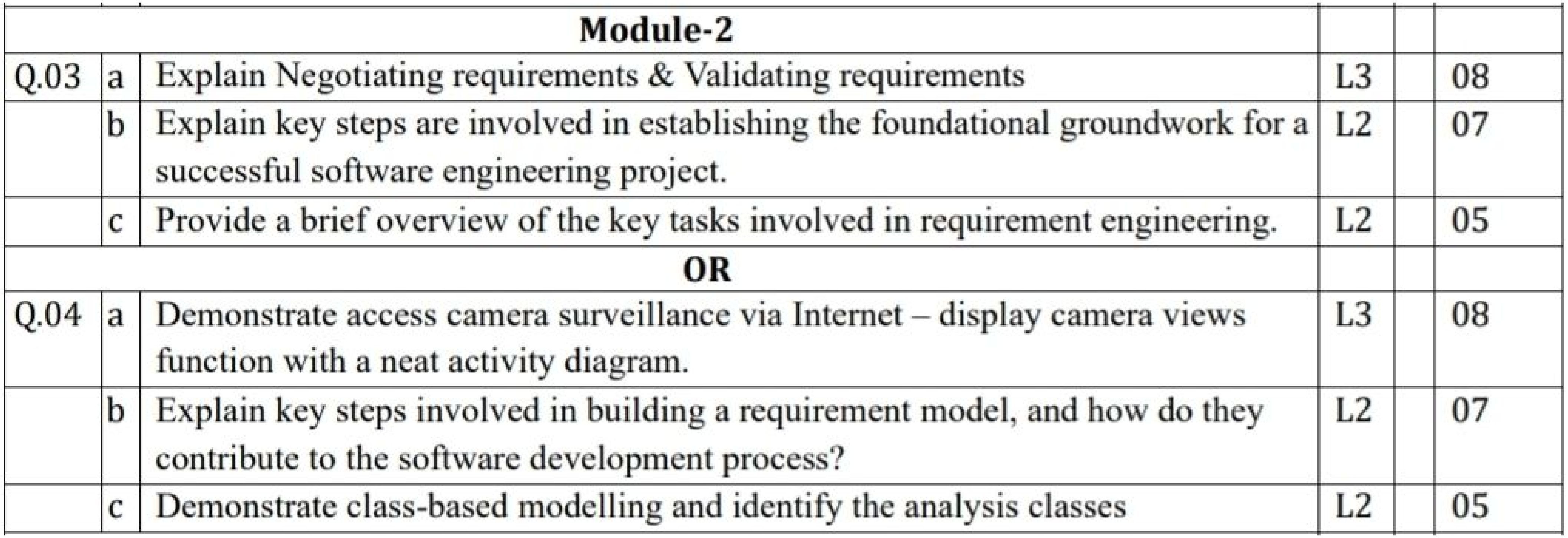
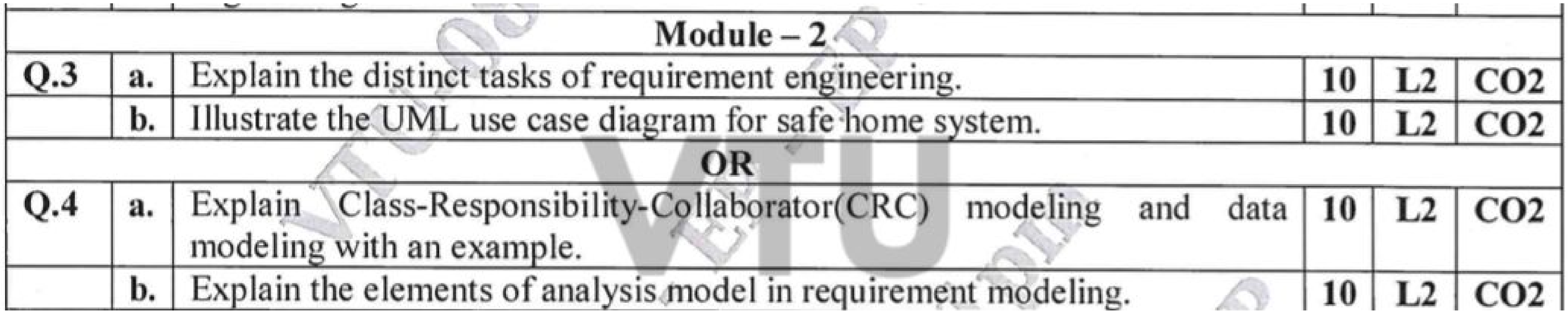
MQP



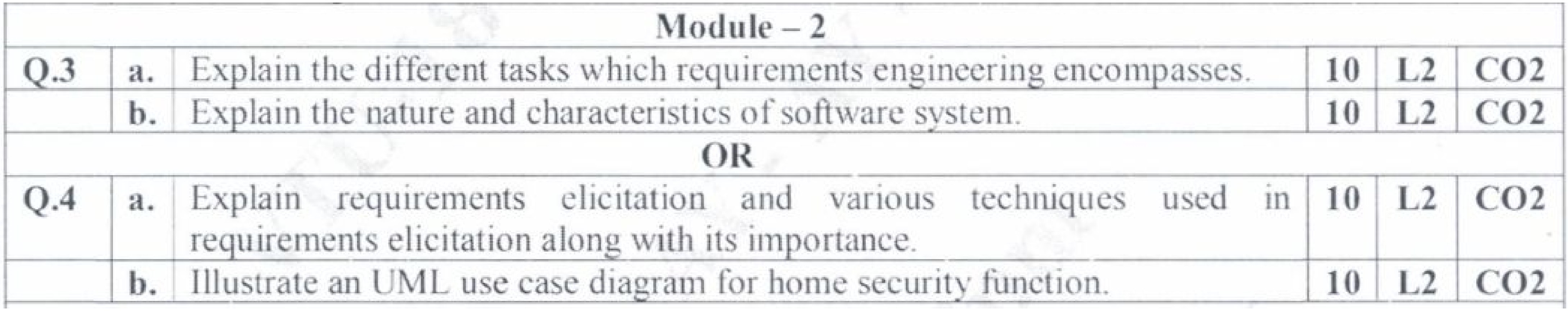
		Module-2			
Q.03	a	Explain Negotiating requirements & Validating requirements	L3		08
	b	Explain key steps are involved in establishing the foundational groundwork for a successful software engineering project.	L2		07
	c	Provide a brief overview of the key tasks involved in requirement engineering.	L2		05
		OR			
Q.04	a	Demonstrate access camera surveillance via Internet – display camera views function with a neat activity diagram.	L3		08
	b	Explain key steps involved in building a requirement model, and how do they contribute to the software development process?	L2		07
	c	Demonstrate class-based modelling and identify the analysis classes	L2		05

DEC/JAN 2024-25



		Module – 2			
Q.3	**a.**	Explain the distinct tasks of requirement engineering.	**10**	**L2**	**CO2**
	b.	Illustrate the UML use case diagram for safe home system.	**10**	**L2**	**CO2**
		OR			
Q.4	**a.**	Explain Class-Responsibility-Collaborator(CRC) modeling and data modeling with an example.	**10**	**L2**	**CO2**
	b.	Explain the elements of analysis model in requirement modeling.	**10**	**L2**	**CO2**

JUN/JULY 2025



		Module – 2			
Q.3	**a.**	Explain the different tasks which requirements engineering encompasses.	**10**	**L2**	**CO2**
	b.	Explain the nature and characteristics of software system.	**10**	**L2**	**CO2**
		OR			
Q.4	**a.**	Explain requirements elicitation and various techniques used in requirements elicitation along with its importance.	**10**	**L2**	**CO2**
	b.	Illustrate an UML use case diagram for home security function.	**10**	**L2**	**CO2**

3a. Explain Negotiating requirements & Validating requirements

NEGOTIATING REQUIREMENTS

The intent of negotiation is to develop a project plan that meets stakeholder needs while at the same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team. The best negotiations strive for a "**win-win**" result. That is, stakeholders win by getting the system or product that satisfies the majority of their needs and you win by working to realistic and achievable budgets and deadlines.

Boehm defines a set of negotiation activities at the beginning of each software process iteration. Rather than a single customer communication activity, the following activities are defined:

1. Identification of the system or subsystem's key stakeholders.

2. Determination of the stakeholders' "win conditions."

3. Negotiation of the stakeholders' win conditions to reconcile them into a set of win-win conditions for all concerned.

4.Successful completion of these initial steps achieves a win-win result, which becomes the key criterion for proceeding to subsequent software engineering activities.

VALIDATING REQUIREMENTS

As each element of the requirements model is created, it is examined for inconsistency, omissions, and ambiguity. The requirements represented by the model are prioritized by the stakeholders and grouped within requirements packages that will be implemented as software increments.

A review of the requirements model addresses the following questions:

- Is each requirement consistent with the overall objectives for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally,

a specific individual) noted for each requirement?

These and other questions should be asked and answered to ensure that the requirements model is an accurate reflection of stakeholder needs and that it provides a solid foundation for design.

3b. Explain key steps are involved in establishing the foundational groundwork for a successful software engineering project.

ESTABLISHING THE GROUNDWORK

Identifying Stakeholders:

A ***stakeholder*** is anyone who has a direct interest in or benefits from the system that is to be developed. At inception, you should create a list of people who will contribute input as requirements are elicited..

Recognizing Multiple Viewpoints:

Because many different stakeholders exist, the requirements of the system will be explored from many different points of view. The information from multiple viewpoints is collected, emerging requirements may be inconsistent or may conflict with one another.

Working toward Collaboration:

The job of a requirements engineer is to identify areas of commonality and areas of conflict or inconsistency. It is, of course, the latter category that presents a challenge. Collaboration does not necessarily mean that requirements are defined by committee. 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 .

Asking the First Questions

Questions asked at the inception of the project should be "**context free**" . The first set of context- free questions focuses on the customer and other stakeholders, the overall project goals and benefits. For example, you might ask:

- Who is behind the request for this work?
- Who will use the solution?
- What will be the economic benefit of a successful solution?
- Is there another source for the solution that you need?

These questions help to identify all stakeholders who will have interest in the software to be built. In addition, the questions identify the measurable benefit of a

successful implementation and possible alternatives to custom software development.

The next set of questions enables you to gain a better understanding of the problem and allows the customer to voice his or her perceptions about a solution:

- How would you characterize "good" output that would be generated by a successful solution?
- What problem(s) will this solution address?
- Can you show me (or describe) the business environment in which the solution will be used?
- Will special performance issues or constraints affect the way the solution is

approached? The final set of questions focuses on the effectiveness of the communication activity itself. Gause and Weinberg call these "**meta-questions**" and propose the following list:

- Are you the right person to answer these questions? Are your answers
- "official"?
- Are my questions relevant to the problem that you have?
- Am I asking too many questions?
- Can anyone else provide additional information?
- Should I be asking you anything else?

These questions will help to "**break the ice**" and initiate the communication that is essential to successful elicitation. But a question-and-answer meeting format is not an approach that has been overwhelmingly successful.

3 c. Provide a brief overview of the key tasks involved in requirement engineering

: It encompasses **seven** distinct tasks: **inception, elicitation, elaboration, negotiation, specification, validation, and management**.

Inception: It establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team.

Elicitation: In this stage, proper information is extracted to prepare to document the

requirements. 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.

- **Problems of scope.** The boundary of the system is ill-defined or the customers/users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.
- **Problems of understanding.** The customers/users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain, have trouble communicating needs to the system engineer, omit information that is believed to be "obvious," specify requirements that conflict with the needs of other customers/users, or specify requirements that are ambiguous or un testable.
- **Problems of volatility.** The requirements change over time.

Elaboration: The information obtained from the customer during inception and elicitation is expanded and refined during elaboration. This 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 that describe **how** the end user (and other actors) will interact with the system.

Negotiation: To negotiate the requirements of a system to be developed, it is necessary to identify conflicts and to resolve those conflicts. You have to reconcile these conflicts through a process of negotiation. Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority. Using an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, requirements are eliminated, combined, and/or modified so that each party achieves some measure of satisfaction.

Specification: 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.

Validation: The work products produced as a consequence of requirements engineering are assessed for quality during a validation step. Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies,

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

2.3.1 Developing an Activity Diagram

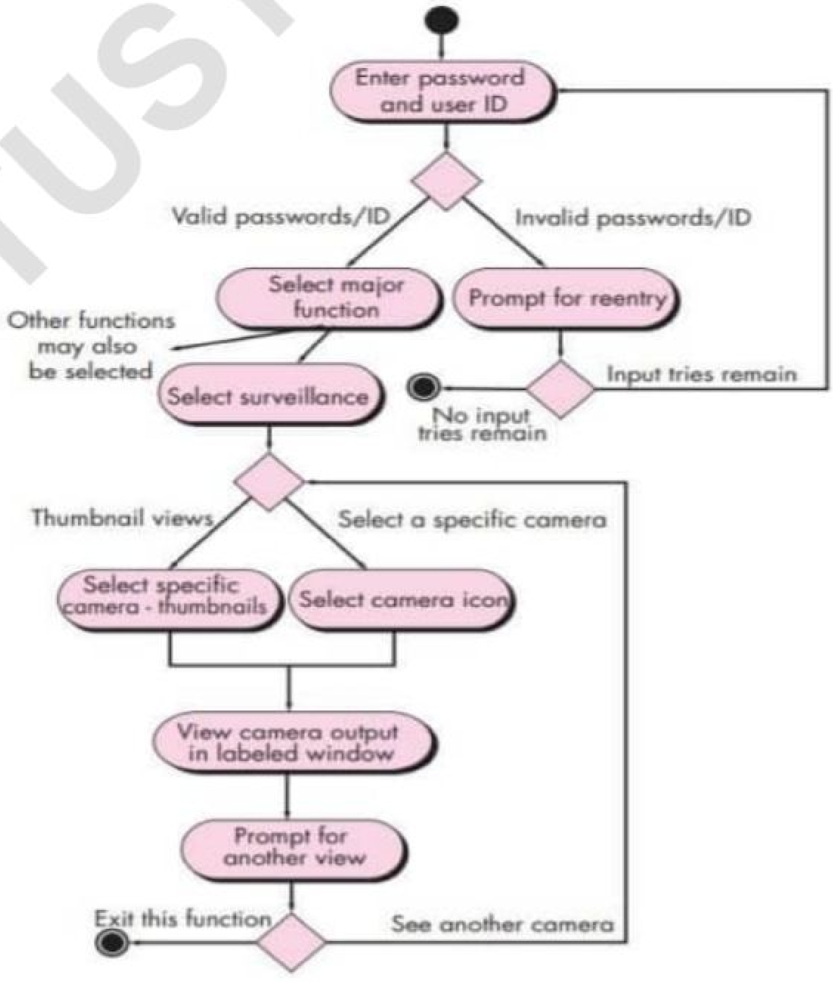
The UML activity diagram supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario. Similar to the flowchart,
An activity diagram uses:

- **Rounded rectangles** to imply a specific system function
- **Arrows** to represent flow through the system
- **Decision diamonds** to depict a branching decision.
- **Solid horizontal lines** to indicate that parallel activities are occurring.

A UML activity diagram represents the actions and decisions that occur as some function is performed.

FIGURE 6.5

Activity diagram for Access camera surveillance via the Internet—display camera views function.



4b. Explain key steps involved in building a requirement model, and how do they contribute to the software development process?

Here are the key steps involved in building a requirements model and their contributions to the software development process:

1. Requirements Elicitation

Contribution: Ensures that all necessary information is gathered from stakeholders to define the system's needs, which drives the direction of the entire project.

2. Requirements Analysis

Contribution: Organizes and refines gathered information, resolving any conflicts or ambiguities, ensuring the requirements are realistic and feasible.

3. Requirements Specification

Contribution: Documents the requirements in a clear, structured way, providing a formalized blueprint for development and ensuring all stakeholders have a shared understanding.

4. Requirements Validation

Contribution: Verifies that the documented requirements are correct, complete, and aligned with the user's and business's needs, reducing the risk of miscommunication.

5.Modeling the Requirements

Contribution: Visualizes complex requirements using models (e.g., use cases, activity diagrams), making them easier to understand and communicate, ensuring clarity for the development team.

6. Requirements Prioritization

Contribution: Ranks the requirements based on importance and urgency, ensuring that the most critical features are developed first and helping manage project scope.

7. Requirements Traceability

Contribution: Tracks each requirement through design, implementation, and testing, ensuring all requirements are addressed and making it easier to manage changes.

8. Managing Changes to Requirements

Contribution: Provides a structured process for handling changes to requirements, ensuring that changes are controlled and do not disrupt the development timeline or scope.

2.5 CLASS-BASED MODELING

Class-based modeling represents the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships between the objects, and the collaborations that occur between the classes that are defined.

The elements of a class-based model include classes and objects, attributes, operations, class responsibility collaborator (CRC) models, collaboration diagrams, and packages.

2.5.1 Identifying Analysis Classes

We can begin to identify classes by examining the usage scenarios developed as part of the requirements model and performing a "**grammatical parse**" on the use cases developed for the system to be built.

Analysis classes manifest themselves in one of the following ways:

- **External entities** (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.

- **Things** (e.g., reports, displays, letters, signals) that are part of the information domain for the problem.
- ***Occurrences or events*** (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation.
- **Roles** (e.g., manager, engineer, salesperson) played by people who interact with the system.
- **Organizational units** (e.g., division, group, team) that are relevant to an application.
- **Places** (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system.
- **Structures** (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects.

2.5.4 Class-Responsibility-Collaborator (CRC) Modeling

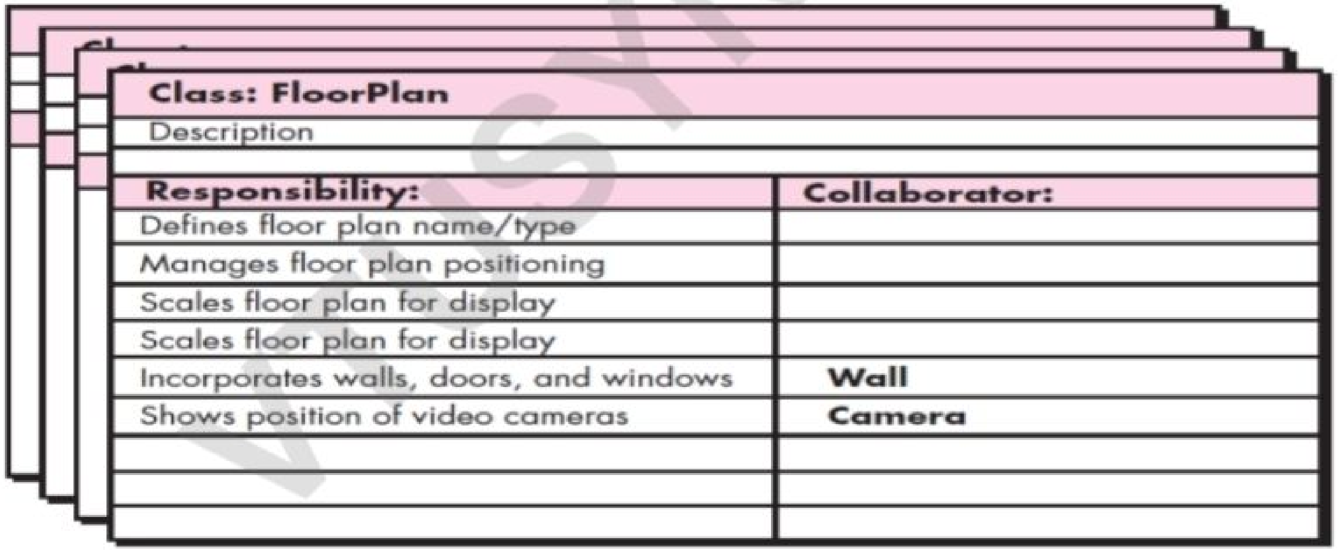
Class-responsibility-collaborator (CRC) modeling provides a simple means for identifying and organizing the classes that are relevant to system or product requirements.

Ambler describes CRC modeling in the following way:

A CRC model is really a collection of standard **index cards** that represent classes.

The cards are divided into **three sections.** Along the top of the card, you write the name of the **class**. In the body of the card, you list the class responsibilities on the left and the collaborators on the right. The CRC model may make use of actual or virtual index cards. The intent is to develop an organized representation of classes. **Responsibilities** are the attributes and operations that are relevant for the class. i.e., a responsibility is "anything the class knows or does" **Collaborators** are those classes that are required to provide a class with the information needed to complete a responsibility. In general, a collaboration implies either a request for information or a request for some action.

A simple CRC index card is illustrated in following figure.



Class: FloorPlan	
Description	
Responsibility:	**Collaborator:**
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Scales floor plan for display	
Incorporates walls, doors, and windows	**Wall**
Shows position of video cameras	**Camera**

Fig : A CRC model index card

2.4 DATA MODELING CONCEPTS

Data modeling is 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 new software or for re-engineering a legacy application.

The most widely used data Model by the Software engineers is **Entity-Relationship Diagram (ERD),** it addresses the issues and represents all data objects that are entered, stored, transformed, and produced within an application.

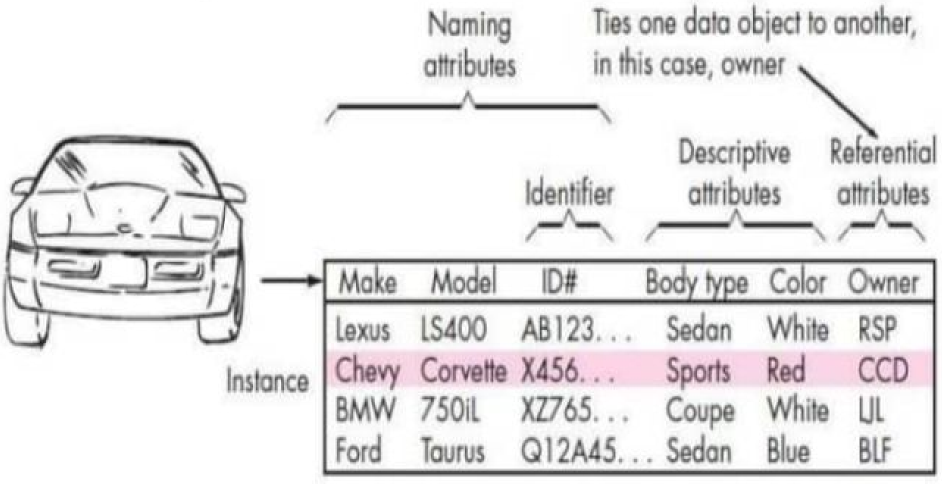
2.4.1 Data Objects

A data object is a representation of composite information that must be understood by software. Therefore width (a single value) would not be a valid data object, but dimensions (incorporating height, width, and depth) could be defined as an object.

A data object can be an **external entity** (e.g., anything that produces or consumes information), a **thing** (e.g., a report or a display), **an occurrence** (e.g., a telephone call) or event (e.g., an alarm), **a role** (e.g., salesperson)**, an organizational unit** (e.g., accounting department), **a place** (e.g., a warehouse), or a **structure** (e.g., a file).

For example**, a person or a car** can be viewed as a data object in the sense that either can be defined in terms of a set of attributes. The description of the data object incorporates the data object and all of its attributes.

A data object encapsulates data only—there is no reference within a data object to operations that act on the data. Therefore, the data object can be represented as a table as shown in following table. The headings in the table reflect attributes of the object.



Make	Model	ID#	Body type	Color	Owner
Lexus	LS400	AB123. . .	Sedan	White	RSP
Chevy	Corvette	X456. . .	Sports	Red	CCD
BMW	750iL	XZ765. . .	Coupe	White	LJL
Ford	Taurus	Q12A45. . .	Sedan	Blue	BLF

Fig : Tabular representation of data objects

2.4 DATA MODELING CONCEPTS

Data modeling is 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 new software or for re-engineering a legacy application.

The most widely used data Model by the Software engineers is **Entity-Relationship Diagram (ERD),** it addresses the issues and represents all data objects that are entered, stored, transformed, and produced within an application.

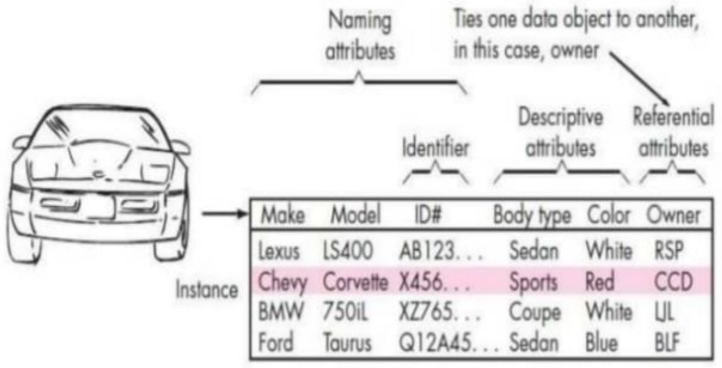
2.4.1 Data Objects

A data object is a representation of composite information that must be understood by software. Therefore width (a single value) would not be a valid data object, but dimensions (incorporating height, width, and depth) could be defined as an object.

A data object can be an **external entity** (e.g., anything that produces or consumes information), a **thing** (e.g., a report or a display), **an occurrence** (e.g., a telephone call) or event (e.g., an alarm), **a role** (e.g., salesperson), **an organizational unit** (e.g., accounting department), **a place** (e.g., a warehouse), or a **structure** (e.g., a file).

For example**, a person or a car** can be viewed as a data object in the sense that either can be defined in terms of a set of attributes. The description of the data object incorporates the data object and all of its attributes.

A data object encapsulates data only—there is no reference within a data object to operations that act on the data. Therefore, the data object can be represented as a table as shown in following table. The headings in the table reflect attributes of the object.



Make	Model	ID#	Body type	Color	Owner
Lexus	LS400	AB123. . .	Sedan	White	RSP
Chevy	Corvette	X456. . .	Sports	Red	CCD
BMW	750iL	XZ765. . .	Coupe	White	LJL
Ford	Taurus	Q12A45. . .	Sedan	Blue	BLF

Fig : Tabular representation of data objects

2.4.2 Data Attributes

Data attributes define the properties of a data object and take on one of three different characteristics. They can be used to (1) name an instance of the data object, (2) describe the instance, or (3) make reference to another instance in another table.

2.4.3 Relationships

Data objects are connected to one another in different ways. Consider the two data objects, **person and car.** These objects can be represented using the following simple notation and relationships are 1) A person owns a car, 2) A person is insured to drive a car.

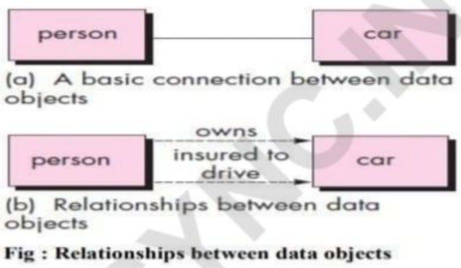

person
car
(a) A basic connection between data objects
owns
insured to drive
person
car
(b) Relationships between data objects

Fig : Relationships between data objects

- ***Scenario-based elements*** depict how the user interacts with the system and the specific sequence of activities that occur as the software is used.

- ***Class-based elements model*** the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships between the objects, and the collaborations that occur between the classes that are defined.

- ***Behavioral elements*** depict how external events change the state of the system or the classes that reside within it.

- **Flow-oriented elements** represent the system as an information transform, depicting how data objects are transformed as they flow through various system functions.

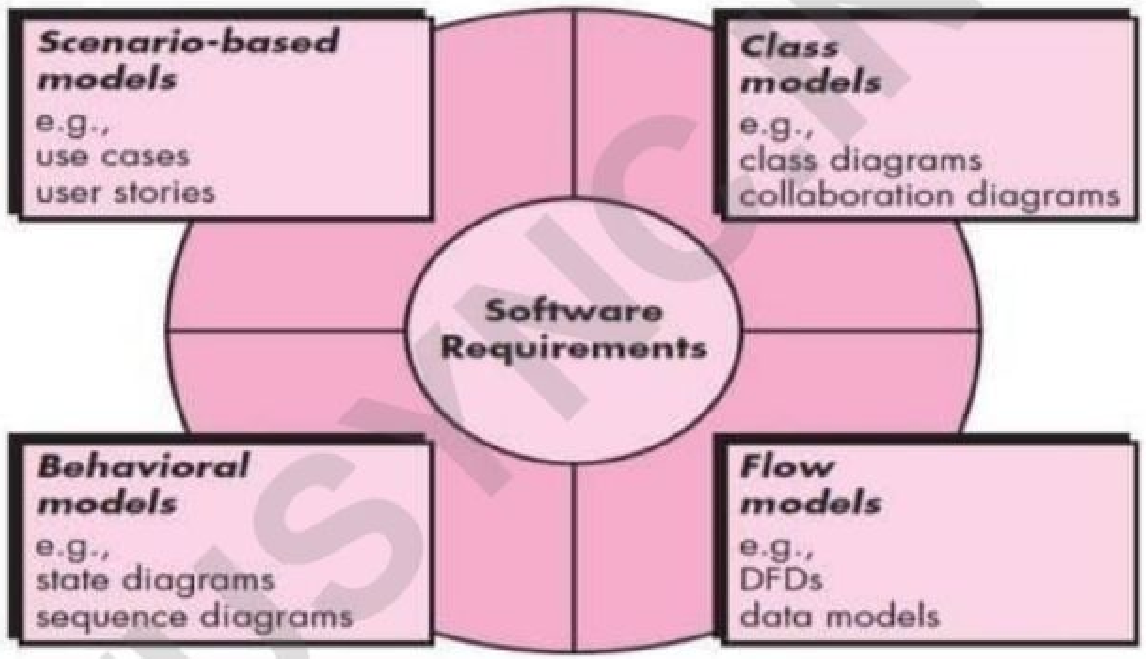
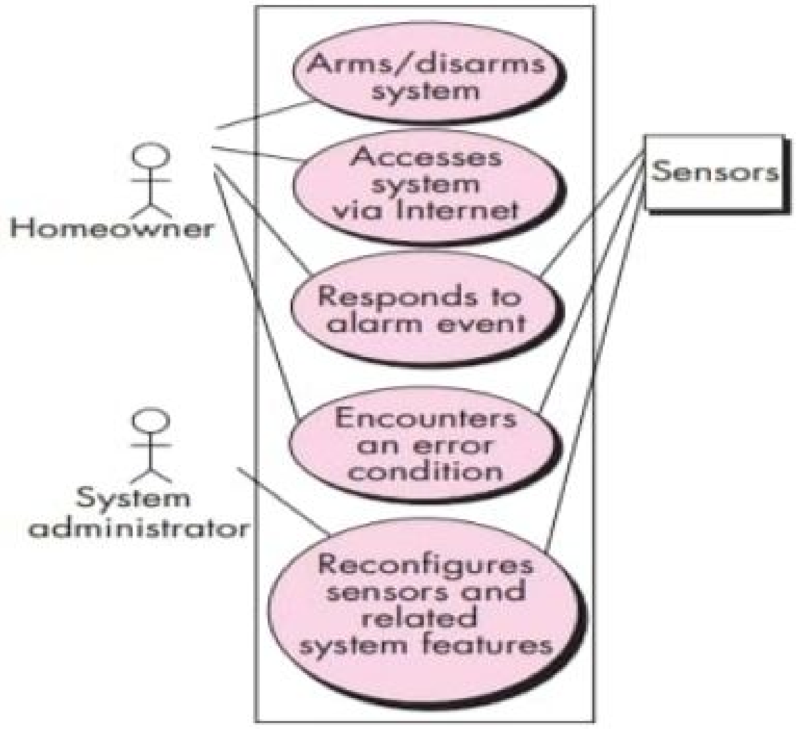

Scenario-based models
e.g., use cases user stories
Class models
e.g., class diagrams collaboration diagrams
Software Requirements
Behavioral models
e.g., state diagrams sequence diagrams
Flow models
e.g., DFDs data models

Fig : Elements of the analysis model

FIGURE 5.2

UML use case diagram for *SafeHome* home security function



Arms/disarms system
Accesses system via Internet
Responds to alarm event
Encounters an error condition
Reconfigures sensors and related system features
Homeowner
System administrator
Sensors