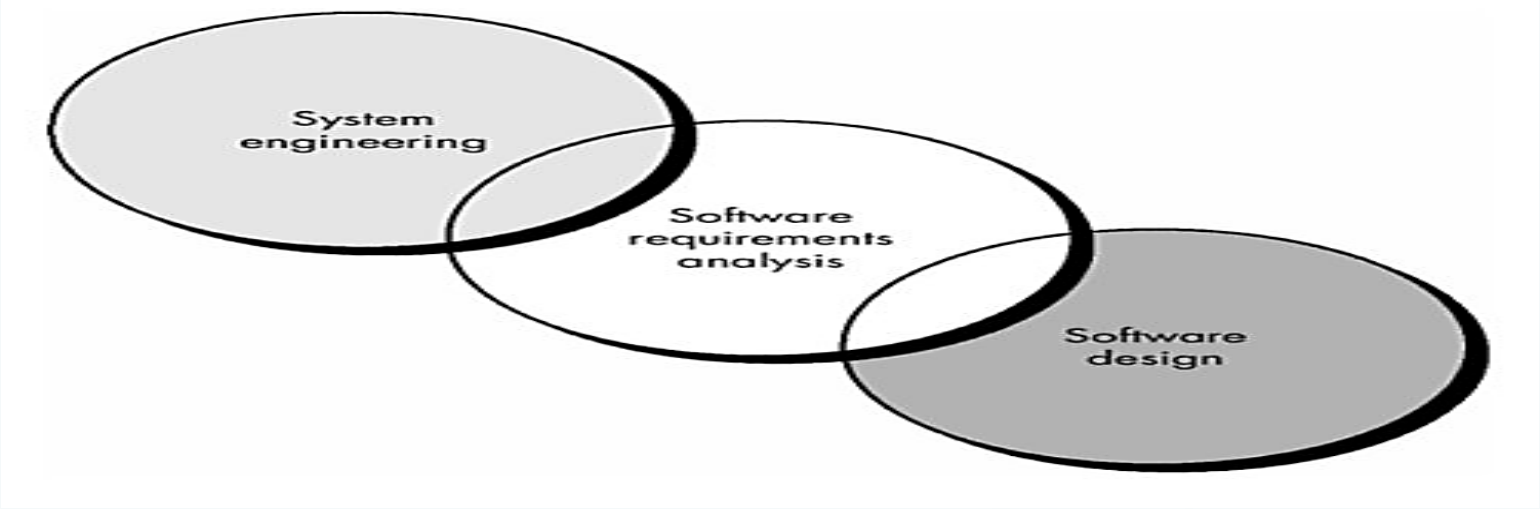
- **Prioritization:**
  - Identify and list requirements according to their importance

- **Requirements validation:**
  - Check and validate the gathered requirements to see if they are complete, correct, and sound.


Requirements definition and specification
Requirements validation
Domain understanding
Prioritization
Requirements collection
Conflict resolution
Classification

Requirement analysis

- Requirements analysis is a software engineering task that bridges the gap between system level requirements engineering and software design.
- Requirement Analysis is also known as Requirement Engineering.



- it is sometimes referred to loosely by names such as requirements gathering or requirements capturing.

Requirements engineering activities result in the specification of software's operational characteristics (function, data, and behavior), indicate software's interface with other system elements, and establish constraints that software must meet.

- Requirements analysis allows the software engineer/ analyst to refine the software allocation and build models of the data, functional, and behavioral domains that will be treated by software.
- Requirements analysis provides the software designer with a representation of information, function, and behavior that can be translated to data, architectural, interface, and component-level designs.

- Software requirements analysis may be divided into five areas of effort:

  (1) Problem recognition,

  (2) Evaluation and synthesis,

  (3) Modeling,

  (4) Specification, and

  (5) Review.

- First goal is recognition of the basic problem elements as perceived by the customer/users.
- The analyst studies the System Specification (if one exists) and the Software Project Plan.
- It is important to understand software in a system context and to review the software scope that was used to generate planning estimates.
- Next, communication for analysis must be established so that problem recognition is ensured.

Evaluation and solution synthesis is the next major area of effort for analysis. The analyst must define all externally observable data objects, evaluate the flow and content of information, define and elaborate all software functions.

- Upon evaluating current problems and desired information (input and output), the **analyst begins to synthesize one or more solutions**.
- Throughout evaluation and solution synthesis, the analyst's **primary focus is on "what," not "how."** What data does the system produce and consume, what functions must the system perform, what behaviors does the system exhibit, what interfaces are defined and what constraints apply?
- During the evaluation and solution synthesis activity, **the analyst creates models of the system in an effort to better understand data and control flow, functional processing, operational behavior, and information content**.
- The model serves as a foundation for software design and as the basis for the creation of specifications for the software.

- Modeling helps the analyst to understand the functionality of the system, system flow information and the process flow in the system
- Specification is a complete description of the behavior of a system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software. SRS document is a result of specification.
- Review is necessary to find out that whether the objectives of the project have been achieved or not. It is used for detecting and correcting defects in software artifacts, and preventing their leakage into field operations. Most common method used is FTR

Major tasks and efforts:

- Problem recognition (or system understanding)
  - Discover and understand the system requirements
  - Refine the requirements
- Evaluation and synthesis:
  - what are the alternative solutions
  - focus on what solution should be selected or used instead of how to implement a solution.
- Modeling: to represent the various aspects of the system
  - the required data
  - information and control flow
  - operation behavior
- Specification of:
  - software functions, and performance
  - interfaces between system elements
  - system constraints

Requirements Analysis

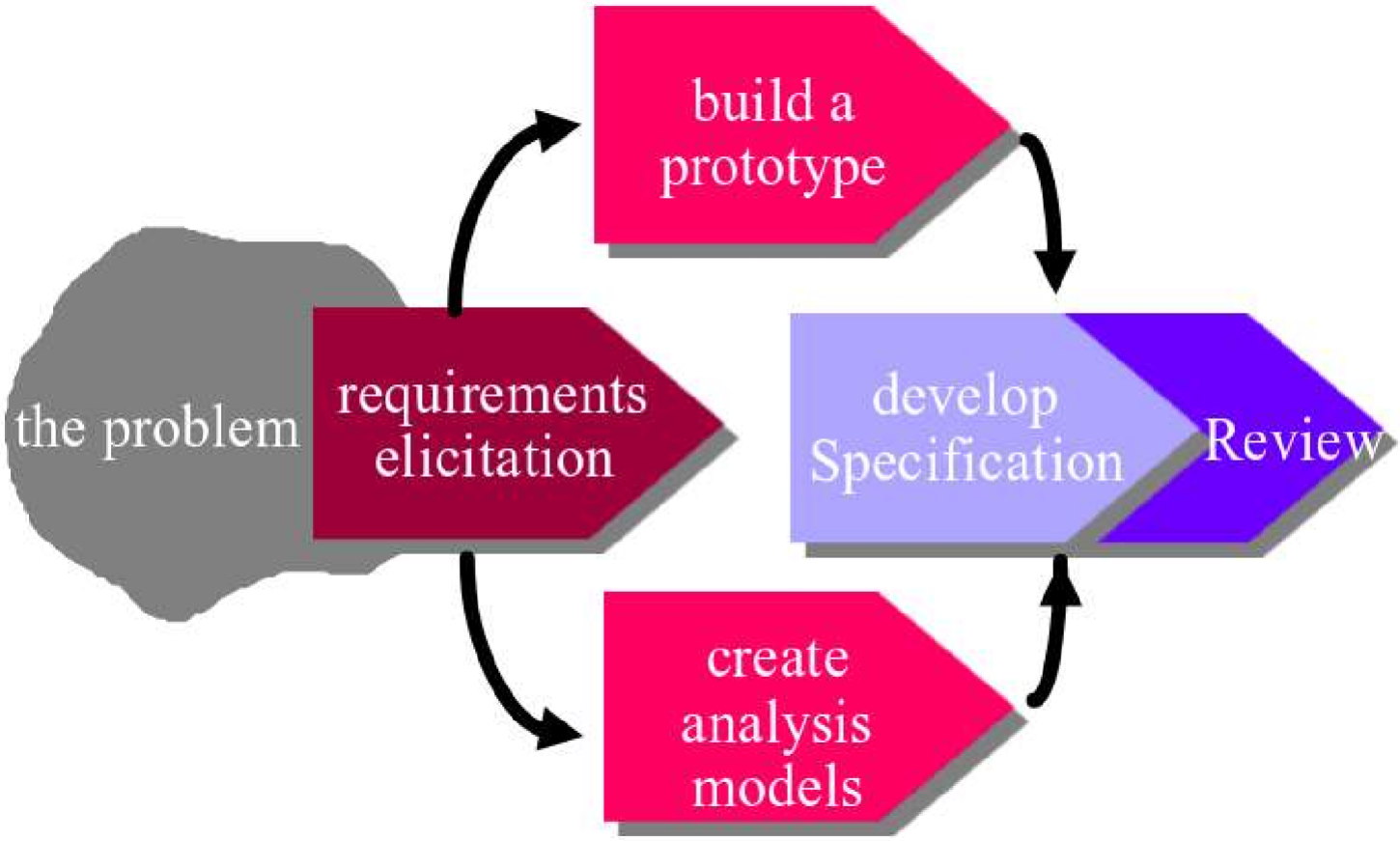
Requirements analysis
-> A process of discovery, refinement, modeling, and specification.
During the process, both the developers and customers take an active role.

Focus on: “what” instead of “how”

Input of the requirements analysis process:
- Software Project Plan
- System specification (if one exists)

Output: Software requirements specification document
- provides the software engineer with models that can be translated in to data, architectural, interface, and procedure design.
- customer and developer can check the quality of the software and provide the feedback.

Who perform requirements analysis: system analysts


build a prototype
the problem
requirements elicitation
develop Specification
Review
create analysis models

Requirement elicitation for software

- Before requirements can be analyzed, modeled, or specified they must be gathered through an elicitation (intelligence collection) process.
- Requirements elicitation is the practice of obtaining the requirements of a system from users, customers and other stakeholders. The practice is also sometimes referred to as **requirements gathering**.

Initiating the process

The most commonly used requirements elicitation technique is to conduct a **meeting or interview**. The analyst starts by asking context-free questions. That is, a set of questions that will lead to a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of the first encounter itself.

- The first set of context-free questions focuses on the customer, the overall goals, and the benefits.
- The next set of questions enables the analyst to gain a better understanding of the problem and the customer to voice his or her perceptions about a solution.
- The final set of questions focuses on the effectiveness of the meeting.
- The Q&A session should be used for the first encounter only and then replaced by a meeting format that combines elements of problem solving, negotiation, and specification.

Q1 set: Context free questions to lead the basic understanding of the problem (Identifies the sponsor, project owner, or funding source.)

- Who is behind the solution? (Identifies the end-users, target audience, or customer base.)
- Who will use the solution? (Defines the fundamental need or gap in the market/process.)
- What is the desired outcome? (Establishes the ultimate goal or benefit the project aims to deliver.)
- …..

Q2 set: Questions to gain a better understanding of the problem and the customer's perceptions about a solution.

- How would you characterize "good" output that would be generated by a successful solution? (Aids in defining success metrics, quality standards, and key performance indicators (KPIs) from the user's point of view.)
- What problems will this solution address? (Helps enumerate specific pain points, prioritize features, and define the scope of work.)

- Q3 set: Meta-questions focus on the effectiveness of the meeting.

- Are you the right person to answer these questions? Are your answers "official"? (Confirms the interviewee's expertise and authority regarding the project details.)

Facilitated application specification techniques

- (FAST), this approach encourages the creation of a **joint team of customers and developers** who work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements

- The basic guidelines of FAST:

- hold a meeting at a neutral site
- establish rules for preparation and participation
- have a formal meeting agenda
- control the meeting by a “facilitator”
- use a “definition mechanisms”
- have a common goal to identify the problem
- propose elements of solutions and requirements
- negotiate different approaches

Example of FAST

Joint Application Design(JAD)

- sit down with the client and design a paper UI that they can see what the application will look like and behave like.
- Give the user a chance to work through common scenarios and see if the application will work for them.
- Keep refining until the user feels the application is doing what they want it to do.
- As you get functionality implemented, bring the user in and have them work through those scenarios and see if it still works.
- If they want a change, have a solid estimate of how long the change will add to the schedule and how much it will cost.

Benefits of Facilitated application specification techniques

Bridges Expectation Gap: Reduces misunderstandings customers want and what developers build. between what

Improves Quality: Leads to better requirement analysis and error detection early on.

Faster Results: Accelerates the requirements gathering process.

Shared Ownership: Fosters commitment by involving stakeholders directly.

Quality function deployment

- Quality function deployment (QFD) is a quality management technique that **translates the needs of the customer into technical requirements for software.**
- QFD "**concentrates on maximizing customer satisfaction** from the software engineering process". To accomplish this, QFD emphasizes an understanding of **what is valuable to the customer and then deploys these values throughout the engineering process.**
- QFD identifies three types of requirements:

1. Normal requirements.

- The objectives and goals that are stated for a product or system during meetings with the customer.
- If these requirements are present, the customer is satisfied.

Examples of normal requirements might be requested types of graphical displays.(eg. A smartphone having a clear, working screen; a hotel room having clean sheets.)

2. Expected requirements.

- These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them.
- Their absence will be a cause for significant dissatisfaction.
- Examples of expected requirements are: ease of human/machine interaction, and ease of software installation.(A customer wants a car with "good fuel economy" or a laptop with "long battery life".)

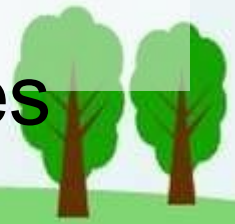
3. Exciting requirements.

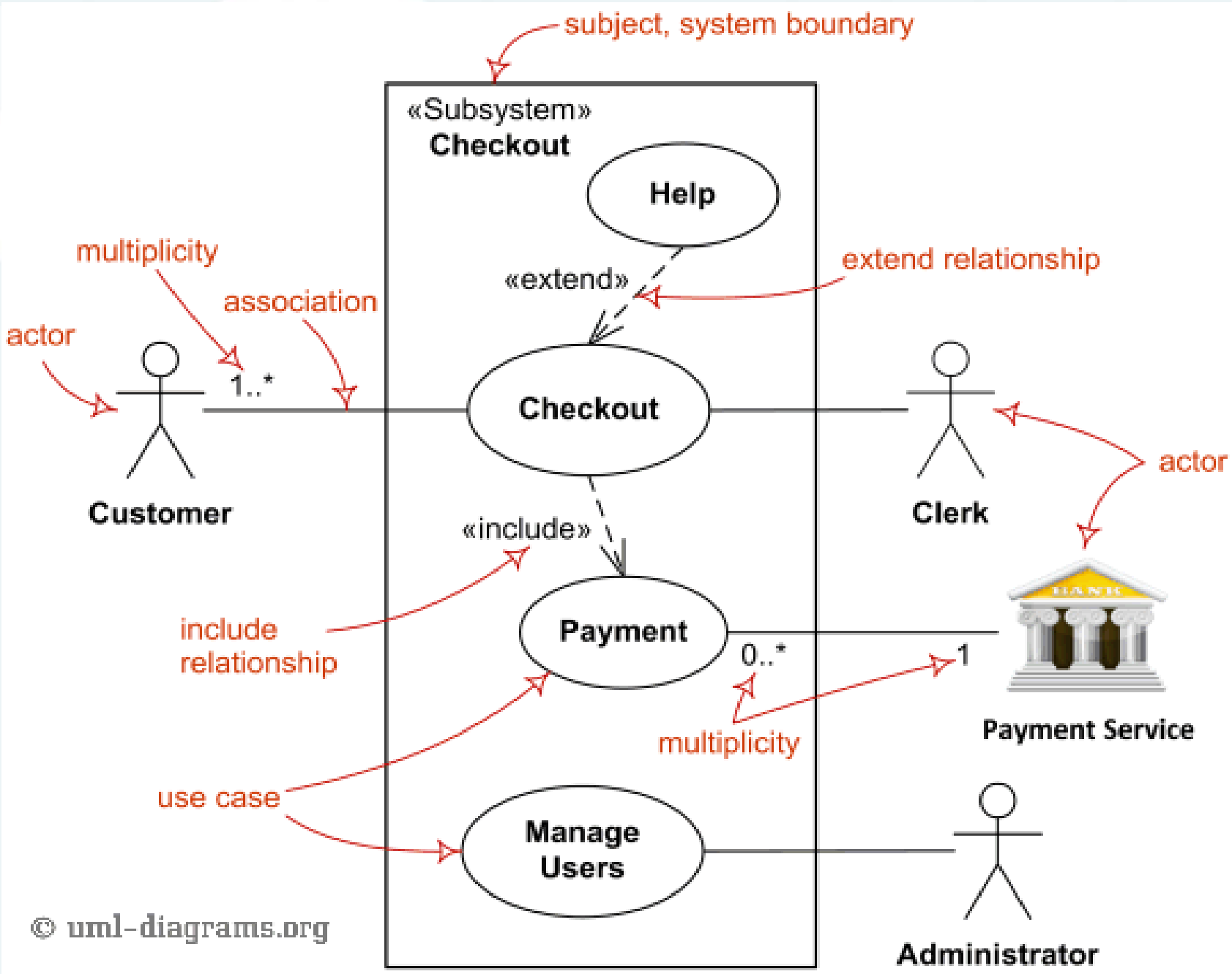
- These features go beyond the customer's expectations and prove to be very satisfying when present.
- For example, word processing software is requested with standard features. The delivered product contains a number of page layout capabilities that are quite pleasing and unexpected.

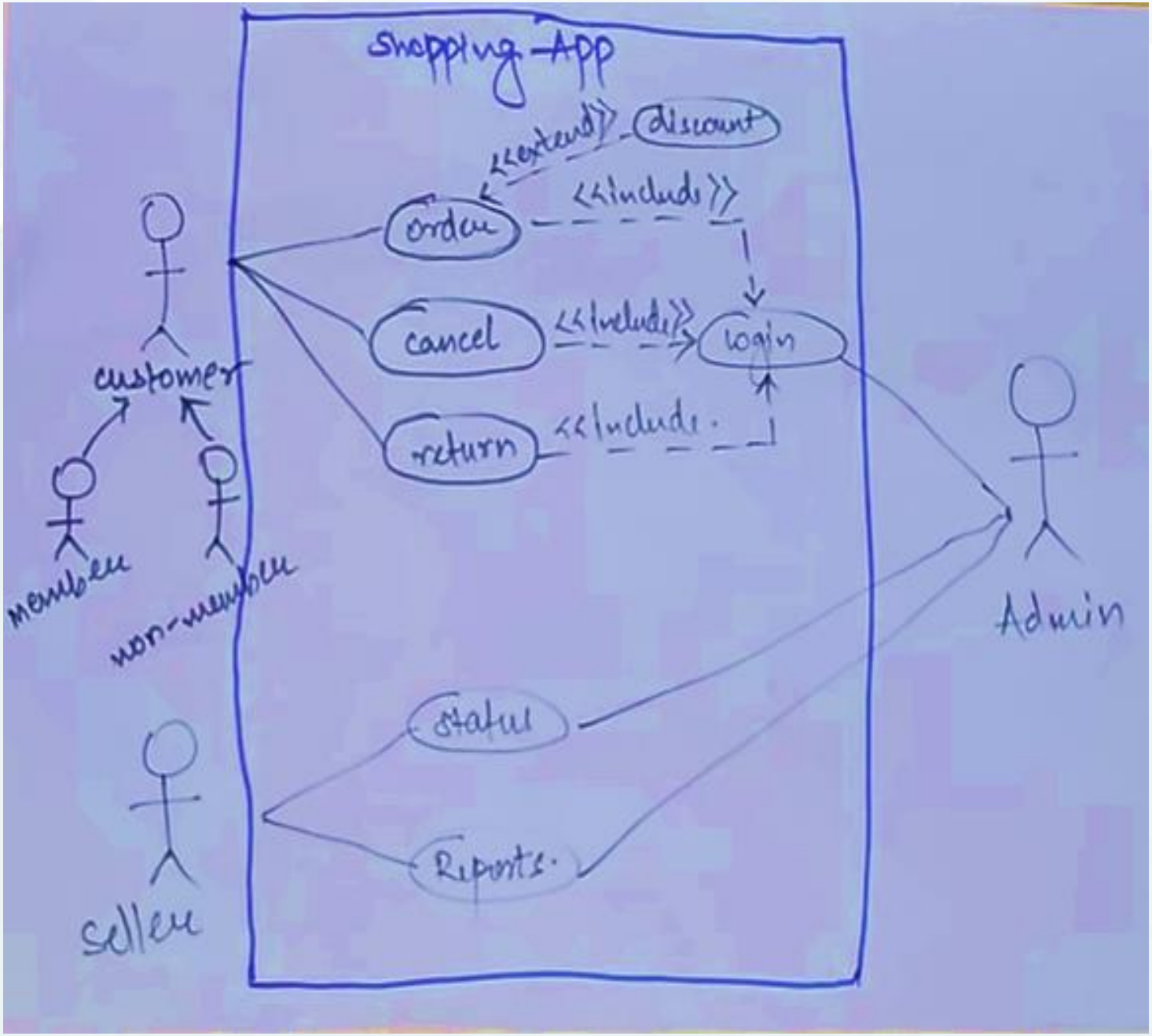
Use cases

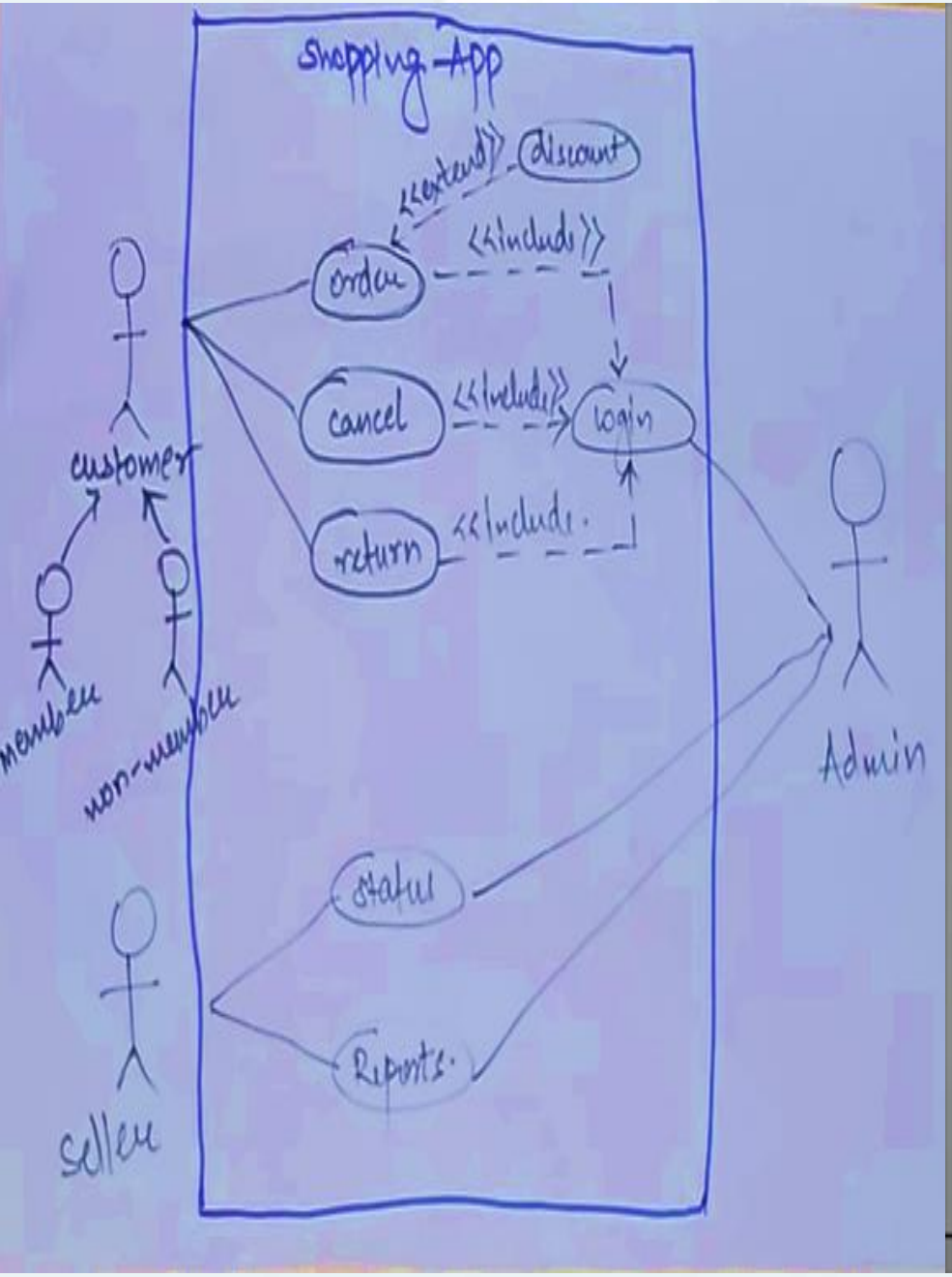
- Provide a description of how the system will be used. The use-case describes the manner in which an actor interacts with the system.
- To create a use-case, the analyst must first identify the different types of people (or devices) that use the system or product.
- These actors actually represent roles that people (or devices) play as the system operates. It is important to note that an actor and a user are not the same thing. A typical user may play a number of different roles when using a system, whereas an actor represents a class of external entities that play just one role.

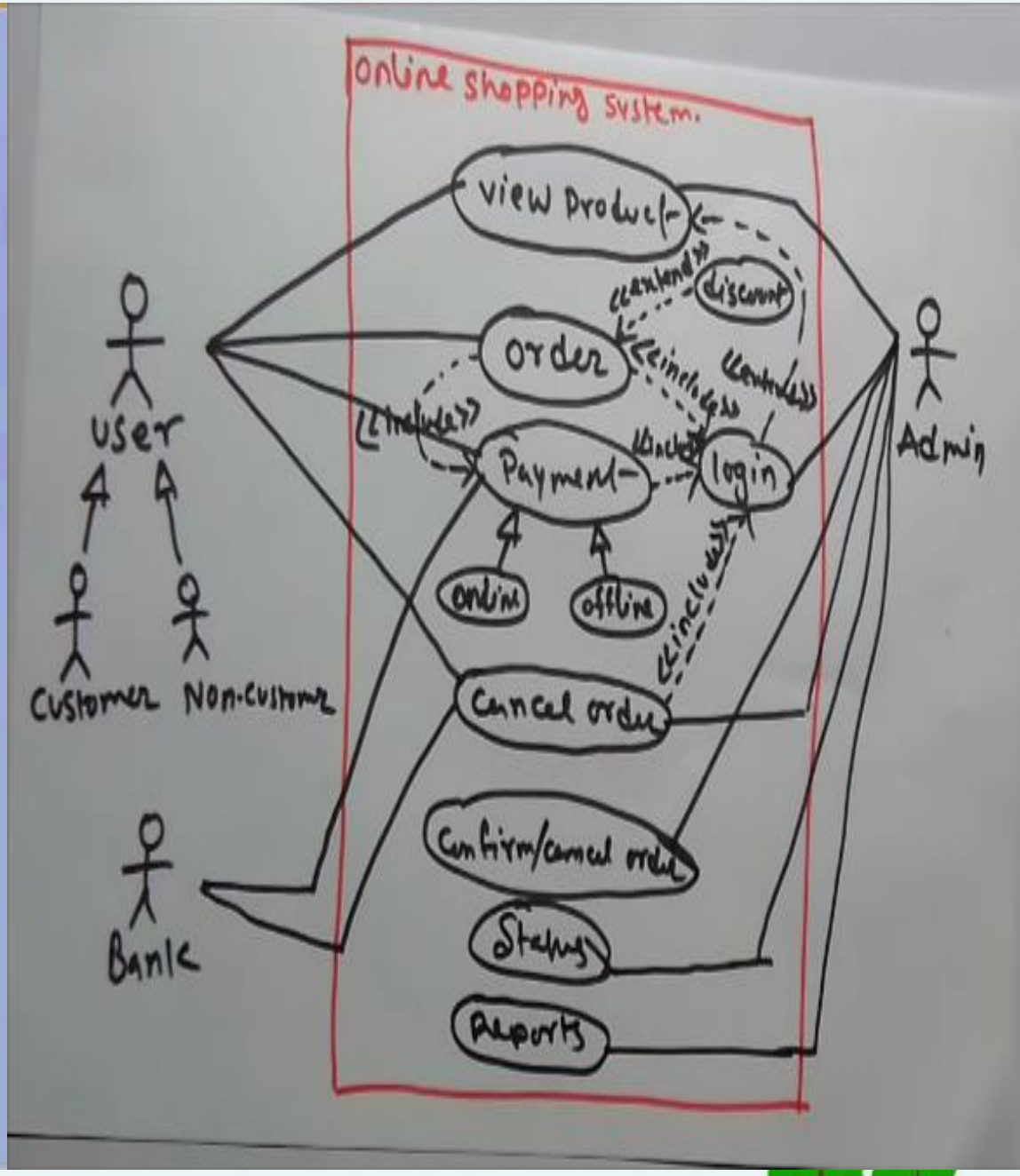
- Because requirements elicitation is an evolutionary activity, not all actors are identified during the first iteration.
- Primary actors interact to achieve required system function and derive the intended benefit from the system. They work directly and frequently with the software.
- Secondary actors support the system so that primary actors can do their work.
- Once actors have been identified, use-cases can be developed.




subject, system boundary
«Subsystem»
Checkout
Help
«extend»
extend relationship
multiplicity
association
actor
1..*
Checkout
Customer
Clerk
actor
«include»
include
relationship
Payment
0..*
1
multiplicity
Payment Service
use case
Manage
Users
Administrator
© uml-diagrams.org


Shopping App
<<extend>>
discount
order
<<include>>
cancel
<<include>>
login
return
<<include>>
customer
member
non-member
Admin
status
Reports
seller


Shopping App
<<extend>>
discount
order
<<includes>>
cancel
login
return
customer
member
non-member
Admin
status
Reports
seller


online shopping system.
view product
discount
order
Payment
login
User
Admin
online
offline
Customer
Non-customer
cancel order
confirm/cancel order
Status
Reports
Bank

Analysis principles

- All analysis methods are related by a set of operational principles:
- The information domain of a problem must be represented and understood.
- The functions that the software is to perform must be defined.
- The behavior of the software (as a consequence of external events) must be represented.
- The models that depict information function and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.
- The analysis process should move from essential information toward implementation detail.

By applying these principles, the analyst approaches a problem systematically.
The information domain is examined so that function may be understood more completely.
Models are used so that the characteristics of function and behavior can be communicated in a compact fashion.

Partitioning is applied to reduce complexity. Essential and implementation views of the software are necessary to accommodate the logical constraints imposed by processing requirements and the physical constraints imposed by other system elements.

Set of guiding principles for requirement engineering.

- A set of guidelines for requirement engineering:
  - understand the problem before beginning to create the analysis model.
  - develop prototypes to help user to understand how human-machine interactions
  - record the origin of and the reasons for every requirement
  - use multiple views of requirements(models (e.g., scenario-based, class-based, flow-oriented)
  - prioritize requirements
  - work to eliminate ambiguity

Information domain

Software is built to process data, to transform data from one form to another.

software analysis must begin with a comprehensive examination of the information domain to ensure a systematic understanding of the problem before design begins. Software also process events.

The first operational analysis principle requires to exam the information domain. Information domain contains three different views of the data and control:

- **information content and relationship:**
  information content --> represent the ***individual*** *data* and ***control objects***

  there are different relationships between data and objects

- **information flow:**
  represents the manner in which data and control change as each moves through a system. Data and control moves between two transformations (functions)

- **information structure:**
  represent the internal organization of various data and control items
  - data tree structure
  - data table (n-dimension)

Modeling

During software requirements analysis, we create models of the system to be built.
The models focus on:
- what the system must do (its requirements), not how it does it (its design).

The models usually have a graphic notation to represent:
- information, processing, system behavior, and other features

The second and third operational analysis principles require:
- build models of function and behavior

- Functional models
  Software transforms information. Three generic functions:
    - input, processing, output
- Behavior models
  Most software responds to events from the outside world
  A behavior model creates a representation of the states of the software and events that cause software to change state

Important roles of models:
- The model aids the analyst in understanding the information, function, and behavior of a system.
- The model becomes the focal point for review in the aspects of completeness, consistency, and accuracy of the specification.
- The model becomes the foundation for design, providing the designer with an essential representation of software.

Partitioning

- Partitioning decomposes a problem into its constituent parts.
- Problems are often too large and complex to be understood as a whole. For this reason, we tend to partition (divide) such problems into parts that can be easily understood, so that overall function can be accomplished.
- The fourth operational analysis principle suggests that the information, functional, and behavioral domains of software can be partitioned.
- Establish a hierarchical representation of information (or function):
- - exposing increasing detail by moving vertically in the hierarchy
- - decomposing the problem by moving horizontally in the hierarchy.

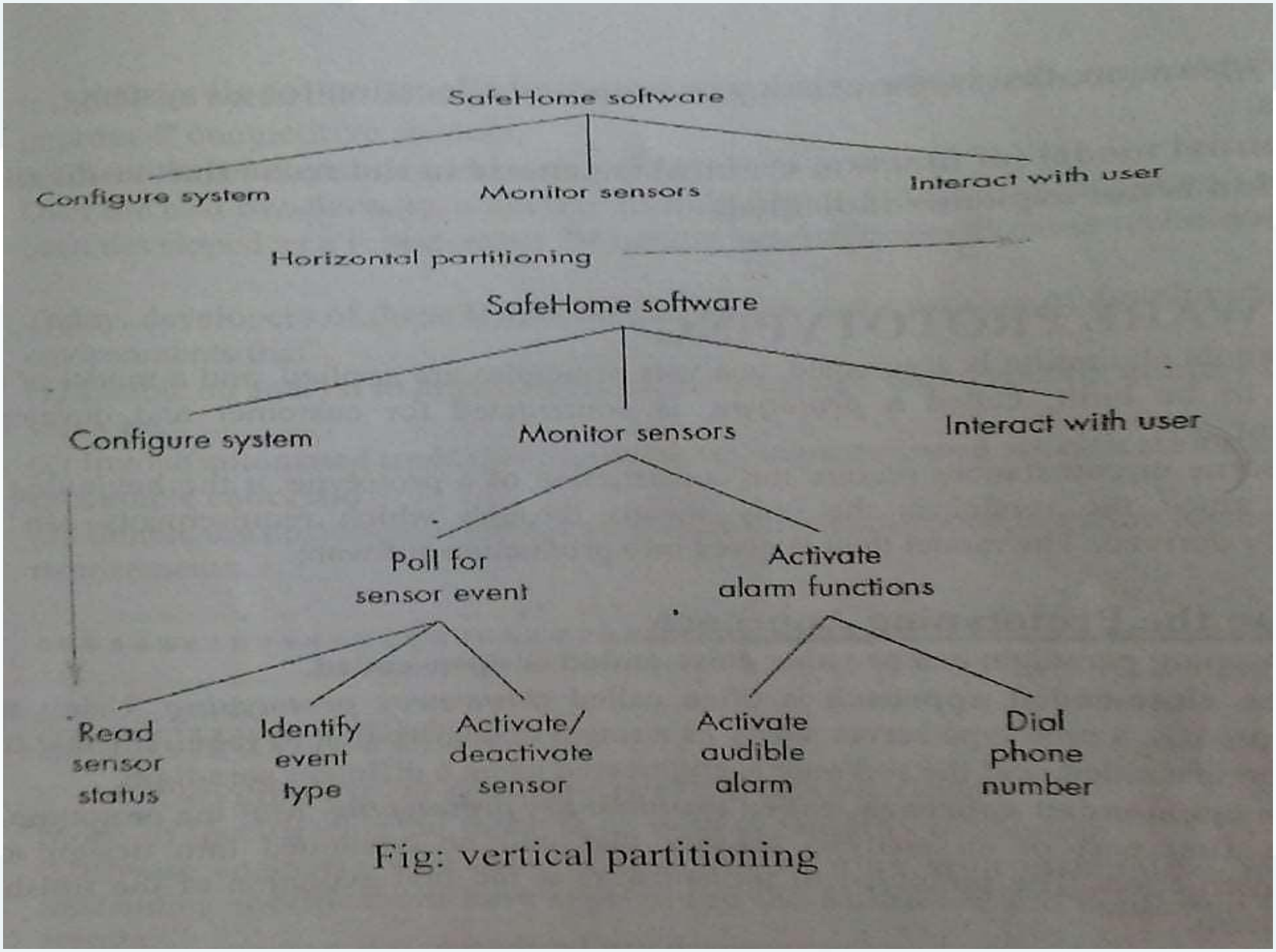


Fig: vertical partitioning

So, vertical partitioning means starting from the whole system at the top and step-by-step breaking one function into smaller and smaller sub-functions until each piece is simple to implement.

Software prototyping

- Requirements elicitation is conducted, analysis principles are applied, and a model of the software to be built, called a prototype, is constructed for customer and developer assessment.
- Finally, some circumstances require the construction of a prototype at the beginning of analysis, since the model is the only means through which requirements can be effectively derived. The model then evolves into production software.

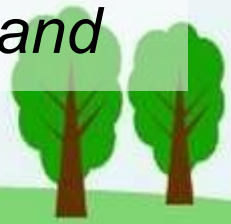
Selecting the Prototyping Approach

- The prototyping paradigm can be either close-ended or open-ended.
- The close-ended approach is often called **throwaway** prototyping. Using this approach, a prototype serves solely as a rough demonstration of requirements. It is then discarded, and the software is engineered using a different paradigm.

(*You quickly build a simple model to test a specific idea or clarify confusing requirements. Once the stakeholders understand what they need, the prototype is literally "thrown away"*)

- An open-ended approach, called **evolutionary prototyping**, uses the prototype as the first part of an analysis activity that will be continued into design and construction. The prototype of the software is the first evolution of the finished system.

(*You build a high-quality, functional version of the most important core features first. Instead of discarding it, you keep adding and refining features based on user feedback until it eventually becomes the final product.)*



- Before a close-ended or open-ended approach can be chosen, it is necessary to determine whether the system to be built is suitable to prototyping.
- A number of prototyping candidacy factors can be defined: application area,application complexity, customer characteristics, and project characteristics.
- Prototyping Methods and Tools
- For software prototyping to be effective, a prototype must be developed rapidly so that the customer may assess results and recommend changes.

Prototyping Methods and Tools:

- Fourth Generation Techniques

Fourth generation techniques (4GT) encompass a broad array of database query and reporting languages, program and application generators, and other very high-level nonprocedural languages. Because 4GT enable the software engineer to **generate executable code quickly, they are ideal for rapid prototyping.**

- Reusable Software Components

Another approach to rapid prototyping is to assemble, rather than build, the prototype by using a set of existing software components. Melding prototyping and program component reuse will work only **if a library system is developed** so that components that do exist can be cataloged and then retrieved. It should be noted that an existing software product can be used as a prototype for a new,improved" competitive product.

Fourth Generation Techniques (4GT) are a set of software development tools and languages designed to enable rapid application development with minimal manual coding. They are specifically suited for tasks like database querying, report generation, and screen definition.

Key characteristics and benefits include:

Non-procedural nature: Unlike traditional procedural languages (like C or Java) where the programmer specifies how a task is done, 4GT ***focuses on specifying what is needed, and the tool automatically determines the execution logic***.

Rapid Development: They allow software engineers to generate functional, executable code much faster than third-generation languages (3GLs), significantly accelerating the development process.

Ideal for Prototyping: Because of their speed and ease of use, 4GT are well-suited for creating rapid prototypes, allowing stakeholders to visualize and refine requirements quickly before significant resources are committed to full-scale development.

Examples of tools that embody 4GT principles include **report generators, screen painters, application generators, and specialized database languages**. Modern low-code and no-code platforms are contemporary extensions of these core concepts.

- Formal Specification and Prototyping Environments

Over the past two decades, a number of formal specification languages and tools have been developed as a replacement for natural language specification techniques. Today, developers of these formal languages are in the process of developing interactive environments that

(1) Enable an analyst to interactively create language-based specifications of a system or software,

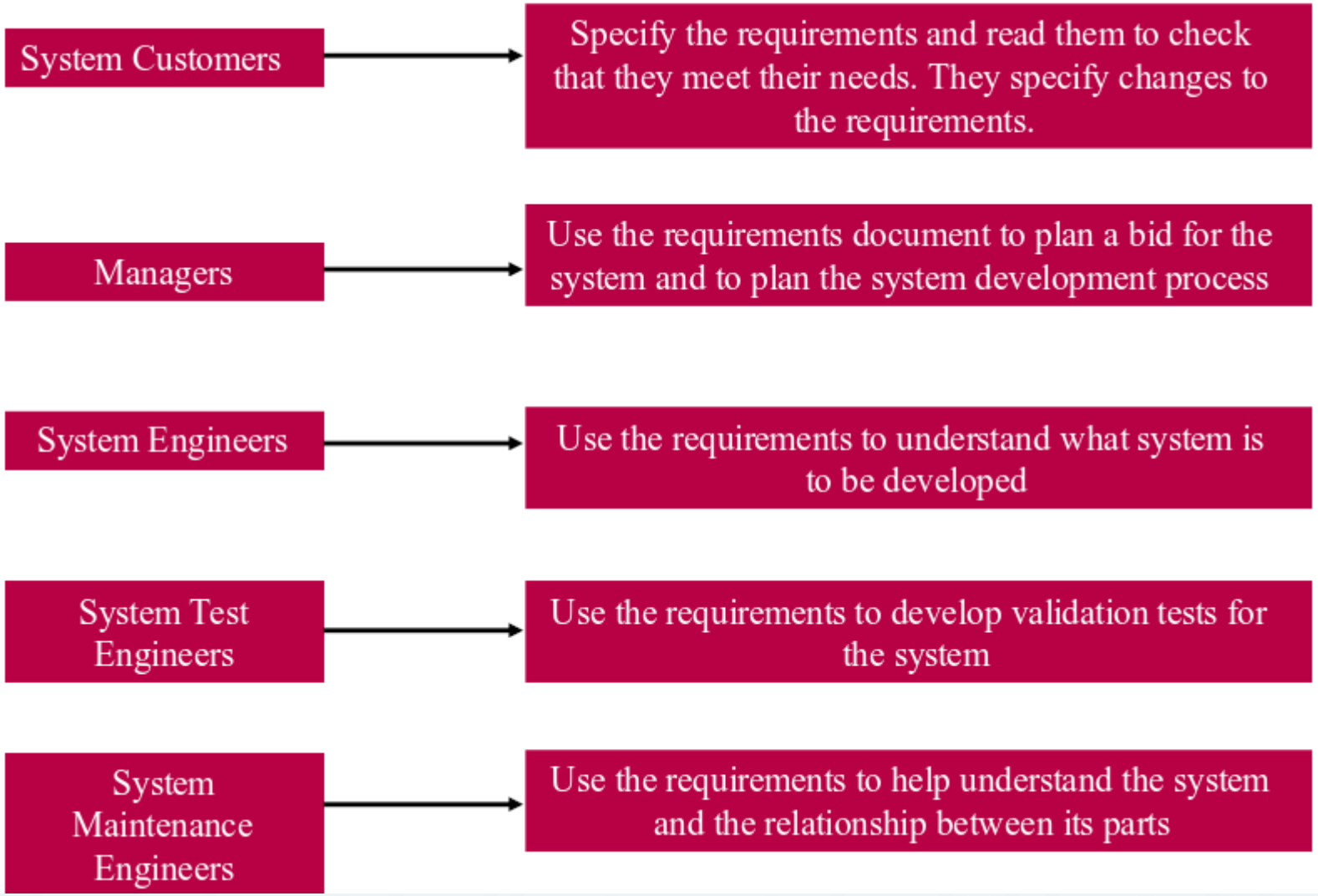
(2) Invoke automated tools that translate the language-based specification into executable code, and

(3) Enable the customer to use the prototype executable code to refine formal requirements.

Specification

- End product of analysis .
- The requirements specification is the official statement of what is required of the system
- developers Should include both a definition and a specification of requirements.
- It is NOT a design document. As far as possible, it should set out WHAT the system should do rather than HOW it should do it require. elicitation build a prototype analysis models develop spec. Review

User of requirement specification



Specification principles

Specification principles:

- Separate functionality from implementation
- Develop a model of the desired behavior of a system
- Establish the context in which software operates
- Define the environment in which the system operates
- Create a cognitive model rather than a design or implementation model. The cognitive model describes a system as perceived by its user community.
- Specification is an abstract model of a real system
- Establish the content and structure of a specification (easy to be changed)

Guideline for representation

- Representation format and content should be relevant to the problem

A general outline for the contents of a Software Requirements Specification can be developed. However, the representation forms contained within the specification are likely to vary with the application area.

- Information contained within the specification should be nested

Representations should reveal layers of information so that a reader can move to the level of detail required. Paragraph and diagram numbering schemes should indicate the level of detail that is being presented. It is sometimes worthwhile to present the same information at different levels of abstraction to aid in understanding.

-Diagrams and other notational forms should be restricted in number and consistent in use.

Confusing or inconsistent notation, whether graphical or symbolic, degrades understanding and fosters errors.

- Representations should be revisable

The content of a specification will change. Ideally, CASE tools should be available to update all representations that are affected by each change.

Software requirement specification(SRS)

- A software requirement specification(SRS) is a work product that is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence.
- A typical SRS includes:
- A purpose
- An overall description
- Specific requirement

How to Write an SRS Document

1. Create an Outline (Or Use an SRS Template)

our first step is to create an outline for your software requirements specification. This may be something you create yourself. Or we may use an existing SRS template.

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

2. Start With a Purpose

- The introduction to SRS is very important. It sets the expectation for the product you're building.
- So, start by defining the purpose of your product.
- Intended Audience and Intended Use

Define who in your organization will have access to the SRS — and how they should use it. This may include developers, testers, and project managers. It could also include stakeholders in other departments, including leadership teams, sales, and marketing.

- Project Scope
- Describe the software being specified. And include benefits, objectives, and goals. This should relate to overall business goals, especially if teams outside of development will have access to the SRS.

- 3. Give an Overview of What You'll Build

our next step is to give a description of what you're going to build. Is it an update to an existing product? Is it a new product? Is it an add-on to a product you've already created?

- These are important to describe upfront, so everyone knows what you're building.
- We should also describe why you're building it and who it's for.

User Needs

- User needs — or user classes and characteristics — are critical. You'll need to define who is going to use the product and how.
- You'll have primary and secondary users who will use the product on a regular basis. You may also need to define the needs of a separate buyer of the product (who may not be a primary/secondary user). And, for example, if you're building a medical device, you'll need to describe the patient's needs.

Assumptions and Dependencies

- There might be factors that impact your ability to fulfill the requirements outlined in your SRS. What are those factors?
- Are there any assumptions you're making with the SRS that could turn out to be false? You should include those here, as well
- Finally, you should note if your project is dependent on any external factors. This might include software components you're reusing from another project.

- 4. Detail Your Specific Requirements

The next section is key for your development team. This is where you detail the specific requirements for building your product.

Functional Requirements

- Functional requirements are essential to building your product.

External Interface Requirements

- External interface requirements are types of functional requirements. They're important for embedded systems. And they outline how your product will interface with other components.

There are several types of interfaces you may have requirements for, including:

- User
- Hardware
- Software
- Communications

System Features

- System features are types of functional requirements. These are features that are required in order for a system to function.

Other Nonfunctional Requirements

- Nonfunctional requirements can be just as important as functional ones.

These include:

- Performance
- Safety
- Security
- Quality
- The importance of this type of requirement may vary depending on your industry. Safety requirements, for example, will be critical in the medical device industry.

5. Get Approval for the SRS

- Once you've completed the SRS, you'll need to get it approved by key stakeholders. And everyone should be reviewing the latest version of the document.

Specification review

- A review of the Software Requirements Specification (and/or prototype) is conducted by both the software developer and the customer.
- Because the specification forms the foundation of the development phase, extreme care should be taken in conducting the review.
- The review is first conducted at a macroscopic level; that is, reviewers attempt to ensure that the specification is complete, consistent, and accurate when the overall information, functional, and behavioral domains are considered. However, to fully explore each of these domains, the review becomes more detailed, examining not only broad descriptions but the way in which requirements are worded(expressed).

- Once the review is complete, the Software Requirements Specification is "signed off" by both the customer and the developer.
- The specification becomes a "contract" for software development.
- Requests for changes in requirements after the specification is finalized will not be eliminated. But the customer should note that each after the fact change is an extension of software scope and therefore can increase cost and/or protract the schedule.

Analysis modeling

The analysis model is the first technical representation of a system.
Analysis modeling is about creating data,functional and behavioral models of the system.
Analysis modeling uses a combination of text and diagrammatic forms to depict
Requirements for data function and behavior in a way that is relatively easy to understand
And more straightforward to review for correctness completeness and consistency.

A software engineer(analyst) builds the model using requirements provided by the Customer.
the **structured analysis** (a classical modeling method) and **object-oriented analysis** are two main methods
for analysis modeling.

Analysis Model is a technical representation of the system.
It acts as a link between system description and design model.
In Analysis Modeling, information,
behavior and functions
of the system is defined and translated into the architecture, component and interface level design in the design modeling.

What are the steps?

- Data,functional and behavioral requirements are modeled using a number of different diagrammatic formats.
- Data modeling defines data objects,attributes and relationships.
- Functional modeling indicates how data are transformed within a system.
- Behavioral modeling depicts the impact of events.
- Once preliminary models are created,they are refined and analyzed to assess their clarity,completeness,and consistency

Data object descriptions, entity relationship diagram, data flow diagram,state transition diagram Process specification and control specification are created as a part of analysis modeling Activity.

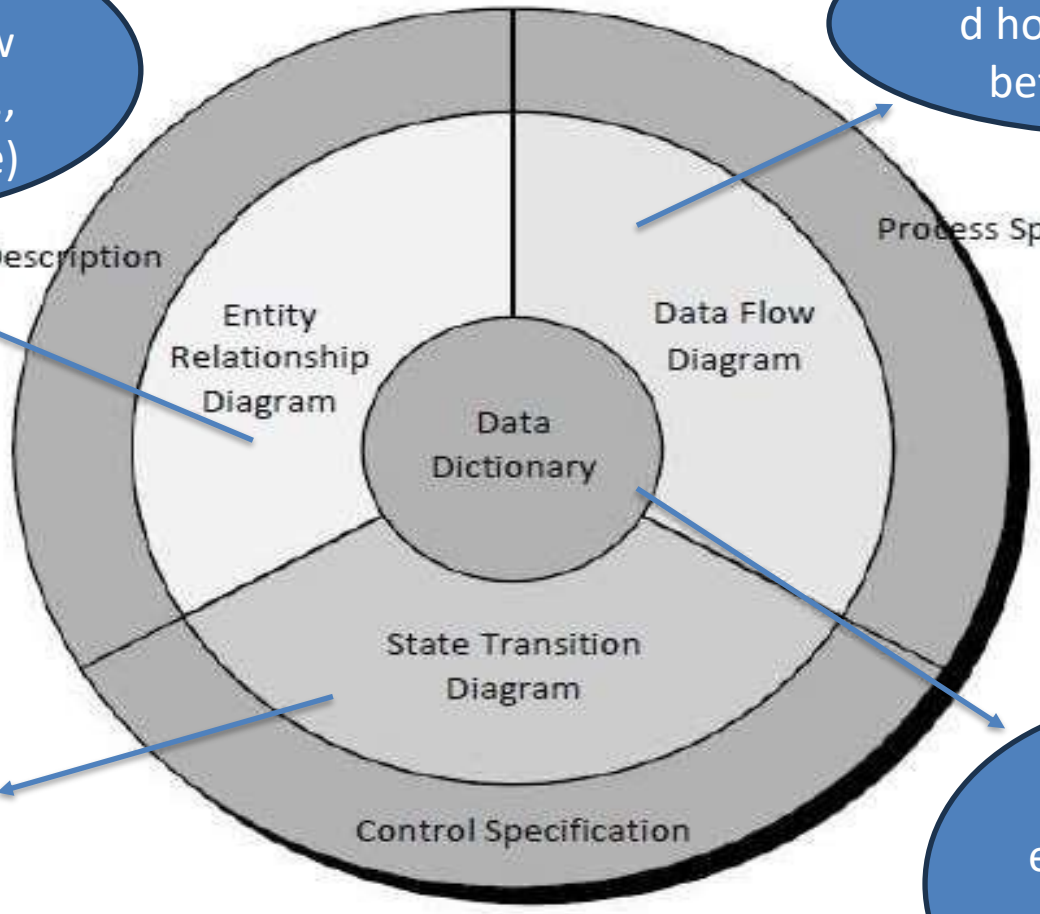
The elements of the analysis model

- Structured analysis is a model building activity. The analysis model must achieve three primary objectives:
- to describe what the customer requires
- to establish a basis for the development of software design
- to define a set of requirements that can be validated once the software is developed.

To accomplish these objectives the analysis model derived during structured analysis Takes the form as the diagram shown in next slide.

shows *data objects* and how they relate (e.g., Student–Course)

shows *processes* an d how data moves between them.



shows how a system/object changes *states* (Idle → Processing → Completed)

A "glossary" that defines every data item (fields, files, messages) used in the system

The structure of the analysis model

- At the core of the model lies the data dictionary(a repository that contains descriptions of all the data objects consumed or produced by the software.
- The entity relationship diagram(ERD)depicts relationships between data objects. The ERD is the notation that is used to conduct the data modeling activity.the attribute of each data object noted in the ERD can be described using a data object description.
- The data flow diagram serves two purposes:

1. to provide an indication of how data are transformed as they move through the system.
2. to depict the functions that transforms the data flow.

The DFD provides additional information that is used during the analysis of the information domain and serves as a basis for the modeling of function

- The state transition diagram(STD) indicates how the system behaves as a consequence of external events. The STD serves as the basis for behavioral modeling.

A description of each function presented in the DFD is contained in a process Specification.

A description of control aspects presented in the STD is contained in a control Specification.

Data modeling

- the process of documenting a complex software system design as an easily understood diagram, using text and symbols to represent the way data needs to flow.
- The diagram can be used as a blueprint for the construction of software.
- Traditionally, have been built during the analysis phase of a project to ensure that the requirements for a new application are fully understood.
- can be thought of as a flowchart that illustrates the relationships between data.
- The data model consists of three interrelated pieces of information: the data object,

the attributes that describes the data object and the relationships that connect data

objects to one another.

- Data modeling methods make the use of Entity Relationship Diagram(ERD).
- ERD enables the software engineers to identify data objects and their relationship using a graphical notation.
- ERD defines all the data that are entered,stored,transformed and produced within an application.

Data modeling identifies following items

- Data objects:

The data object is the representation of composite information.

- The composite information means an object has a number of different properties or attribute.
- For example, Height is a single value so it is not a valid data object, but dimensions contain the height, the width and depth these are defined as an object.

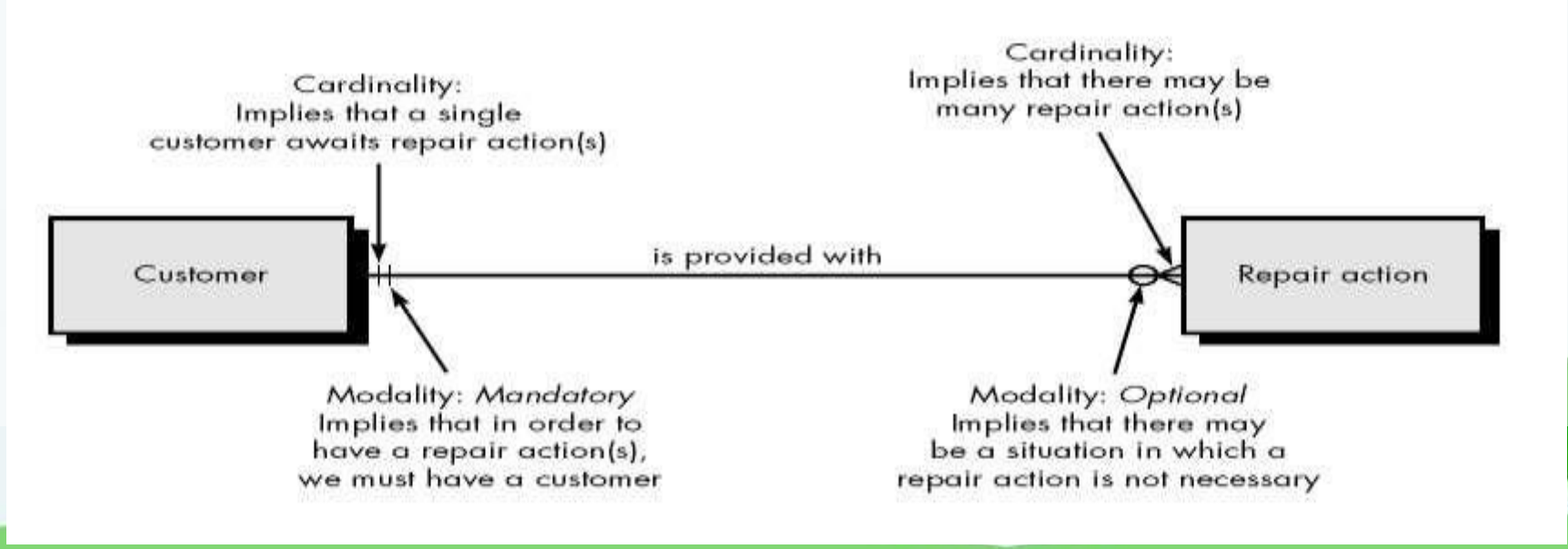
- Attributes:

- Attributes define the properties of dataEach of the data object has a set of attributes.
- Data object has the following characteristics:
- Name an instance of the data object.
- Describe the instance.
- Make reference to another instance in another table.

- # Relationship
- Relationship shows the relationship between data objects and how they are related to each other.
- # Cardinality
- Cardinality state the number of events of one object related to the number of events of another object.
- The cardinality expressed as:
- One to one (1:1)
- One event of an object is related to one event of another object.
- For example, one employee has only one ID.
- One to many (1:N)
- One event of an object is related to many events.
- For example, One collage has many departments.
- Many to many(M:N)
- Many events of one object are related to many events of another object.
- For example, many customer place order for many products.

Modality

- If an event relationship is an optional then the modality of relationship is zero.
- If an event of relationship is compulsory then modality of relationship is one.



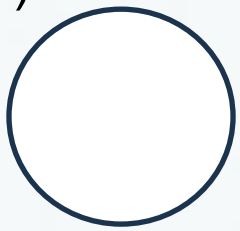
Functional Modeling and Information Flow

- Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms; applies hardware, software, and human elements to transform it; and produces output in a variety of forms.
- Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage. The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system.
- Output may light a single LED or produce a 200-page report. In effect, we can create a flow model for any computer-based system, regardless of size and complexity.

- A computer- based system is represented as an information transform .
- A rectangle is used to **represent an external entity**; that is, a system element (e.g., hardware, a person, another program) or another system that produces information for transformation by the software or receives information produced by the software.

- A circle (sometimes called a bubble) **represents a process(function) or transform** that is applied to data (or control) and changes it in some way.

Eg:Apply Payment
Calculate Commission
Verify Order

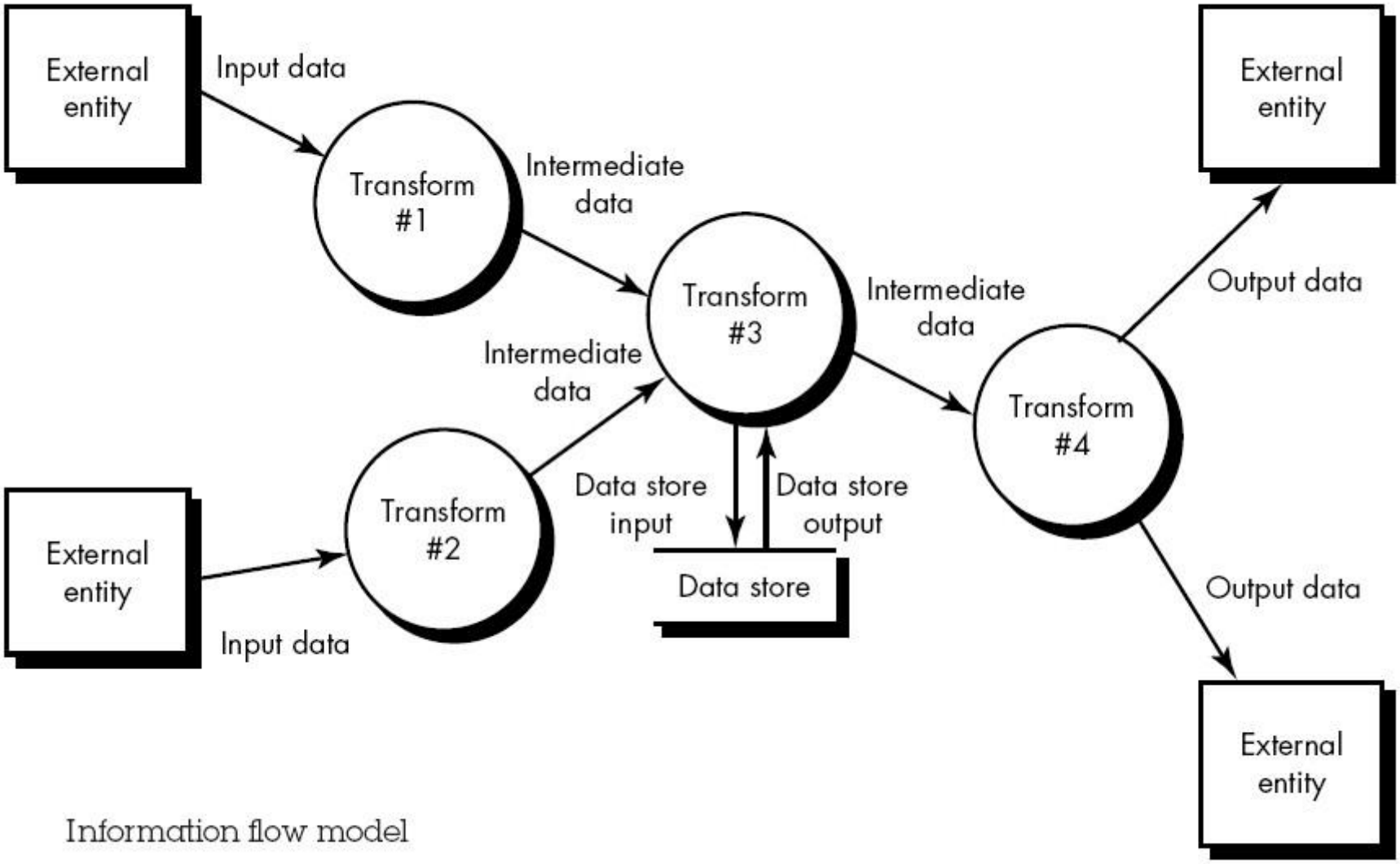


- An arrow **represents information flow** from one or more data items (data objects). All arrows on a data flow diagram should be labeled. The double line represents a data store—stored information that is used by the software.

Eg. Customer_info (LastName, FirstName, SS#, Tel #, etc.)
Order_info (OrderId, Item#, OrderDate, CustomerID, etc.).

A data store or data repository is used in a data-flow diagram to represent a situation when the system must retain data because one or more processes need to use the stored data in a later time.

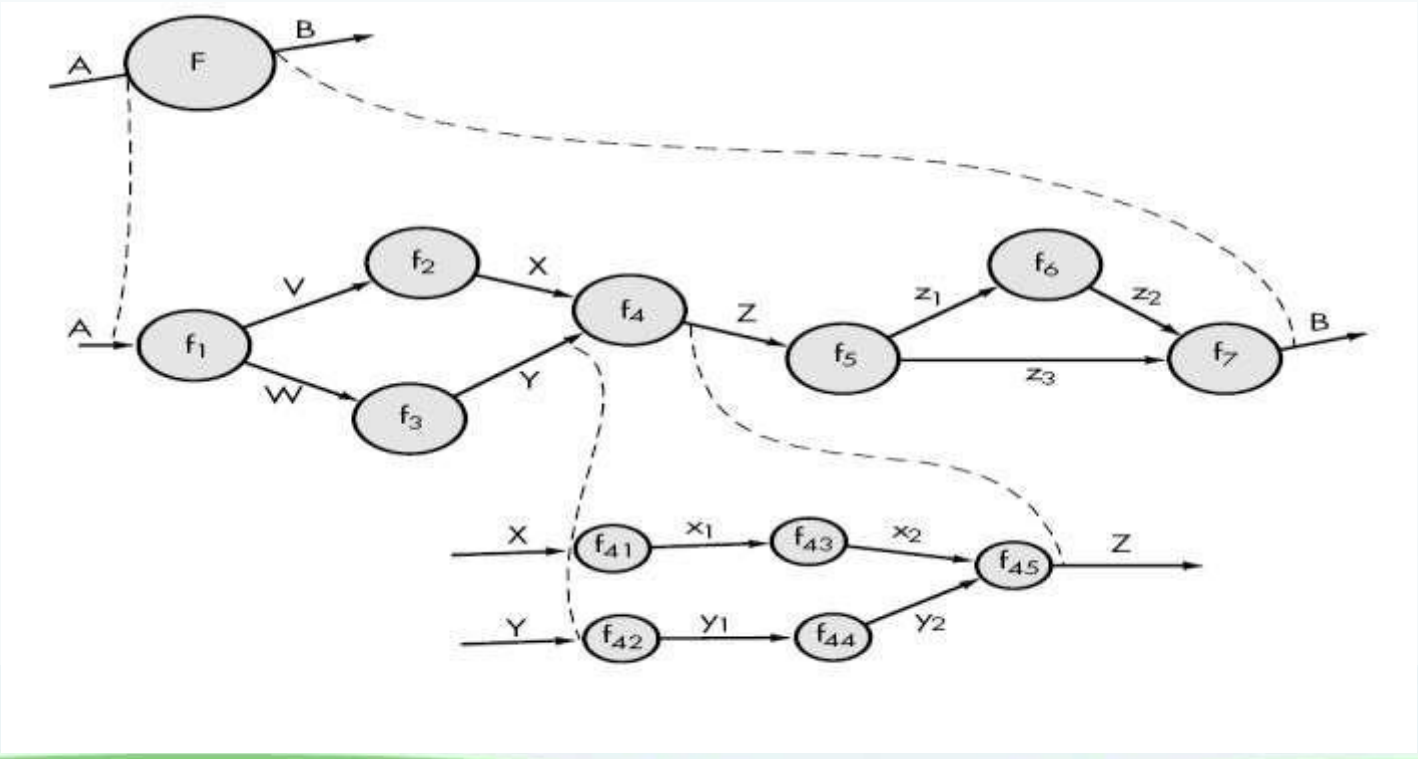
- The simplicity of DFD notation is one reason why structured analysis techniques are widely used.



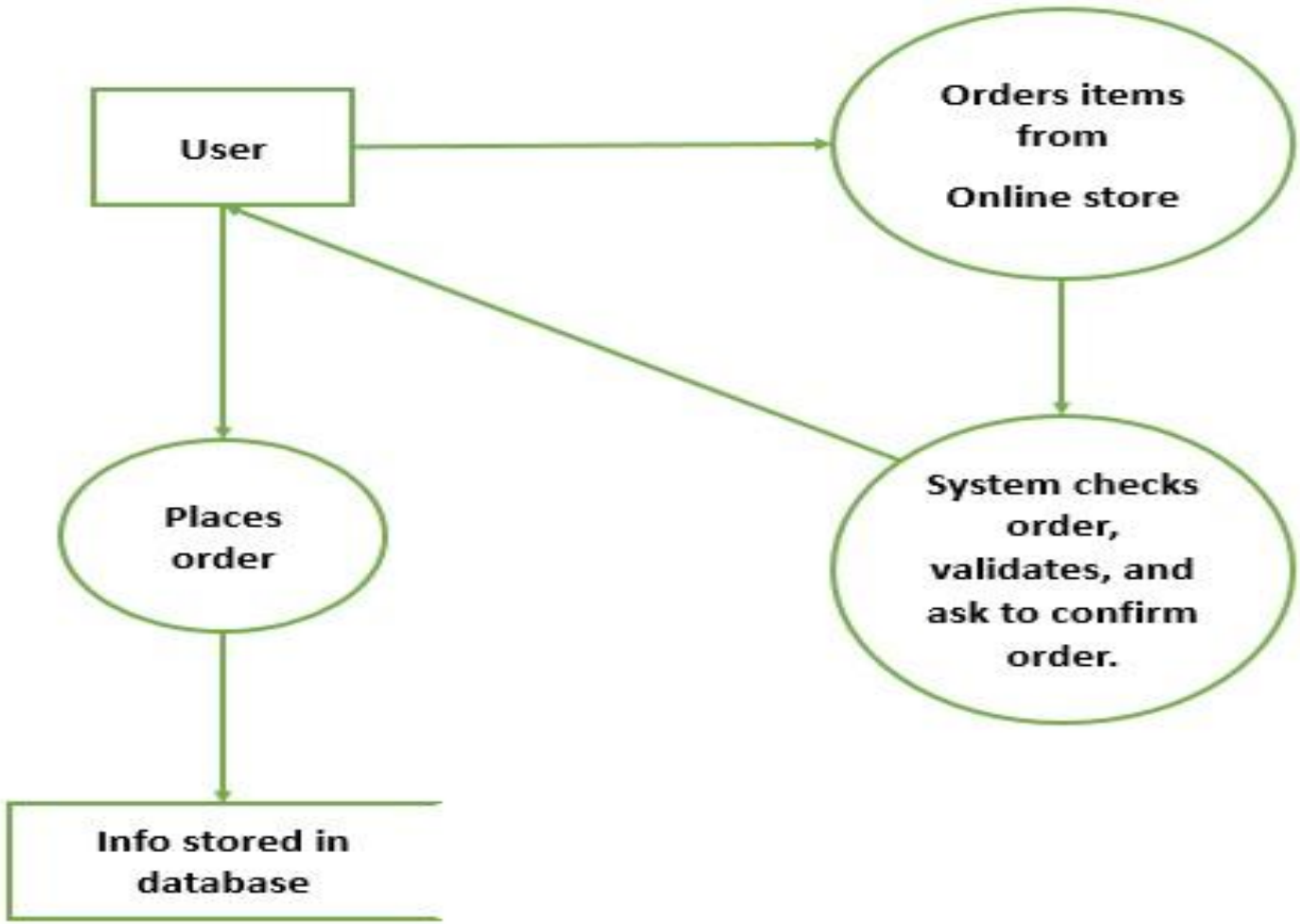
Information flow model

Data flow diagram

- As information moves through software, it is modified by a series of transformations. A data flow diagram is a graphical representation that depicts information flow and the transforms that are applied as data move from input to output. The basic form of a data flow diagram, also known as a data flow graph or a bubble chart, is illustrated in the figure below.



- The data flow diagram may be used to represent a system or software at any level of abstraction.
- In fact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Therefore, the DFD provides a mechanism for functional modeling as well as information flow modeling.
- In so doing, it satisfies the second operational analysis principle (i.e., creating a functional model)

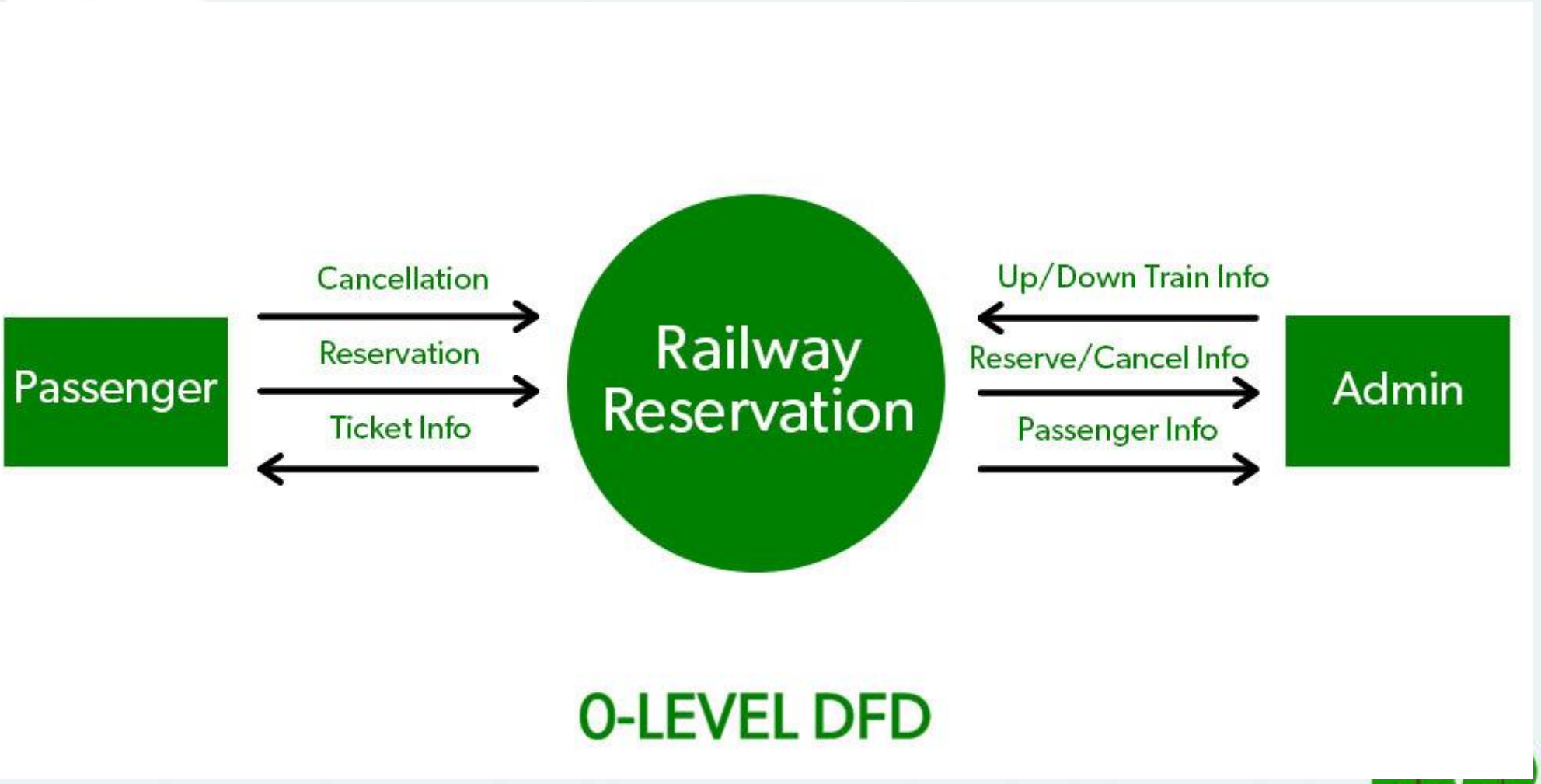

User
Orders items from
Online store
Places order
System checks order, validates, and ask to confirm order.
Info stored in database

- A level 0 DFD, also called a fundamental system model or a context model, represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively.
- Additional processes (bubbles) and information flow paths are represented as the level 0 DFD is partitioned to reveal more detail. For example, a level 1 DFD might contain five or six bubbles with interconnecting arrows.
- Each of the processes represented at level 1 is a sub function of the overall system depicted in the context model.

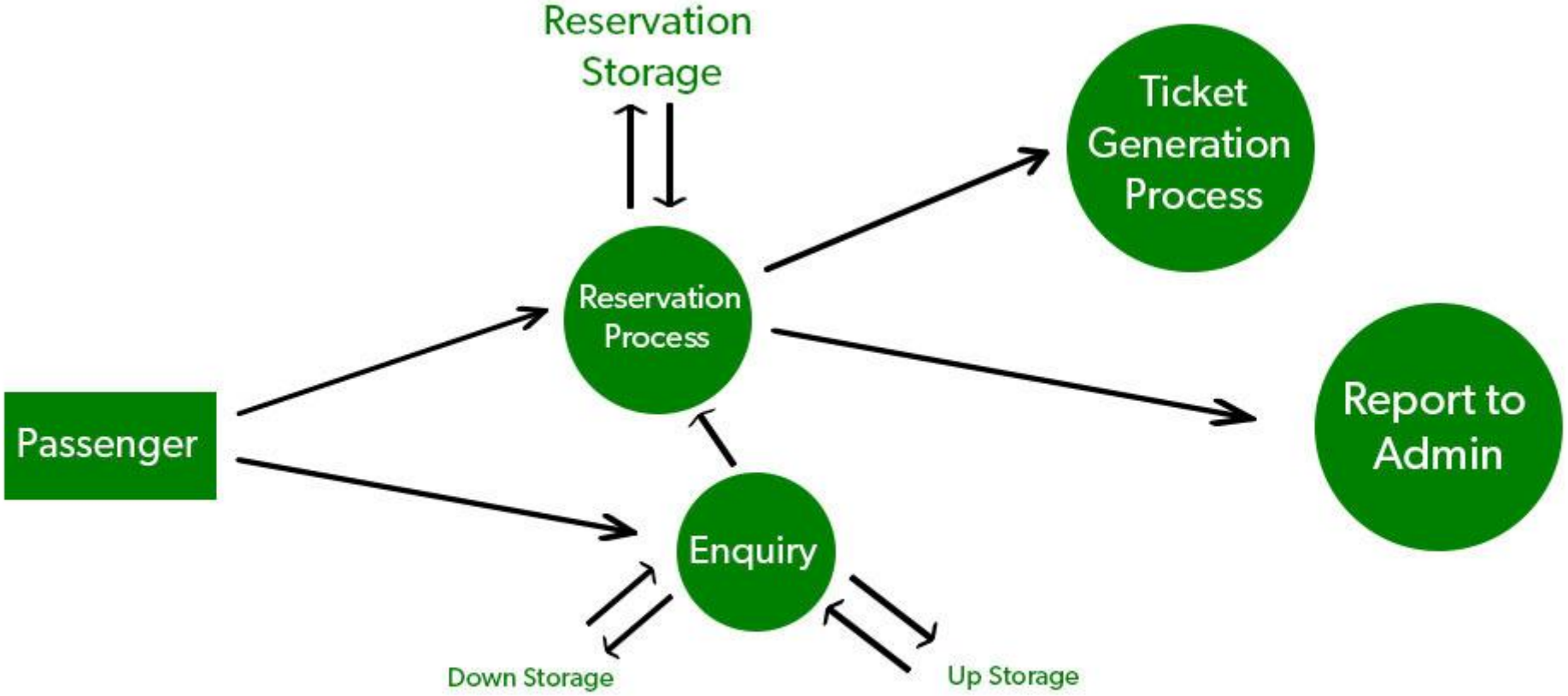
- As we noted earlier, each of the bubbles may be refined or layered to depict more detail. A fundamental model for system F indicates the primary input is A and ultimate output is B. We refine the F model into transforms f1 to f7.
- Note that information flow continuity must be maintained; that is, input and output to each refinement must remain the same. This concept, sometimes called balancing, is essential for the development of consistent models.
- Further refinement of f4 depicts detail in the form of transforms f41 to f45. Again, the input (X, Y) and output (Z) remain unchanged.

- DFD graphical notation must be augmented with descriptive text. A process specification (PSPEC) can be used to specify the processing details implied by a bubble within a DFD.
- The process specification describes the input to a function, the algorithm that is applied to transform the input, and the output that is produced. In addition, the PSPEC indicates restrictions and limitations imposed on the process (function), performance characteristics that are relevant to the process, and design constraints that may influence the way in which the process will be implemented.

Level 0 is the highest-level Data Flow Diagram (DFD), which provides an overview of the entire system. It shows the major processes, data flows, and data stores in the system, without providing any details about the internal workings of these processes.
also known as a **Context Diagram**.

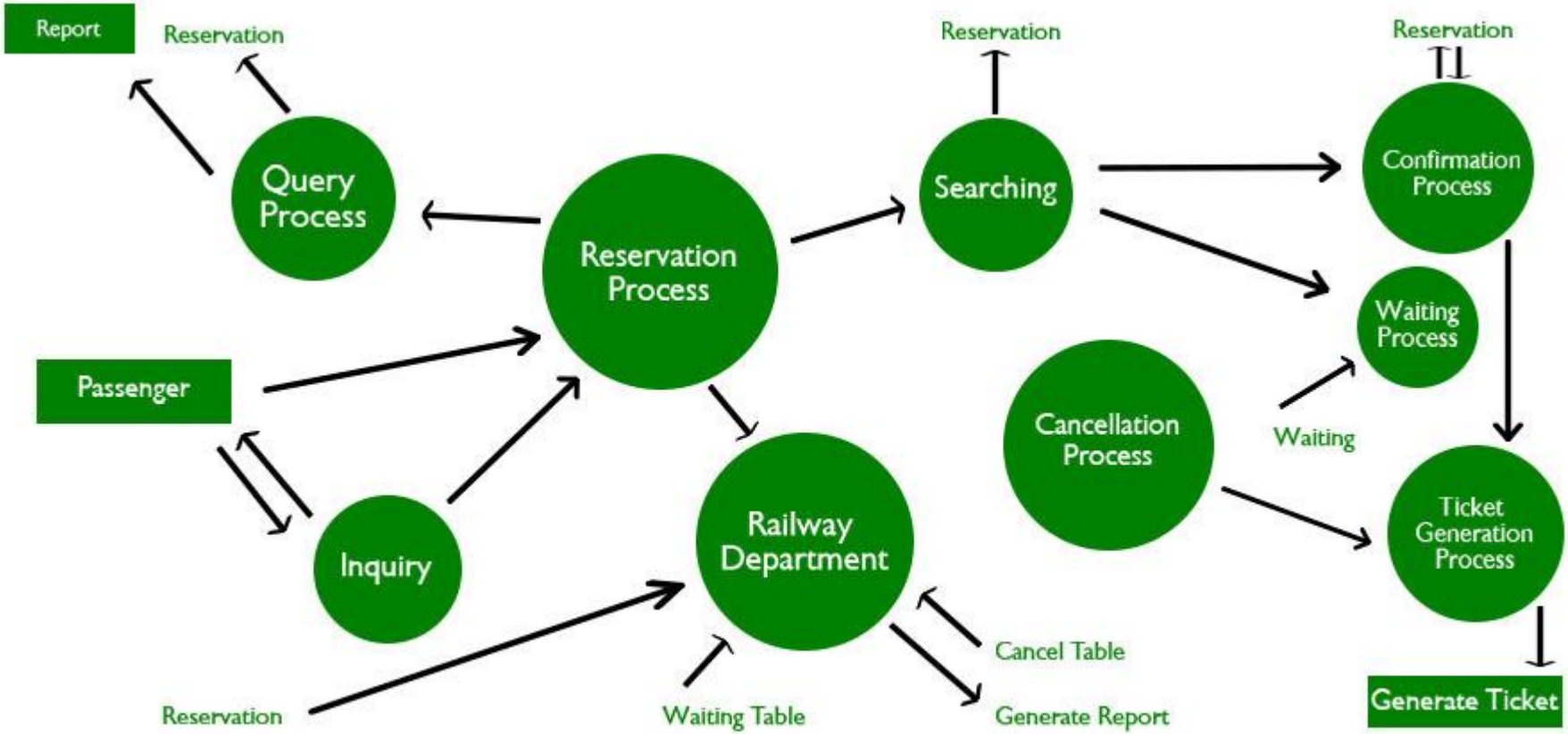


1-Level provides a **more detailed view** of the system by breaking down the major processes identified in the level 0 Data Flow Diagram (DFD) into **sub-processes**. Each sub-process is depicted as a separate process on the level 1 Data Flow Diagram (DFD). The data flows and data stores associated with each **sub-process** are also shown.



1-LEVEL DFD

Each process on the level 3 DFD is depicted with a detailed description of its input, processing, and output. The **data flows** and **data stores** associated with each process are also shown, providing a comprehensive breakdown. **Level 3 DFD** helps in the **detailed analysis** of a specific process and is often used for system-level development and technical documentation.

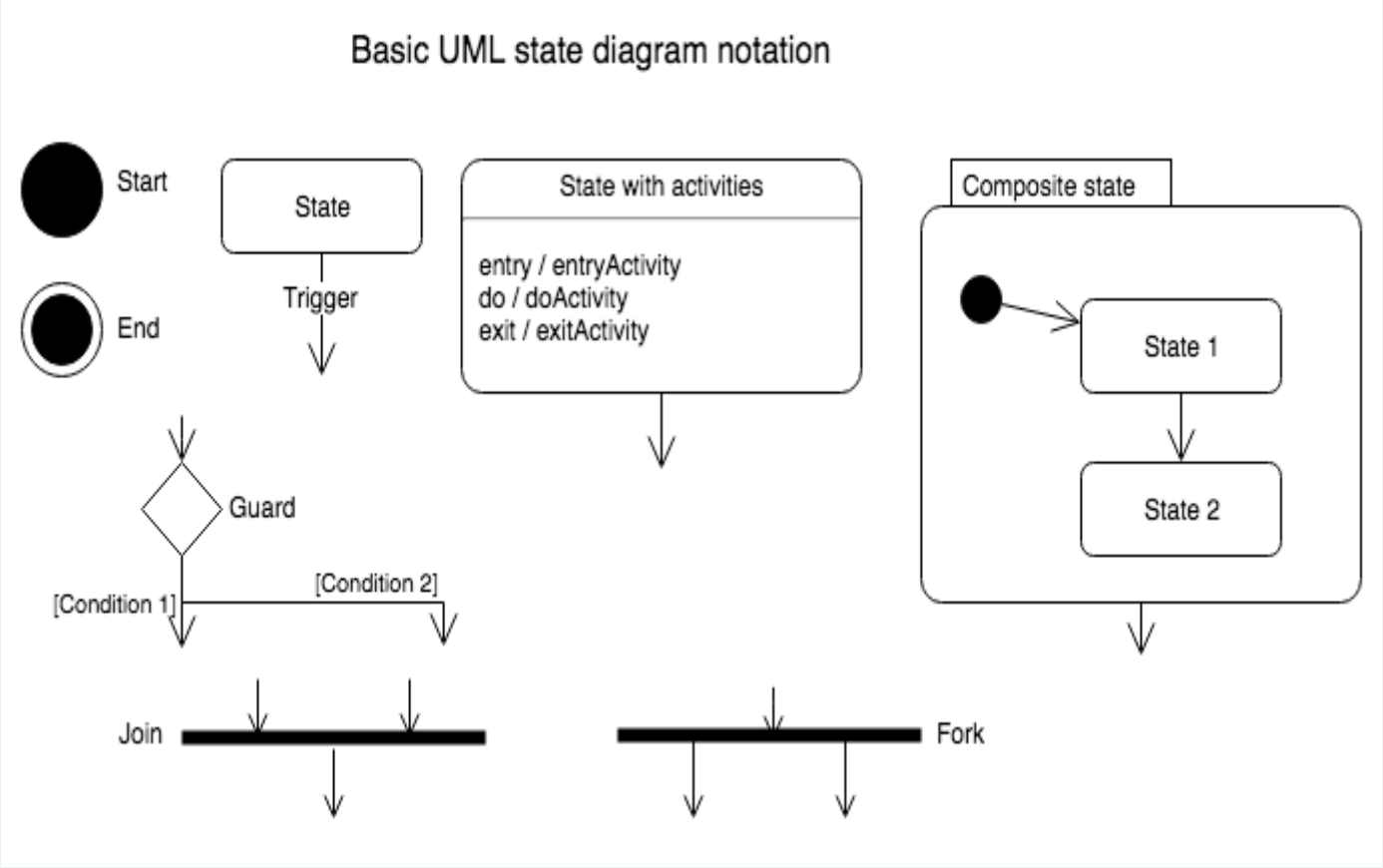


2-LEVEL DFD

Behavioral modeling

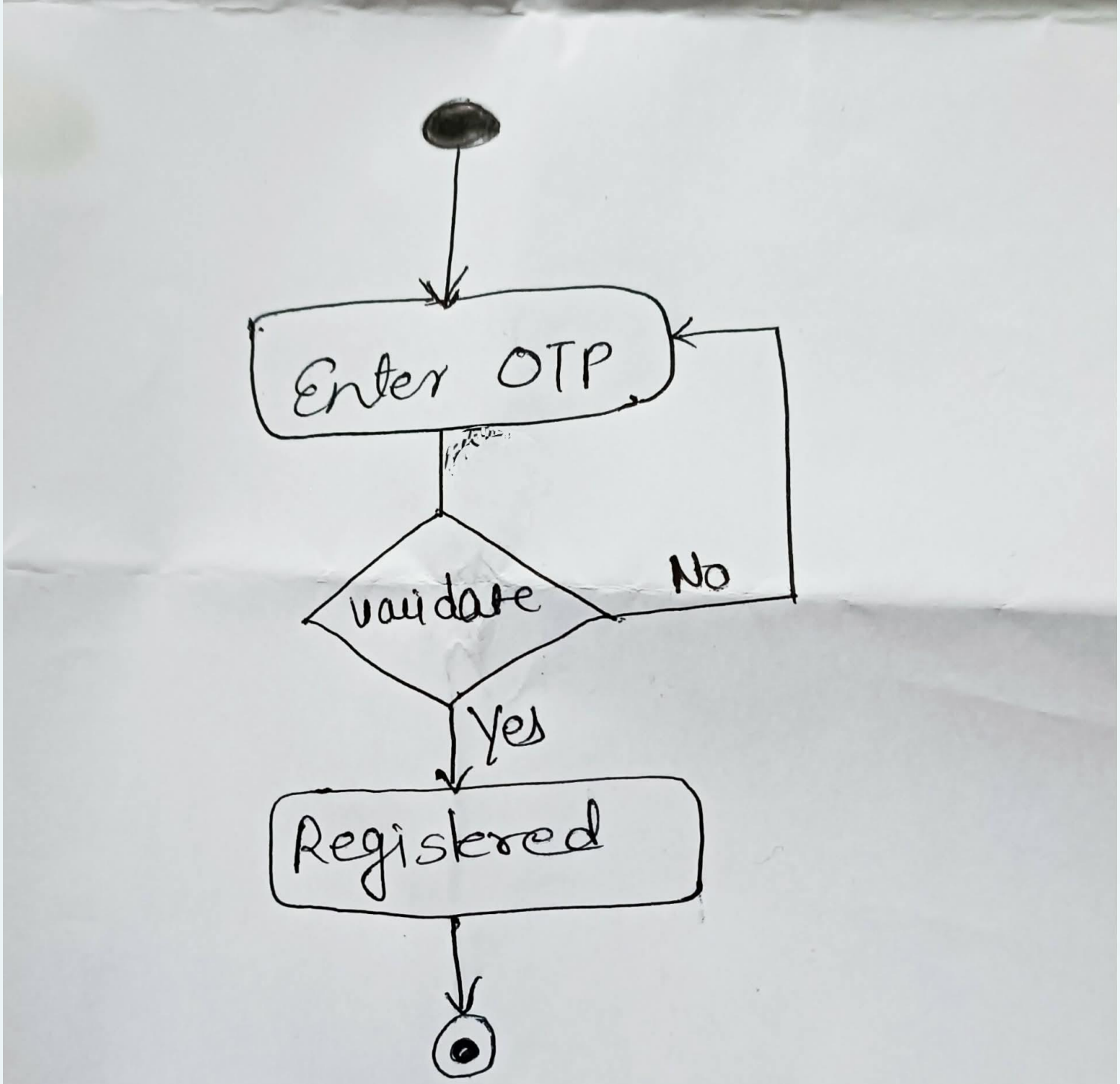
- Behavioral modeling is an operational principle for all requirements analysis methods but, only extended versions of structured analysis provided a notation for this type of modeling.
- The state transition diagram represents the behavior of a system by depicting its states and the events that cause the system to change state.
- STD indicates what actions(e.g process activation)are taken as a consequence of particular event.
- State transition diagram indicates how the system move from state to state.

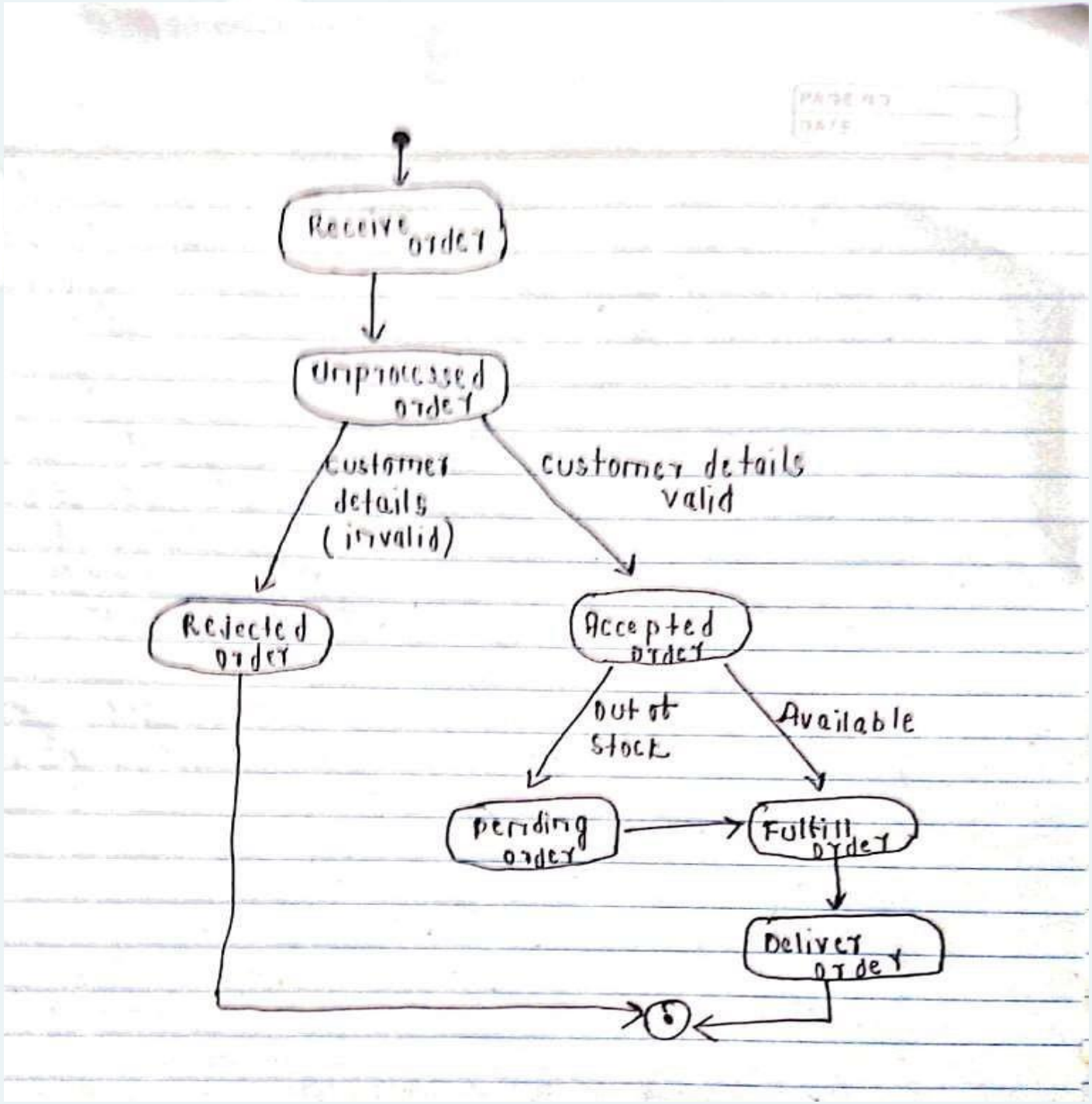
Behavioral modeling

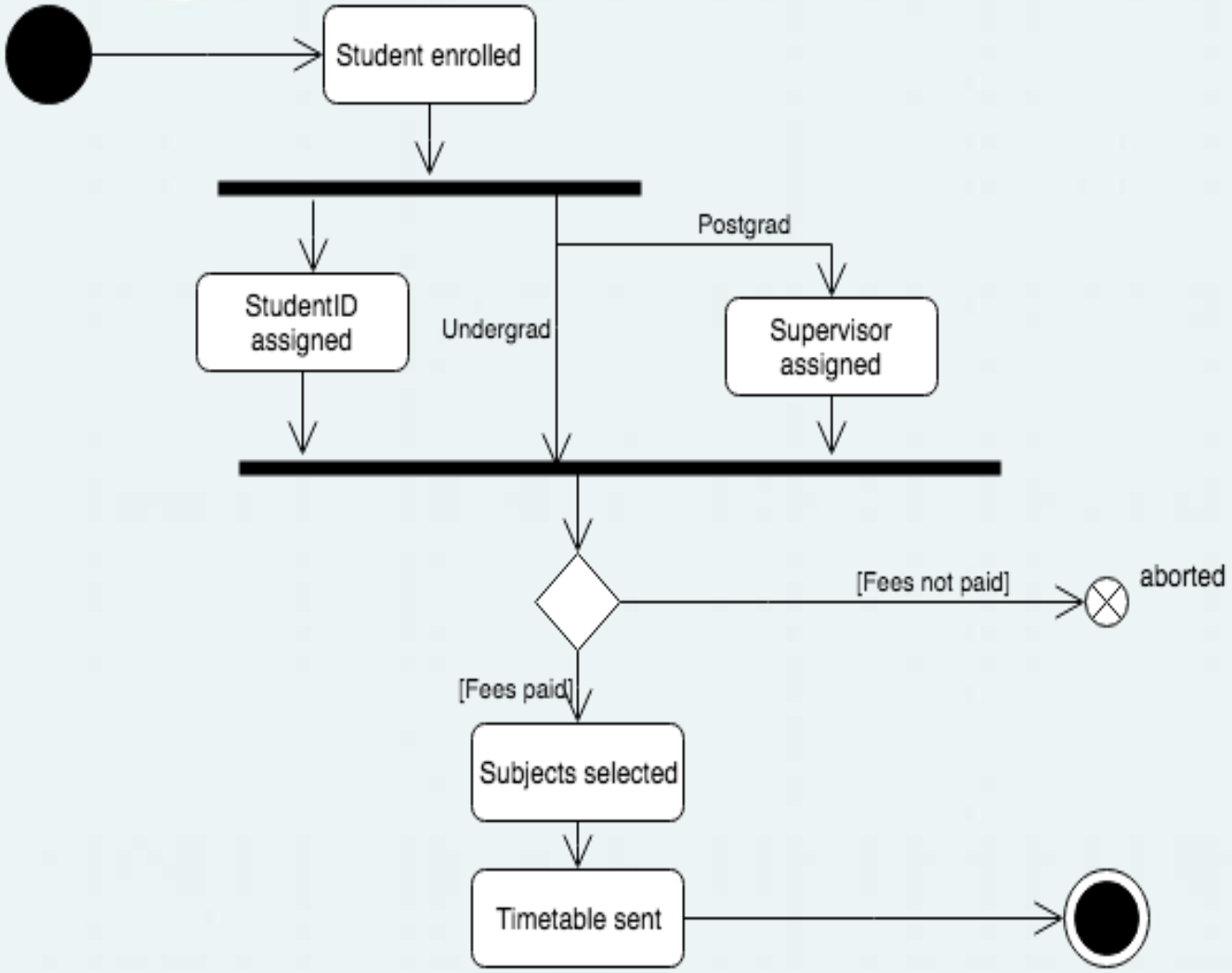


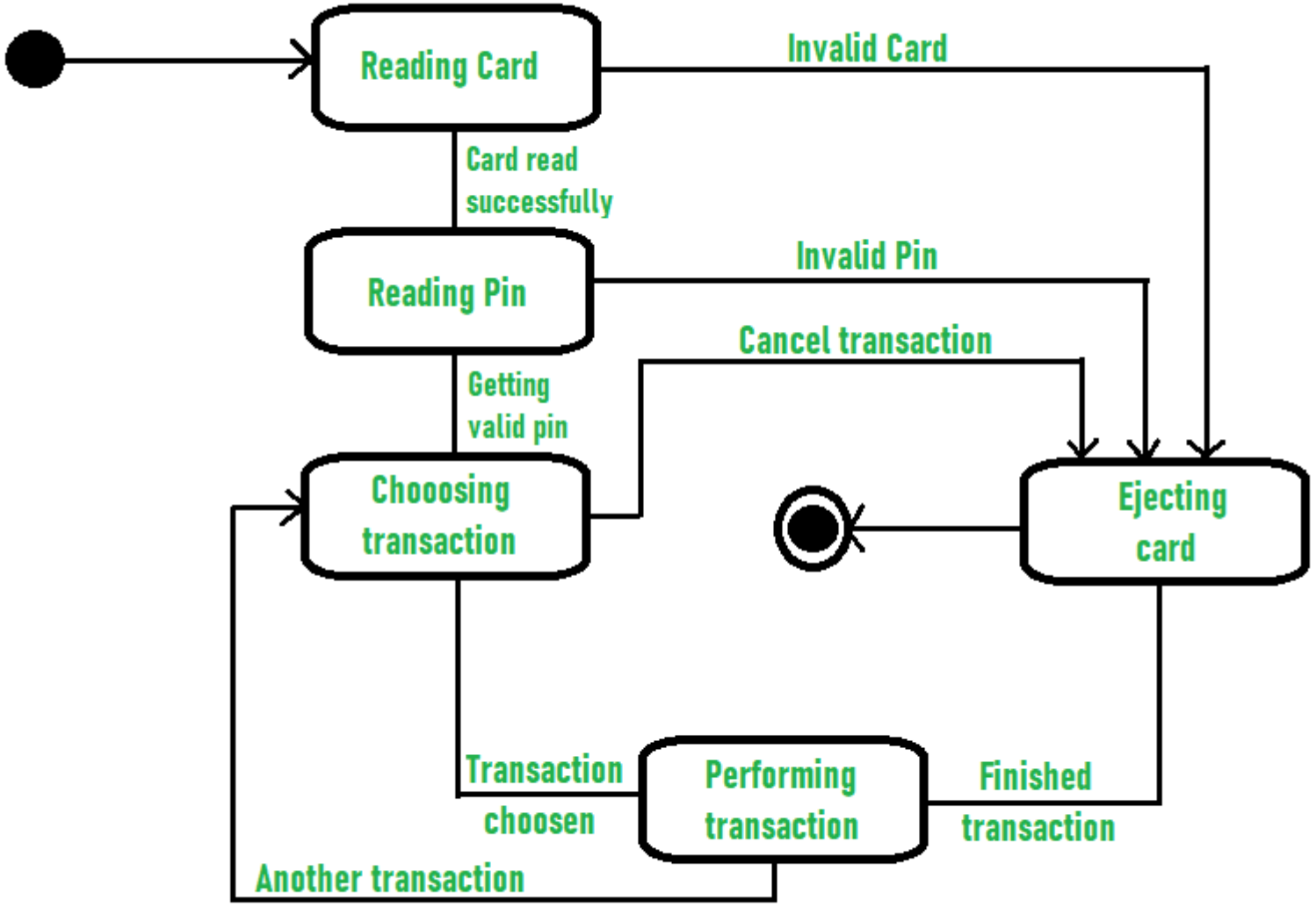
Behavioral modeling

- **Initial State**: A solid, filled black circle, indicating the starting point.
- **Final State**: A solid black circle with a concentric ring (or a circle with an 'X'), showing the process termination.
- **State**: A rectangle with rounded corners, containing the state's name and sometimes internal activities.
- **Transition**: An arrow from one state to another, labeled with the triggering event.
- **Self-Transition**: An arrow looping back to the same state, for events that don't change the state.


Enter OTP
validate
No
Yes
Registered


Receive order
Unprocessed order
customer details (invalid)
customer details valid
Rejected order
Accepted order
out of stock
Available
pending order
Fulfill order
Deliver order

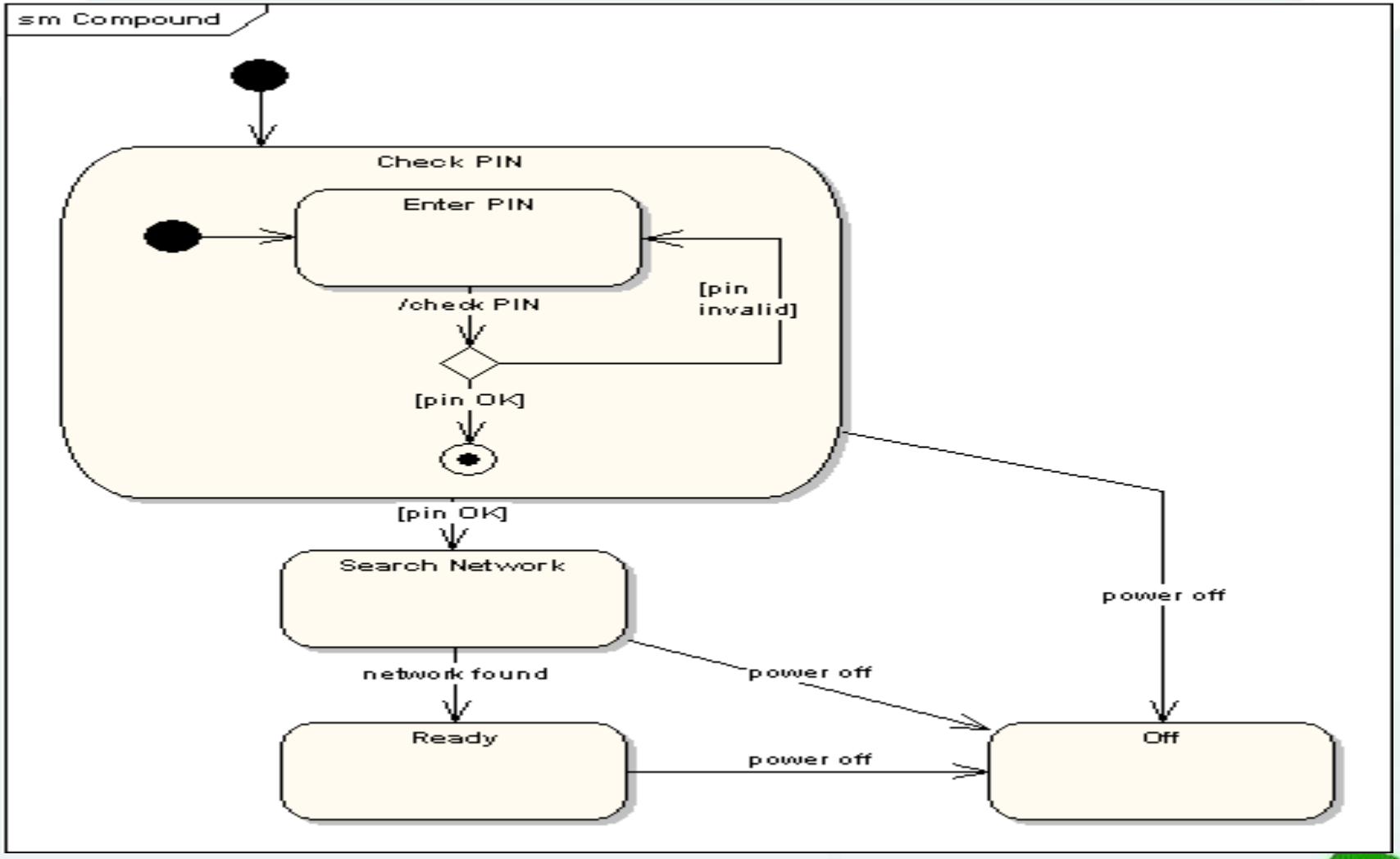

Student enrolled
Postgrad
StudentID assigned
Undergrad
Supervisor assigned
[Fees not paid]
aborted
[Fees paid]
Subjects selected
Timetable sent



State Transition Diagram for ATM System

Compound States

A state machine diagram may include sub-machine diagrams, as in the example below.



The data Dictionary

- A **data dictionary** is a central repository defining all data elements in a system (like names, types, rules), acting as metadata to ensure clarity, consistency, and understanding for developers and stakeholders, detailing what data is used, its format, meaning, and usage across the application or database

The data Dictionary

- The data dictionary is an organized listing of all data elements that are relevant to the system,with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and [even] intermediate calculations.
- Today, the data dictionary is always implemented as part of a CASE "structured analysis and design tool." Although the format of dictionaries varies from tool to tool, most contain the following information:

Name—the primary name of the data or control item, the data store or an external entity.

- Alias(Synonyms)—other names used for the first entry.
- Where-used/how-used—a listing of the processes that use the data or control item and

how it is used (e.g., input to the process, output from the process, as a store, as an external entity.

- Content description—a notation for representing content.
- Supplementary information—other information about data types, preset values (if known), restrictions or limitations, and so forth.

Once a data object or control item name and its aliases are entered into the data dictionary, consistency in naming can be enforced. That is, if an analysis team member decides to name a newly derived data item xyz, but xyz is already in the dictionary, the CASE tool supporting the dictionary posts a warning to indicate duplicate names. This improves the consistency of the analysis model and helps to reduce errors.

“Where-used/how-used” information is recorded automatically from the flow models When a dictionary entry is created, the CASE tool scans DFDs and CFDs to determine which processes use the data or control information and how it is used.

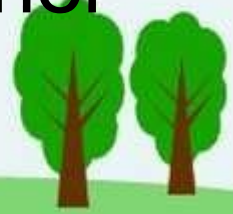
The notation used to develop a content description is noted as:

Notations in Data Dictionary

Data Construct	Notation	Meaning
Composition	=	Is composed of
Sequence	+	Represents AND
Selection	[|]	Represents OR
Repetition	$\{\}^n$	Represents n repetitions or repetition for n times
Parentheses	( )	Optional data that may or may not be present
Comment	*...*	Delimits a comment or commented information

The notation enables a software engineer to represent composite data in one of the three fundamental ways that it can be constructed:

1. As a sequence of data items.
2. As a selection from among a set of data items.
3. As a repeated grouping of data items. Each data item entry that is represented as part of a sequence, selection, or repetition may itself be another composite data item that needs further refinement within the dictionary.



Example:telephone number

Name:telephone number

Aliases:none

Where used/how used:asses against setup(output)

Dial phone(input)

Description:telephone number=[local number/long distance number]

Local number=prefix+access number

Long distance number=1+area code+local number

Area code=[800|888|561]

prefix=*a three digit number that never starts with 0 or1*

Access number=*any four number string*

For large computer based systems the data dictionary grows rapidly in size and complexity. in fact it is extremely difficult to maintain a dictionary manually.

For this reason CASE tools should be used.

Example: Let us understand this by taking an example of the reservation system. In the reservation system, "**passenger**" is a data item whose information is available on the data dictionary as follows:

Keys	Values
Name	Passenger
Aliases	None
Where or how it's used?	Passenger's query (input) Ticket (output)
Description	• Passenger = Passenger_name + Passenger_address • Passenger_name = Passenger_lastname + Passenger_firstname + Passenger_middle_initial • Passenger_address = Local_address + Community_address + Zip_code • Local_address = House_number + street_name + Apartment_number • Community_Address = City_name + State_name

Safe home system

The SafeHome security function enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through the Internet, a PC, or a control panel. During installation, the SafeHome PC is used to program and configure the system. Each sensor is assigned a number and type, a master password is programmed for arming and disarming the system, and telephone number(s) are input for dialing when a sensor event occurs. When a sensor event is recognized, the software invokes an audible alarm attached to the system. After a delay time that is specified by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, provides information about the location, reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until a telephone connection is obtained. The homeowner receives security information via a control panel, the PC, or a browser, collectively called an interface. The interface displays prompting messages and system status information on the control panel, the PC, or the browser window. Homeowner interaction takes the following form…

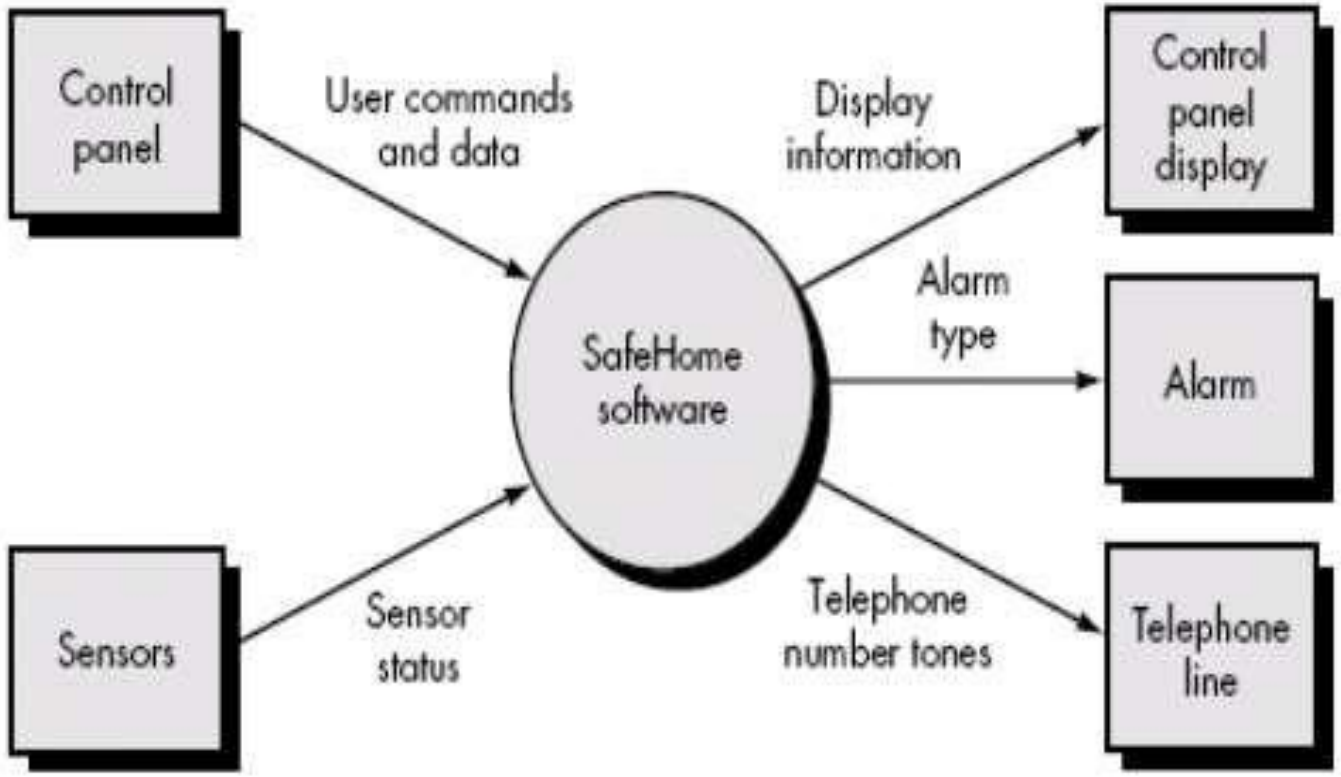
Context Diagram

Find the people who send data into the system

Find the people who get data out of the system.

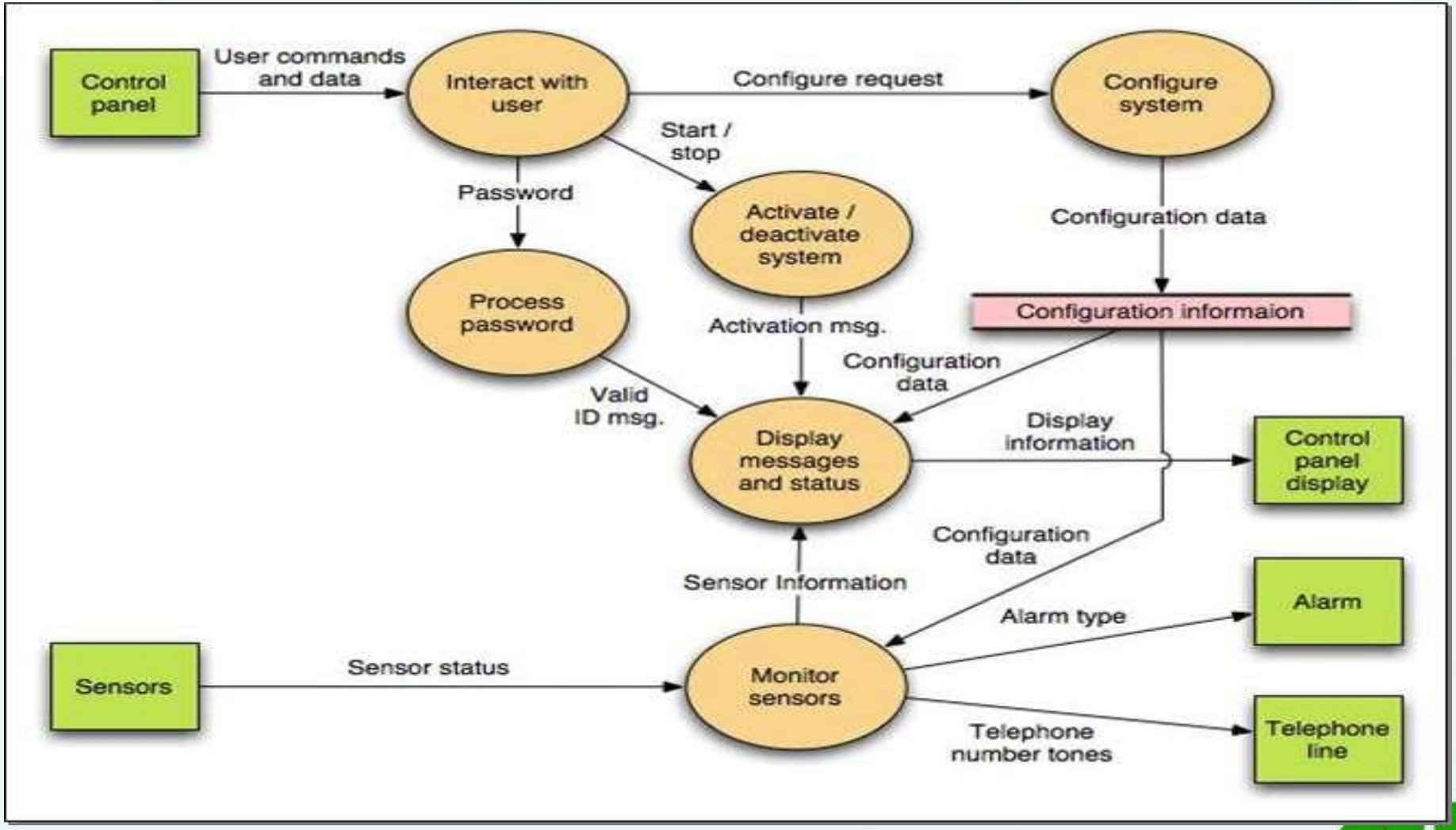
The only data you need is data that is transformed or sent completely out of the system – not data that is handled by an operator within the system.

Context-level DFD for the *SafeHome* security System

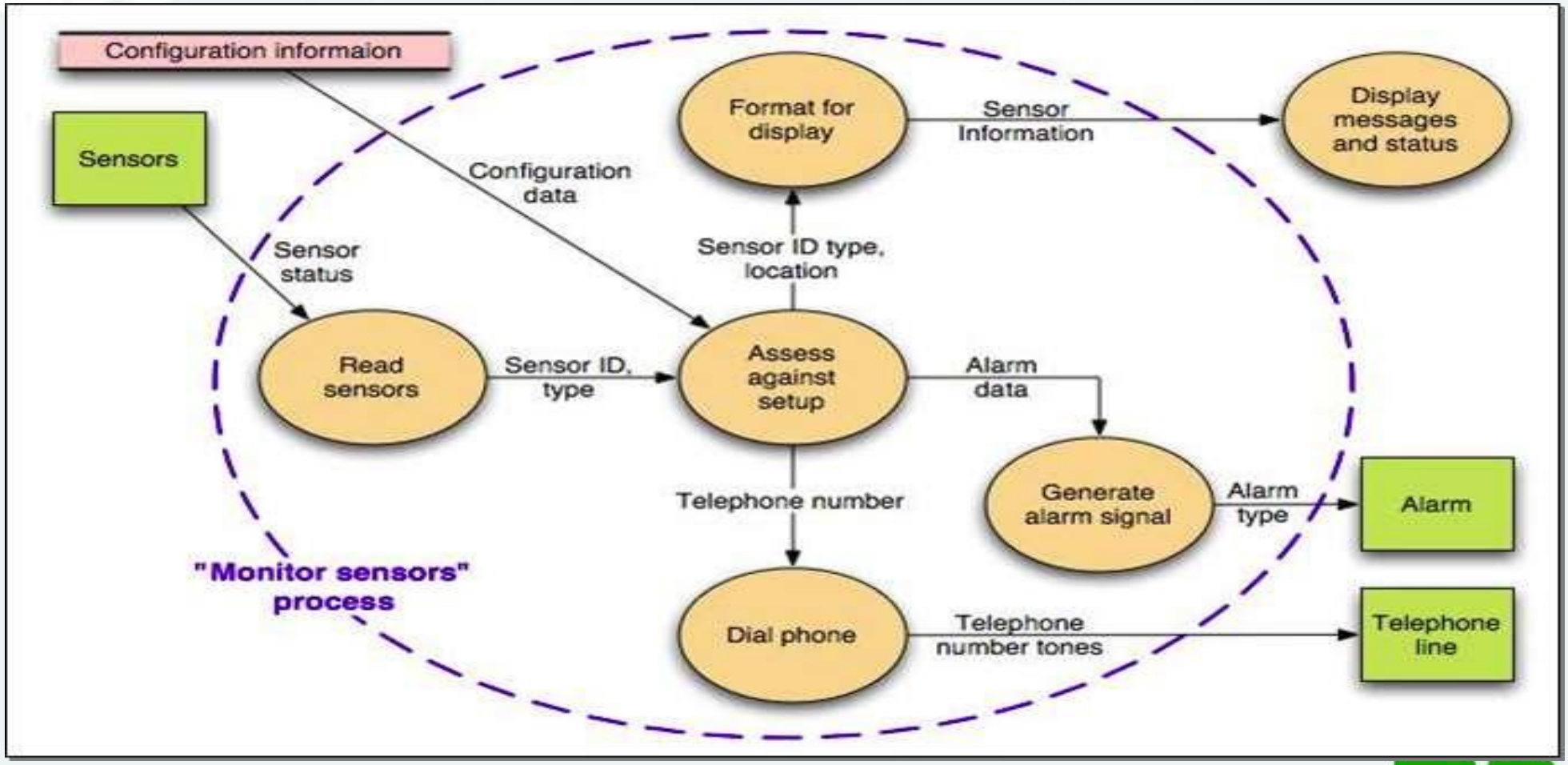


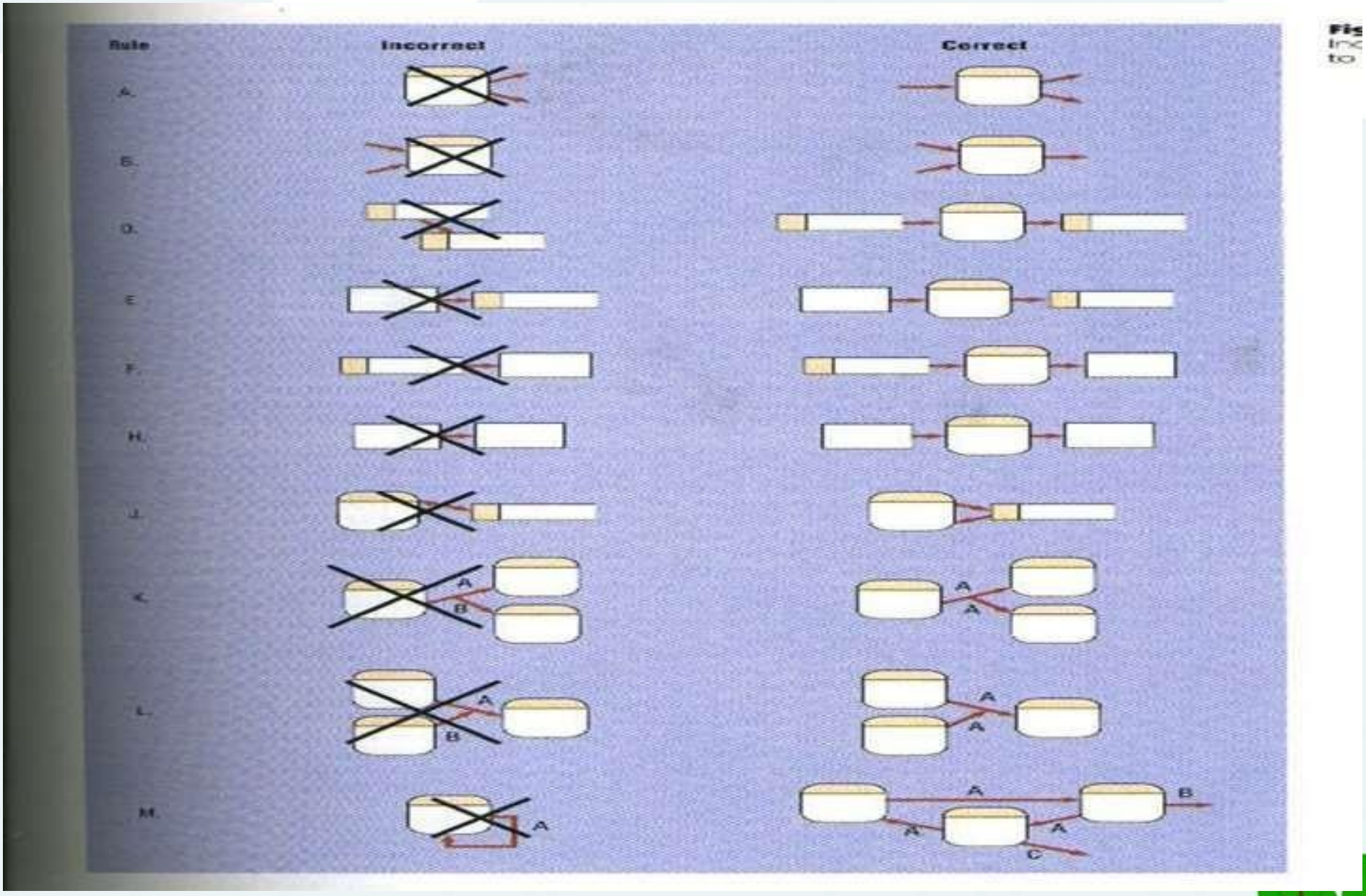
- The *SafeHome* security function enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through the Internet, a PC, or a control panel.
- During installation, the *SafeHome* PC is used to program and configure the system. Each sensor is assigned a number and type, a master password is programmed for arming and disarming the system, and telephone number(s) are input for dialing when a sensor event occurs.
- When a sensor event is recognized, the software invokes an audible alarm attached to the system. After a delay time that is specified by the homeowner during system configuration activities, the software *dials* a telephone number of a monitoring service, provides information about the location, reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until a telephone connection is obtained.
- The homeowner receives security information via a control panel, the PC, or a browser, collectively called an interface. The interface displays prompting messages and system status information on the control panel, the PC, or the browser window. Homeowner interaction takes the following form…

Level 1 DFD


Control panel
User commands and data
Interact with user
Configure request
Configure system
Start / stop
Password
Activate / deactivate system
Configuration data
Process password
Configuration informaion
Activation msg.
Configuration data
Valid ID msg.
Display messages and status
Display information
Control panel display
Configuration data
Sensor Information
Alarm
Alarm type
Sensors
Sensor status
Monitor sensors
Telephone number tones
Telephone line

Level 2 DFD


Configuration informaion
Sensors
Format for display
Sensor Information
Display messages and status
Configuration data
Sensor status
Sensor ID type, location
Read sensors
Sensor ID, type
Assess against setup
Alarm data
Generate alarm signal
Alarm type
Alarm
Telephone number
"Monitor sensors" process
Dial phone
Telephone number tones
Telephone line


Rule
Incorrect
Correct
A.
B.
D.
E
F.
H.
J.
K.
L.
M.
A
B
A
A
A
A
A
B
A
A
C

Old question

A travel agency arranges holidays for customers. Bookings are made directly by customers. When a customer makes an approach, the reservations clerk selects appropriate flight details and hotel details from lists which are regularly updated. The details are entered onto a Provisional Booking file. The customer must confirm this booking within three days by sending a deposit of 10% of the costs. On receipt of the deposit, Reservations transfer the details from the Provisional Bookings file to the Full Bookings file. Four weeks before the flight is due, Accounts send an invoice to the customer for the remaining costs. Accounts notify Customer services when the full payment is received, and Customer Services then send tickets and joining instructions to the customer.

A travel agency arranges holidays for customers. Bookings are made directly by customers. When a customer makes an approach, the reservations clerk selects appropriate flight details and hotel details from lists which are regularly updated. The details are entered onto a Provisional Booking file.
The customer must confirm this booking within three days by sending a deposit of 10% of the costs. On receipt of the deposit, Reservations transfer the details from the Provisional Bookings file to the Full Bookings file.
Four weeks before the flight is due, Accounts send an invoice to the customer for the remaining costs. Accounts notify Customer services when the full payment is received, and Customer Services then send tickets and joining instructions to the customer.

	Grammatical analysis (Noun, Verb)	Allocation
	Customers	External Entity
	Reservation Clerk ,Select ,Flight details , hotel details , lists	Internal Entity,Process,flow,flow, Data store
	Updated	Data store (Internal)
	Entered Provisional booking file	Process Data store
	Confirm ,Send ,Deposit	Process,Process, output flow
	Reservations, transfer, provisional bookings file	Internal entity,process,data store
	Full booking file	Data store
	Accounts , send, Invoice	Internal entity, process, data flaw
	Notify	Process
	Customer services, full payment ,received	Internal entity, data flow , process
	Send, Tickets, Instructions	Process,flow,flow

External entities

Customer

Processes

Enter

Confirm

Send1 ,Send2 ,Send3

Received

Notify

Data flows

Flight details

Hotel details

Deposit

Invoice

Ticket

Instructions

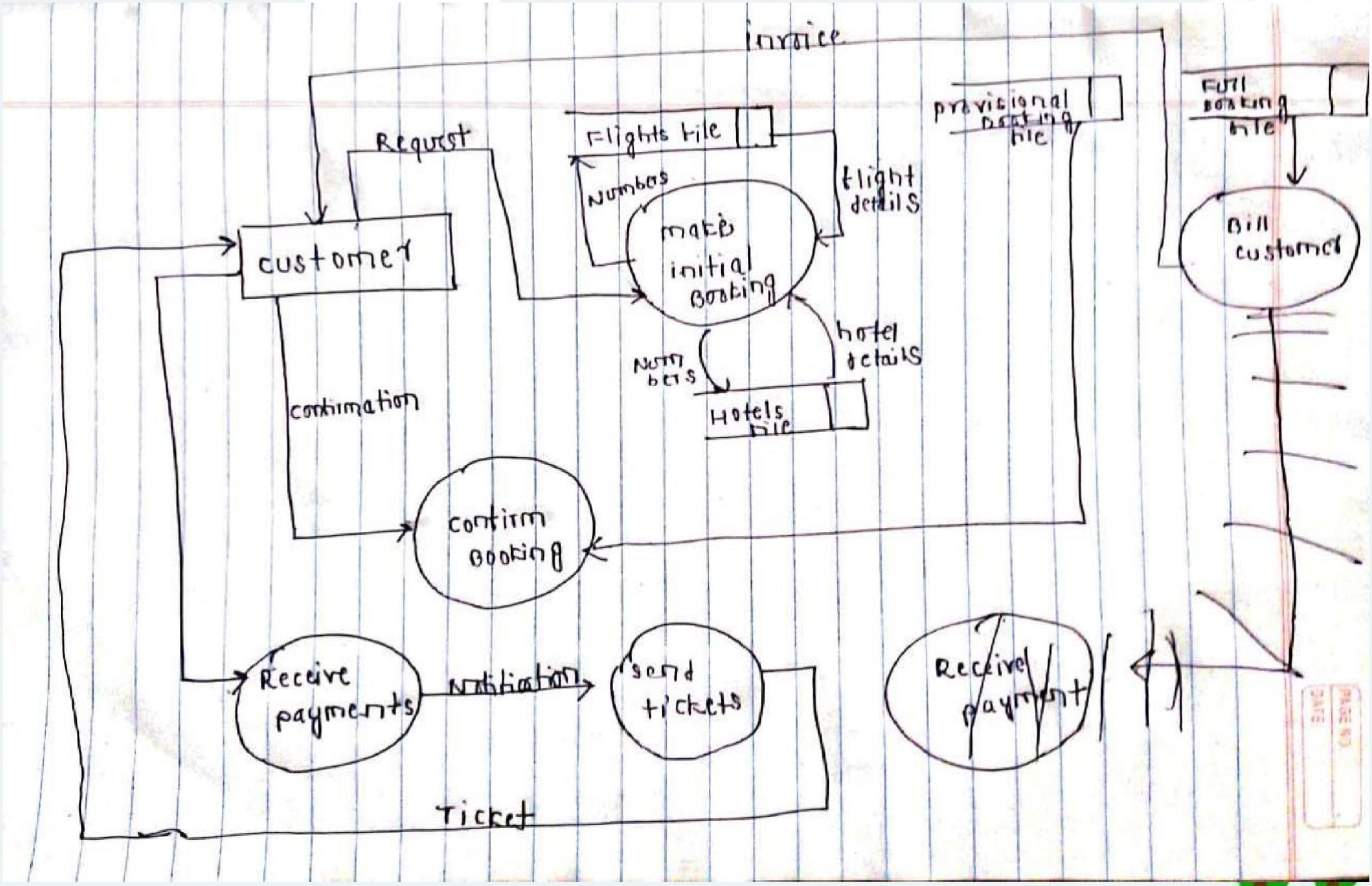
Data stores

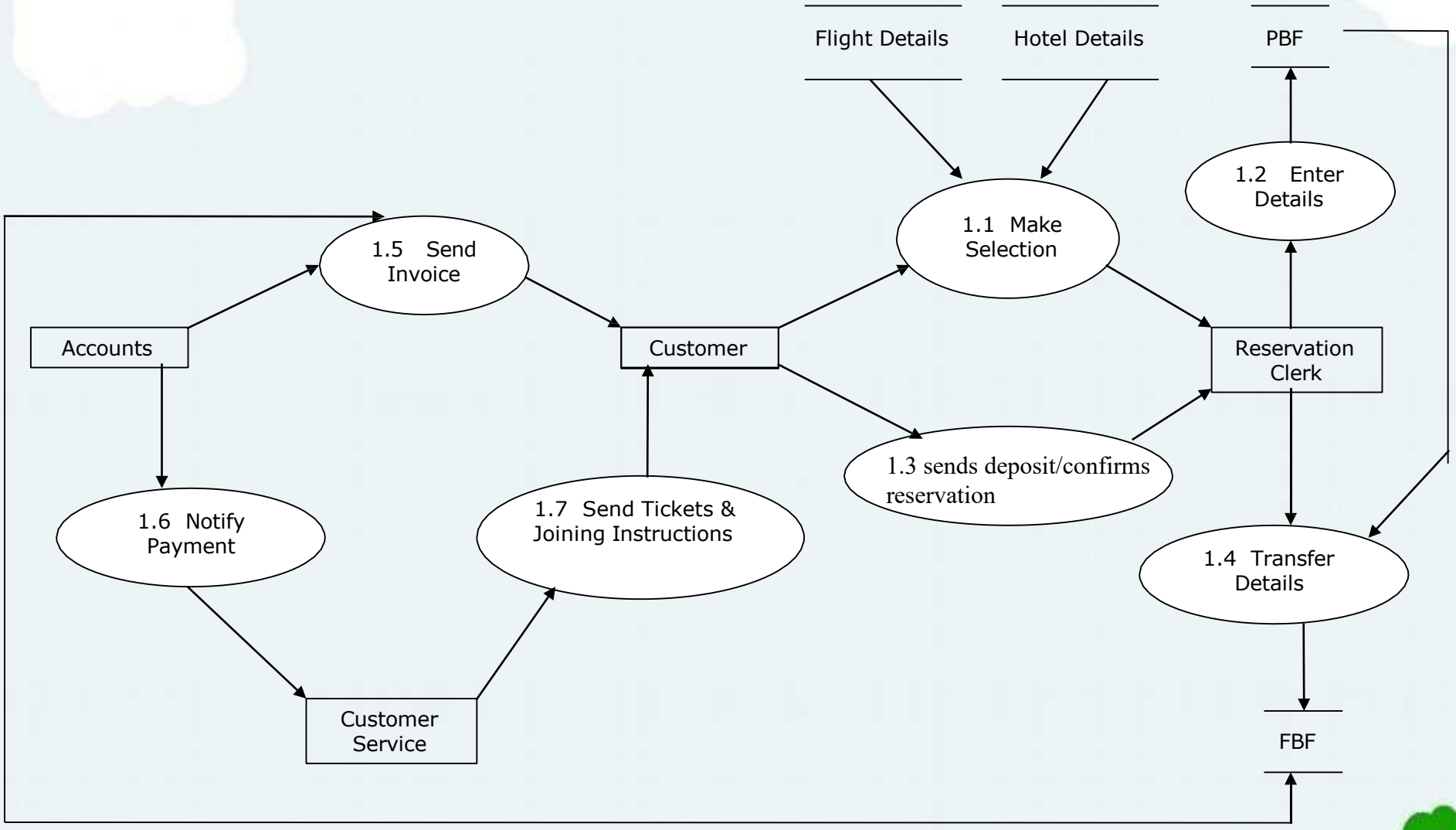
Lists :

Hotel list +Flight list

Provisional Booking file

Full booking file


invoice
Request
Flights file
provisional booking file
Full booking file
Numbers
flight details
customer
make initial Booking
Bill customer
hotel details
Numbers
Hotels file
confirmation
confirm Booking
Receive payments
notification
send tickets
Receive payment
Ticket


Flight Details
Hotel Details
PBF
1.2 Enter Details
1.1 Make Selection
1.5 Send Invoice
Accounts
Customer
Reservation Clerk
1.3 sends deposit/confirms reservation
1.6 Notify Payment
1.7 Send Tickets & Joining Instructions
1.4 Transfer Details
Customer Service
FBF

Entity Relationship Diagram

An entity relationship diagram is a type of flowchart that illustrates how “entities” such as people, object or concepts related to each other within a system

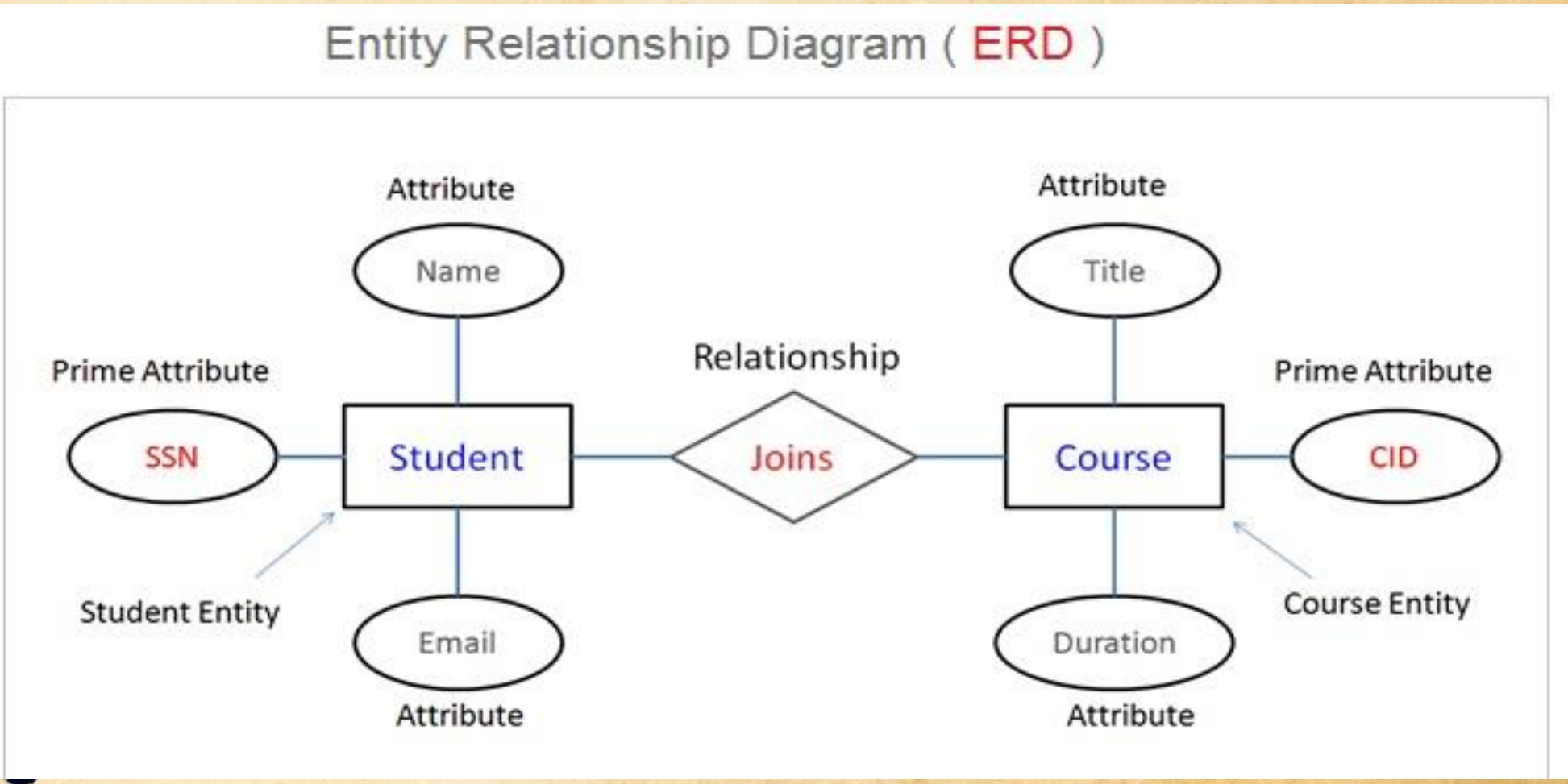
An ER diagram is a means of visualizing how the information a system produces is related. The main components of an ERD:

Entities, which are represented by rectangles. An entity is an object or concept about which you want to store information. e.g BOOK,CAR,STUDENT

Attributes: which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.

Relationship: which are represented by diamond shapes, show how two entities share information in the database.

Entity Relationship Diagram (ERD)



ENTITY RELATIONSHIP DIAGRAM

2015 FALL 3.a.

A Country Bus Company owns a number of busses. Each bus is allocated to a particular route, although some routes may have several busses. Each route passes through a number of towns. One or more drivers are allocated to each stage of a route, which corresponds to a journey through some or all of the towns on a route. Some of the towns have a garage where busses are kept and each of the busses are identified by the registration number and can carry different numbers of passengers, since the vehicles vary in size and can be single or double-decked. Each route is identified by a route number and information is available on the average number of passengers carried per day for each route. Drivers have an employee number, name, address, and sometimes a telephone number.

Identify the entities from the above problem and model it into a ER diagram

Entities

- **Bus - Company owns busses and will hold information about them.**
- **Route - Buses travel on routes and will need described.**
- **Town - Buses pass through towns and need to know about them**
- **Driver - Company employs drivers, personnel will hold their data.**
- **Stage - Routes are made up of stages**
- **Garage - Garage houses buses, and need to know where they are.**

Relationships

- A bus is allocated to a route and a route may have several buses.

Bus-route (m:1) is serviced by

- A route comprises of one or more stages.

route-stage (1:m) comprises

- One or more drivers are allocated to each stage.

driver-stage (m:1) is allocated

- A stage passes through some or all of the towns on a route.

stage-town (m:n) passes-through

- A route passes through some or all of the towns

route-town (m:n) passes-through

- Some of the towns have a garage

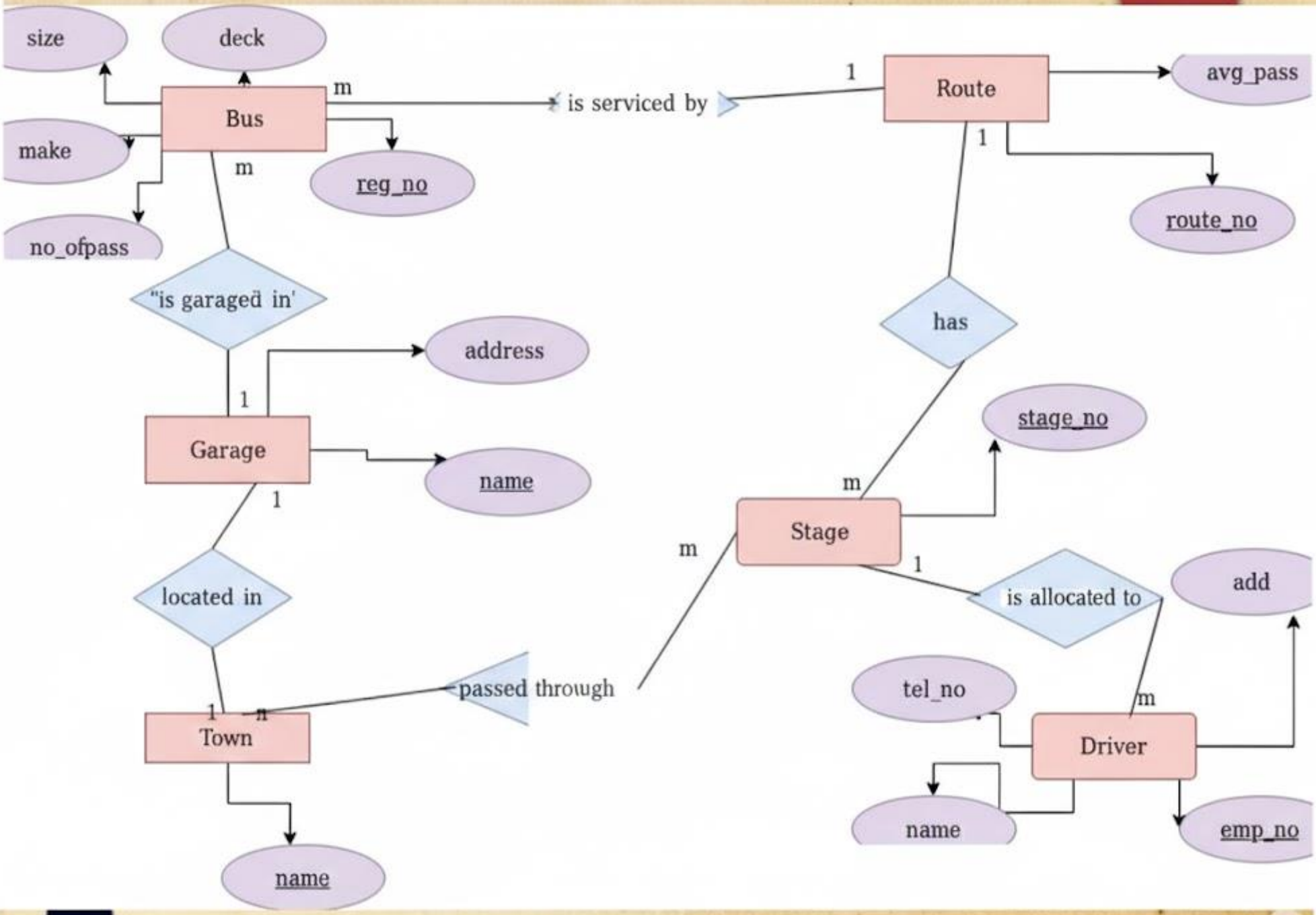
garage-town (1:1) is situated

- A garage keeps buses and each bus has one `home' garage

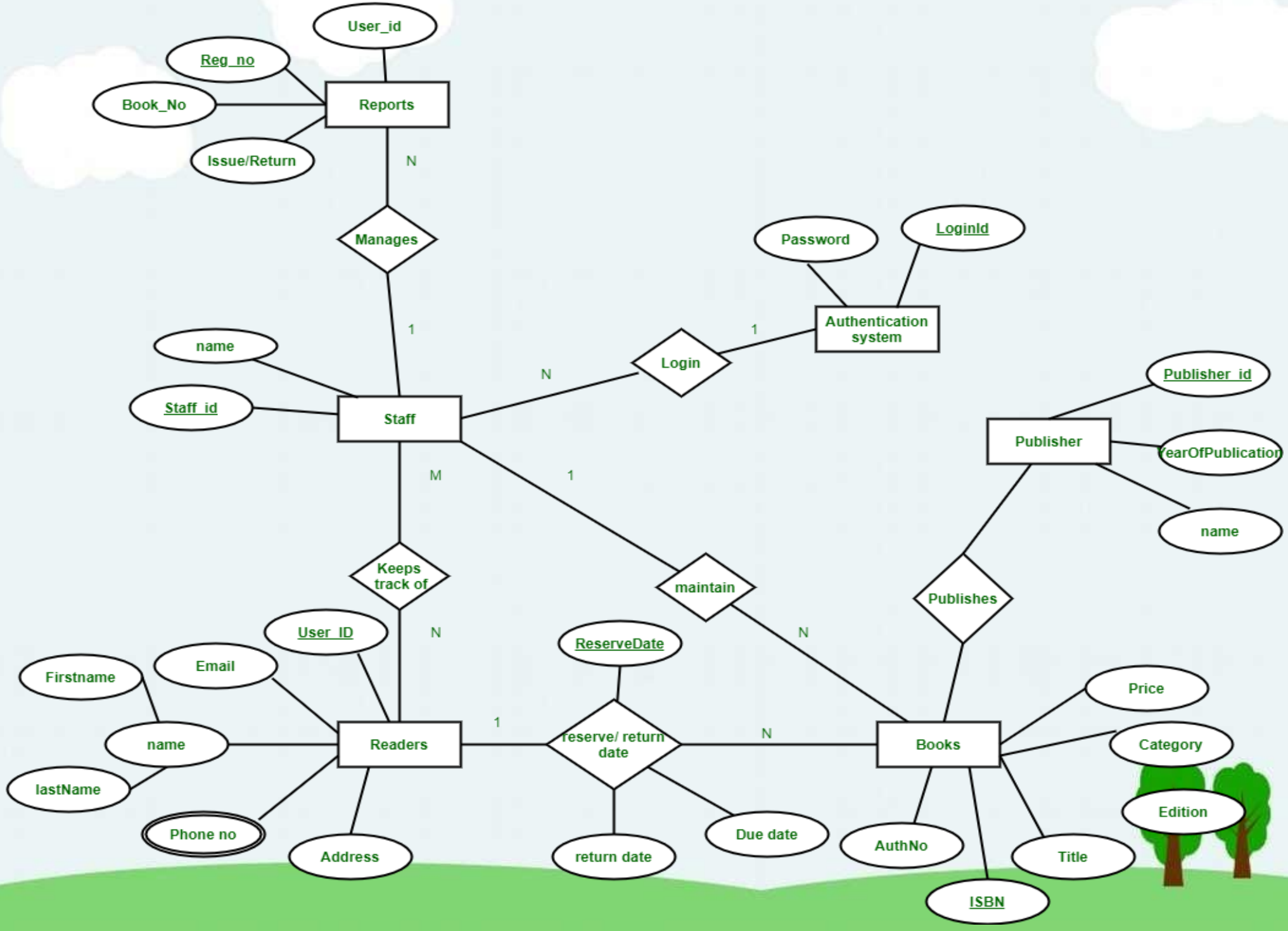
garage-bus (1:m) is garaged

Attributes

- Bus (reg-no,make,size,deck,no-pass)
- Route (route-no,avg-pass)
- Driver (emp-no,name,address,tel-no)
- Town (name)
- Stage (stage-no)
- Garage (name,address)


size
deck
Bus
m
is serviced by
1
Route
avg_pass
make
reg_no
1
route_no
m
no_ofpass
"is garaged in'
has
address
1
Garage
name
stage_no
m
1
Stage
m
1
is allocated to
add
located in
passed through
tel_no
1
n
Town
m
Driver
name
emp_no
name

ER Diagram for Library Management System



Entities and their Attributes -

•**Book Entity :** It has authno, isbn number, title, edition, category, price. ISBN is the Primary Key for Book Entity.

•**Reader Entity :** It has UserId, Email, address, phone no, name. Name is composite attribute of firstname and lastname. Phone no is multi valued attribute. UserId is the Primary Key for Readers entity.

•**Publisher Entity :** It has PublisherId, Year of publication, name. PublisherID is the Primary Key.

•**Authentication System Entity :** It has LoginId and password with LoginID as Primary Key.

•**Reports Entity :** It has UserId, Reg_no, Book_no, Issue/Return date. Reg_no is the Primary Key of reports entity.

•**Staff Entity :** It has name and staff_id with staff_id as Primary Key.

•**Reserve/Return Relationship Set :** It has three attributes: Reserve date, Due date, Return date.

Relationships **between Entities -**

•A reader can reserve N books but one book can be reserved by only one reader. The relationship 1:N.

•A publisher can publish many books but a book is published by only one publisher. The relationship 1:N.

•Staff keeps track of readers. The relationship is M:N.

•Staff maintains multiple reports. The relationship 1:N.

•Staff maintains multiple Books. The relationship 1:N.

•Authentication system provides login to multiple staffs. The relation is 1:N.

DFD

- A picture is worth a thousand words. A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both.
- It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

It is usually beginning with a context diagram as level 0 of the DFD diagram, a simple representation of the whole system.

- To elaborate further from that, we drill down to a level 1 diagram with lower-level functions decomposed from the major functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required.
- Progression to levels 3, 4 and so on is possible but anything beyond level 3 is not very common. Please bear in mind that the level of detail for decomposing a particular function depending on the complexity that function.

DFD Diagram Notations

External Entity

- An external entity can represent a human, system or subsystem. It is where certain data comes from or goes to. It is external to the system we study, in terms of the business process. For this reason, people used to draw external entities on the edge of a diagram.

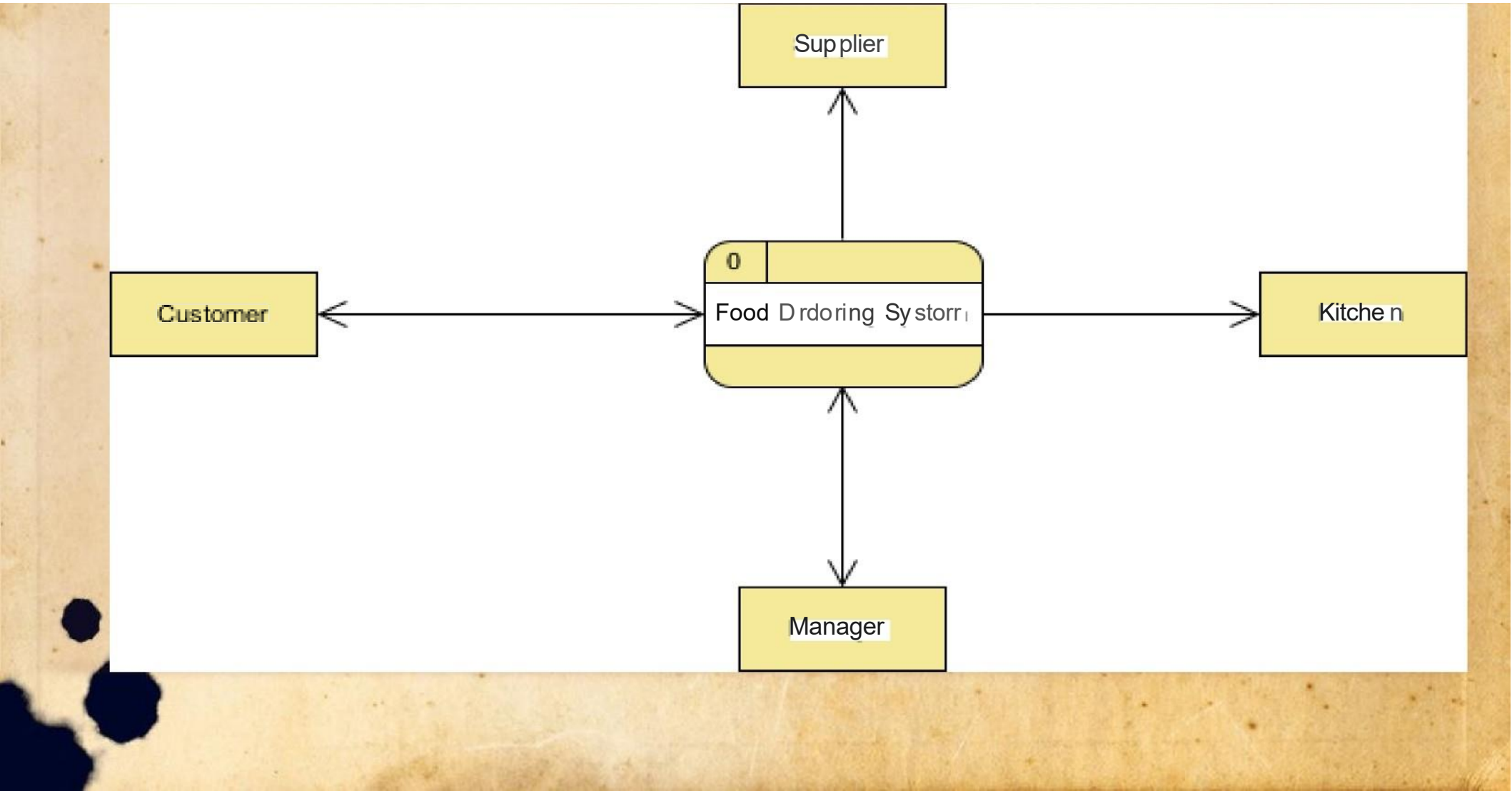
Process

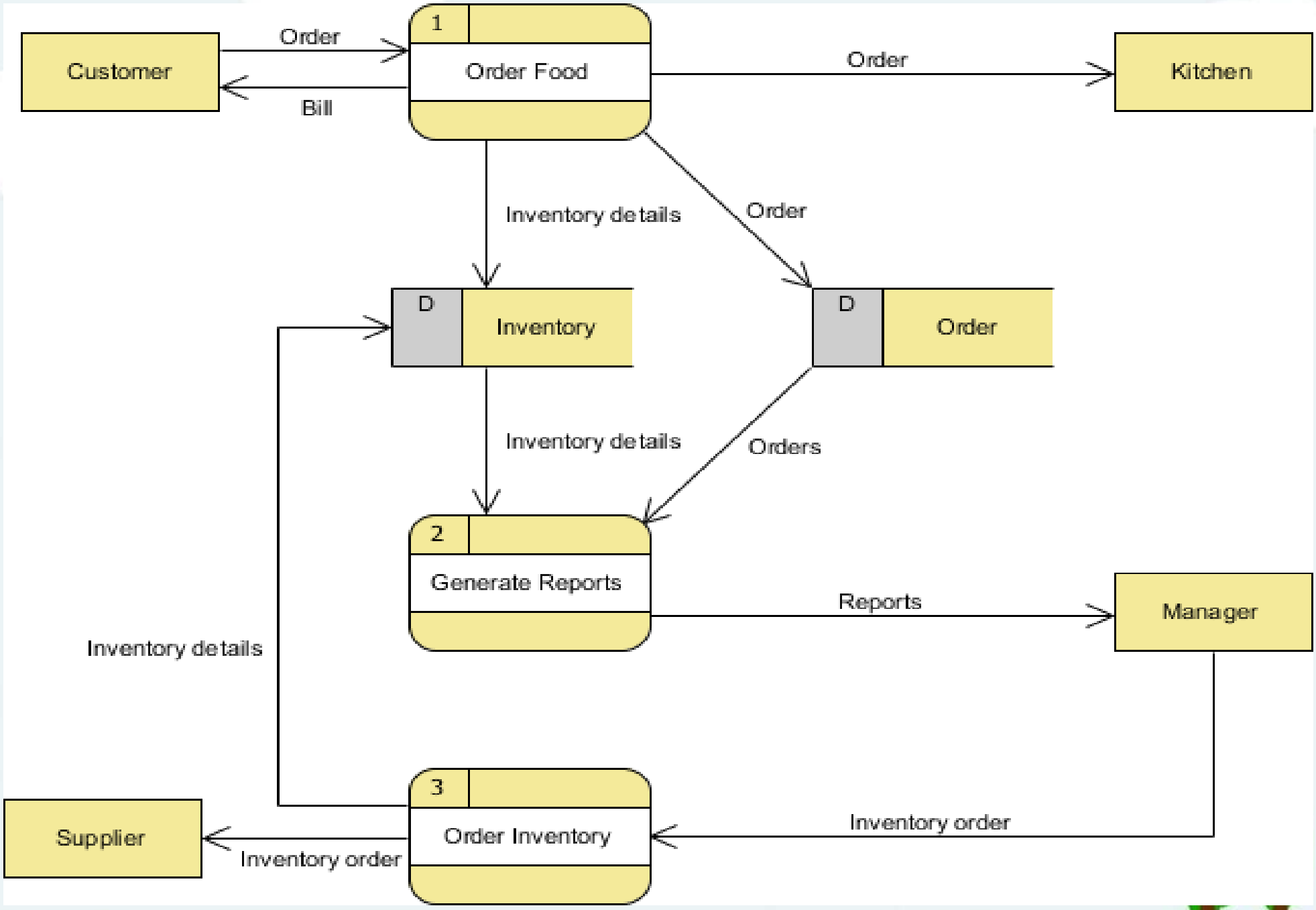
- A process is a business activity or function where the manipulation and transformation of data take place. A process can be decomposed to a finer level of details, for representing how data is being processed within the process.
- Data Store
- A data store represents the storage of persistent data required and/or produced by the process. Here are some examples of data stores: membership forms, database tables, etc.

The Food Ordering System

Context 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 in regards to how it interacts with external entities. Some of the benefits of a Context Diagram are:
- Shows the overview of the boundaries of a system
- No technical knowledge is required to understand with the simple notation
- Simple to draw, amend and elaborate as its limited notation


Supplier
0
Customer
Food Drdoring Systorr
Kitchen
Manager


Customer
Order
Bill
1
Order Food
Order
Kitchen
Inventory details
Order
D
Inventory
D
Order
Inventory details
Orders
2
Generate Reports
Reports
Manager
Inventory details
3
Supplier
Inventory order
Order Inventory
Inventory order

The Food Order System Data Flow Diagram example contains three processes, four external entities, and two data stores.

- Based on the diagram, we know that a Customer can place an Order. The Order Food process receives the Order, forwards it to the Kitchen, store it in the Order data store, and store the updated Inventory details in the Inventory data store. The process also delivers a Bill to the Customer.

- The Manager can receive Reports through the Generate Reports process, which takes Inventory details and Orders as input from the Inventory and Order data store respectively.
- The Manager can also initiate the Order Inventory process by providing Inventory order. The process forwards the Inventory order to the Supplier and stores the updated Inventory details in the Inventory data store.

- Process labels should be verb phrases; data stores are represented by nouns
- A data store must be associated with at least a process
- An external entity must be associated with at least a process
- Don't let it get too complex; normally 5 - 7 average people can manage processes
- DFD is non-deterministic - The numbering does not necessarily indicate sequence, it's useful in identifying the processes when discussing with users
- Data stores should not be connected to an external entity, otherwise, it would mean that you're giving an external entity direct access to your data files
- Data flows should not exist between 2 external entities without going through a process
- A process that has inputs but without outputs is considered to be a black-hole process