UNIT II - SOFTWARE REQUIREMENTS AND ANALYSIS

Mr. Rahul Pawar
Assistant Professor, School of CS&IT,
MCA Department,
Knowledge Campus

Requirements Engineering

- The broad spectrum of tasks and techniques that lead to an understanding of requirements is called ***requirements engineering.***
- From a software process perspective, requirements engineering is a major software engineering action that begins during the communication activity and continues into the modeling activity.
- It must be adapted to the needs of the process, the project, the product, and the people doing the work.
- Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specification, and managing the requirements as they are transformed into an operational system.
- 7 distinct tasks: Inception, Elicitation, Elaboration, Negotiation, Specification, Validation, and Management.

- **Inception**—ask a set of questions that establish …
  - 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 customer and the developer.
- **Elicitation**
  - Elicit requirements from all stakeholders (customer, users, etc.)
  - Why requirement elicitation is difficult?
    - **Problem of Scope**: The boundary of the system is ill-defined.
    - **Problems of Understanding**: The customer/users are not sure of what is needed. They may have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain.
    - **Problems of Volatility**: The requirements change over time.

- **Elaboration**
  - Create an analysis model that identifies data, function and behavioral requirements.
  - It is driven by the creation and refinement of user scenarios that describe how the end-user will act with the system.
- **Negotiation**
  - It is relatively common for different customers or users to propose conflicting requirements
  - Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority.
  - Risks in each requirement are identified and analyzed.
  - Agree on a deliverable system that is realistic for developers and customers.

- **Specification** (means different things to different people)
  - Can be any one (or more) of the following:
    - A written document
    - A set of models
    - A formal mathematical
    - A collection of user scenarios (use-cases)
    - A prototype
- **Validation**
  - Requirements validation examines the specification
    - to ensure that all software requirements have been stated unambiguously; inconsistencies, omissions, and errors have been detected and corrected;

- **Validation (contd..)**

  - mechanism is the technical review
  - review team that validates requirements includes

    looking for errors in content or interpretation

    areas where clarification may be required,

    missing information

    inconsistencies

    conflicting requirements, or unrealistic requirements

- **Requirements management**
  - Set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds.

INFO

Software Requirements Specification Template

A *software requirements specification* (SRS) is a document 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. It is important to note that a formal SRS is not always written. In fact, there are many instances in which effort expended on an SRS might be better spent in other software engineering activities. However, when software is to be developed by a third party, when a lack of specification would create severe business issues, or when a system is extremely complex or business critical, an SRS may be justified.

Karl Wiegers [Wie03] of Process Impact Inc. has developed a worthwhile template (available at **www.processimpact.com/process_assets/srs_template.doc**) that can serve as a guideline for those who must create a complete SRS. A topic outline follows:

Table of Contents
Revision History

1. Introduction
1.1 Purpose
1.2 Document Conventions
1.3 Intended Audience and Reading Suggestions
1.4 Project Scope
1.5 References

2. Overall Description
2.1 Product Perspective
2.2 Product Features
2.3 User Classes and Characteristics
2.4 Operating Environment
2.5 Design and Implementation Constraints
2.6 User Documentation
2.7 Assumptions and Dependencies

3. System Features
3.1 System Feature 1
3.2 System Feature 2 (and so on)

4. External Interface Requirements
4.1 User Interfaces
4.2 Hardware Interfaces
4.3 Software Interfaces
4.4 Communications Interfaces

5. Other Nonfunctional Requirements
5.1 Performance Requirements
5.2 Safety Requirements
5.3 Security Requirements
5.4 Software Quality Attributes

6. Other Requirements

Appendix A: Glossary
Appendix B: Analysis Models
Appendix C: Issues List

A detailed description of each SRS topic can be obtained by downloading the SRS template at the URL noted earlier in this sidebar.

Establishing the Groundwork

- **Steps for understanding the software requirements**

Identifying Stakeholders

- Stakeholder - "anyone who benefits in a direct or indirect way from the system
- Operations managers, product managers, marketing people, internal and external customers, end-users, consultants, product engineers, software engineers, support and maintenance engineers, etc.
- At inception, the requirements engineer should create a list of people who will contribute input as requirements are elicited.

Recognizing Multiple Viewpoints

- Different stakeholders exist - the requirements of the system will be explored from many different points of view
- As information from multiple viewpoints is collected, emerging requirements may be inconsistent or may conflict with one another.
- The requirements engineer is to categorize all stakeholder information including inconsistencies and conflicting requirements in a way that will allow decision makers to choose an internally consistent set of requirements for the system.

Working toward Collaboration

- 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" (business manager, or a senior technologist) may make the final decision about which requirements make the final cut.

Asking the First Questions

- The first set of context-free questions focuses on the customer and other stakeholders, overall goals, and benefits.
- The first questions:

  -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.

- 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 [Gau89] call these "meta-questions" and propose the following (abbreviated) 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?

Eliciting Requirements

- Requirements elicitation (also called *requirements gathering)* combines elements of problem solving, elaboration, negotiation, and specification.

1.Collaborative Requirements Gathering

- Gathering the requirements by conducting the meetings between developer and customer.
- Fix the rules for preparation and participation.
- The main motive is to identify the problem, give the solutions for the elements, negotiate the different approaches and specify the primary set of solution requirements in an environment which is valuable for achieving goal.

2.Quality Function Deployment (QFD)

- In this technique, translate the customer need into the technical requirement for the software.
- QFD "concentrates on maximizing customer satisfaction from the software engineering process"
- QFD identifies 3 types of requirements:
  - **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, specific system functions, and defined levels of performance.

- **Expected requirements:** These requirements are implicit.
  - These are the basic requirement that not be clearly told by the customer, but also the customer expect that requirement. (Ex: easiness of UI)
- **Exciting requirements:** These features are beyond the expectation of the customer.
  - The developer adds some additional features or unexpected feature into the software to make the customer more satisfied.
    For example, the mobile phone with standard features, but the developer adds few additional functionalities like voice searching, multi-touch screen etc. then the customer more exited about that feature.

3. Usage scenarios

- Till the software team does not understand how the features and function are used by the end users it is difficult to move technical activities.
- To achieve above problem the software team produces a set of structure that identify the usage for the software.
- This structure is called as 'Use Cases'.

4. Elicitation work product

- The work product created as a result of requirement elicitation that is depending on the size of the system or product to be built.

- A statement of need and feasibility.
- A bounded statement of scope for the system or product.
- A list of customers, users, and other stakeholders who participated in requirements elicitation.
- A description of the system's technical environment.
- A list of requirements (preferably organized by function) and the domain constraints that apply to each.
- A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- Any prototypes developed to better define requirements

SafeHome

Conducting a Requirements Gathering Meeting

The scene: A meeting room. The first requirements gathering meeting is in progress.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator (pointing at whiteboard): So that's the current list of objects and services for the home security function.

Marketing person: That about covers it from our point of view.

Vinod: Didn't someone mention that they wanted all *SafeHome* functionality to be accessible via the Internet? That would include the home security function, no?

Marketing person: Yes, that's right . . . we'll have to add that functionality and the appropriate objects.

Facilitator: Does that also add some constraints?

Jamie: It does, both technical and legal.

Production rep: Meaning?

Jamie: We better make sure an outsider can't hack into the system, disarm it, and rob the place or worse. Heavy liability on our part.

Doug: Very true.

Marketing: But we still need that . . . just be sure to stop an outsider from getting in.

Ed: That's easier said than done and . . .

Facilitator (interrupting): I don't want to debate this issue now. Let's note it as an action item and proceed.

(Doug, serving as the recorder for the meeting, makes an appropriate note.)

Facilitator: I have a feeling there's still more to consider here.

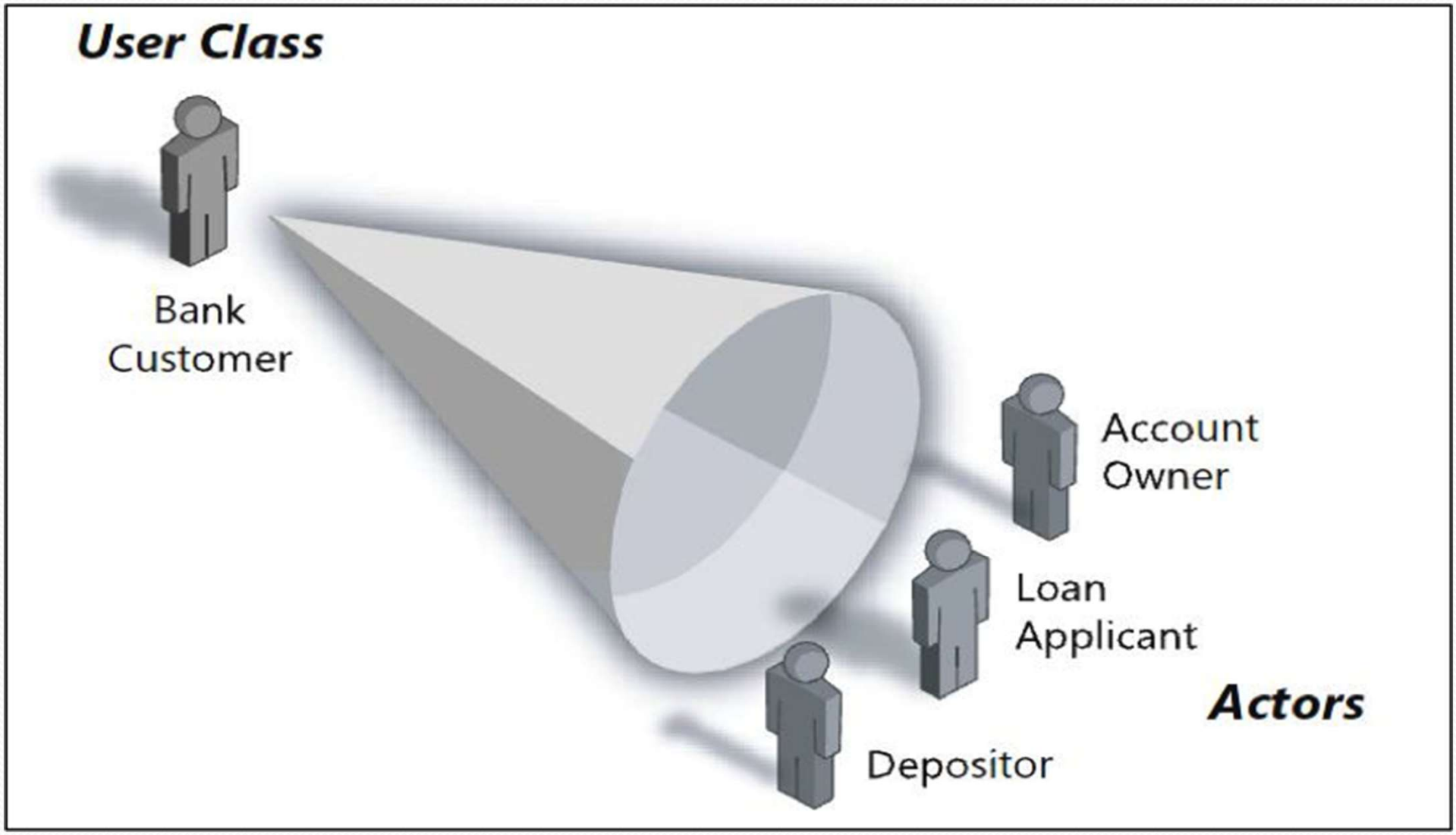
(The group spends the next 20 minutes refining and expanding the details of the home security function.)

Developing Use cases

- Use case tells a story about how an end user (playing one of a number of possible roles) interacts with the system under a specific set of circumstances. The story may be narrative text, an outline of tasks or interactions, a template-based description, or a diagrammatic representation.
- Use case depicts the software or system from the end user's point of view.
- The first step in writing a use case is to define the set of "actors" that will be involved in the story.
- ***An actor is a role that people (users) or devices play as they interact with the software.***
- It is important to note that an actor and an end user are not necessarily 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 (often, but not always, people) that play just one role in the context of the use case.
- As an example, consider a machine operator (a user) who interacts with the control computer for a manufacturing cell that contains a number of robots and numerically controlled machines.
- software for the control computer requires four different modes (roles) for interaction: programming mode, test mode, monitoring mode, and troubleshooting mode.
- Therefore, four actors can be defined: programmer, tester, monitor, and troubleshooter. In some cases, the machine operator can play all of these roles. In others, different people may play the role of each actor.

User and the Actor



- Once actors have been identified, use cases can be developed. Jacobson [Jac92] suggests a number of questions that should be answered by a use case:
  - Who is the primary actor, the secondary actor(s)?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What exceptions might be considered as the story is described?
  - What variations in the actor's interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

- *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.*

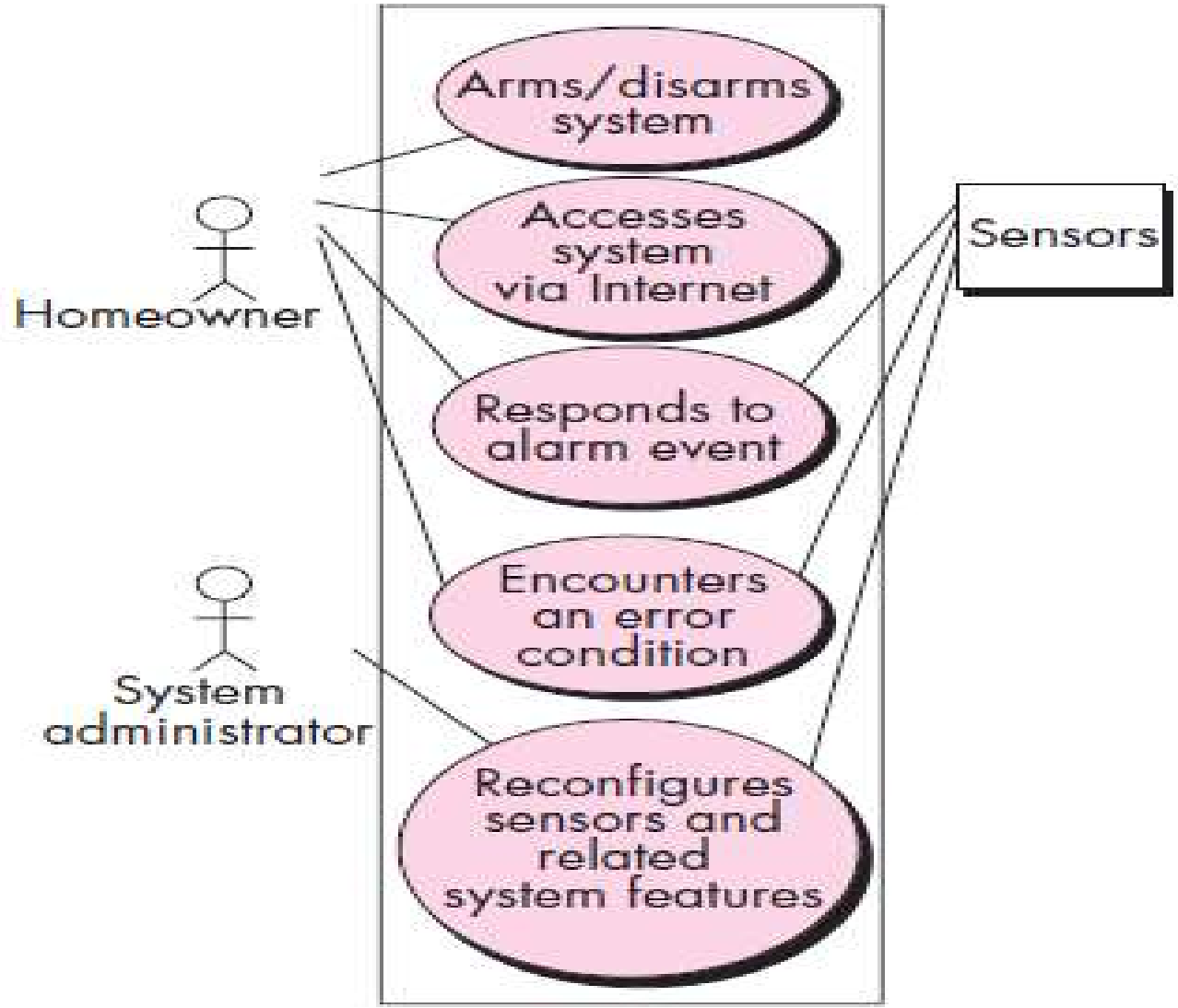
BUILDING THE REQUIREMENTS MODEL

Elements of the Requirements Model

Scenario-based elements **:**The system is described from the user's point of view using a scenario-based approach. For example, basic use cases and their corresponding use-case diagrams.

- *Use Case Modeling* – Define software functional requirements in terms of use cases and actors .
- *Activity Diagrams – represented using Use Cases*

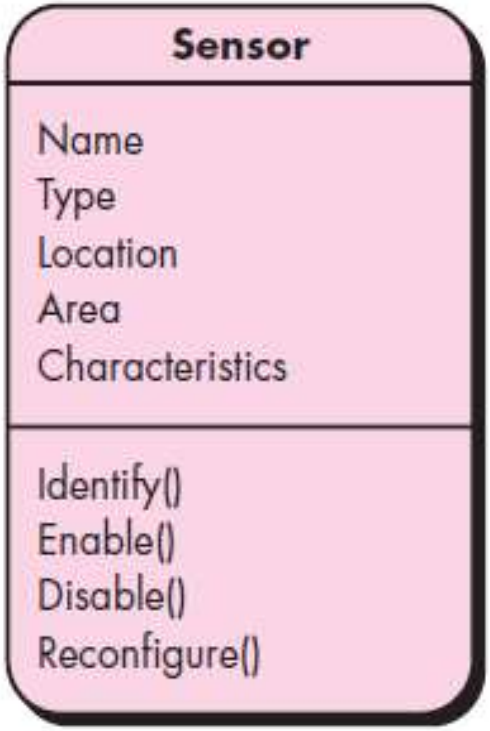
UML use case diagram for *Safe Home* security function



***Class-based elements*:** Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system.

- These objects are categorized into classes—a collection of things that have similar attributes and common behaviors.
- other analysis modeling elements depict the manner in which classes collaborate with one another and the relationships and interactions between classes
- For example, a UML class diagram can be used to depict a **Sensor class for the *SafeHome*** security function.

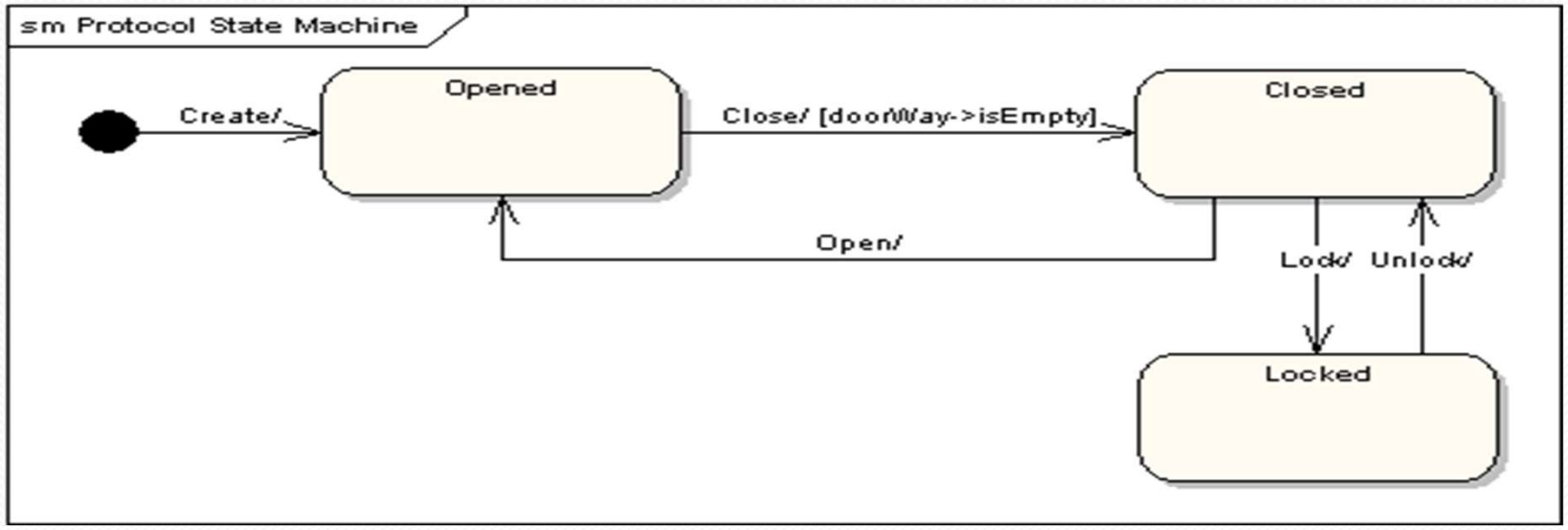
Class diagram for sensor



The diagram lists the attributes of sensors (e.g., name, type) and the operations (e.g., *identify, enable)*

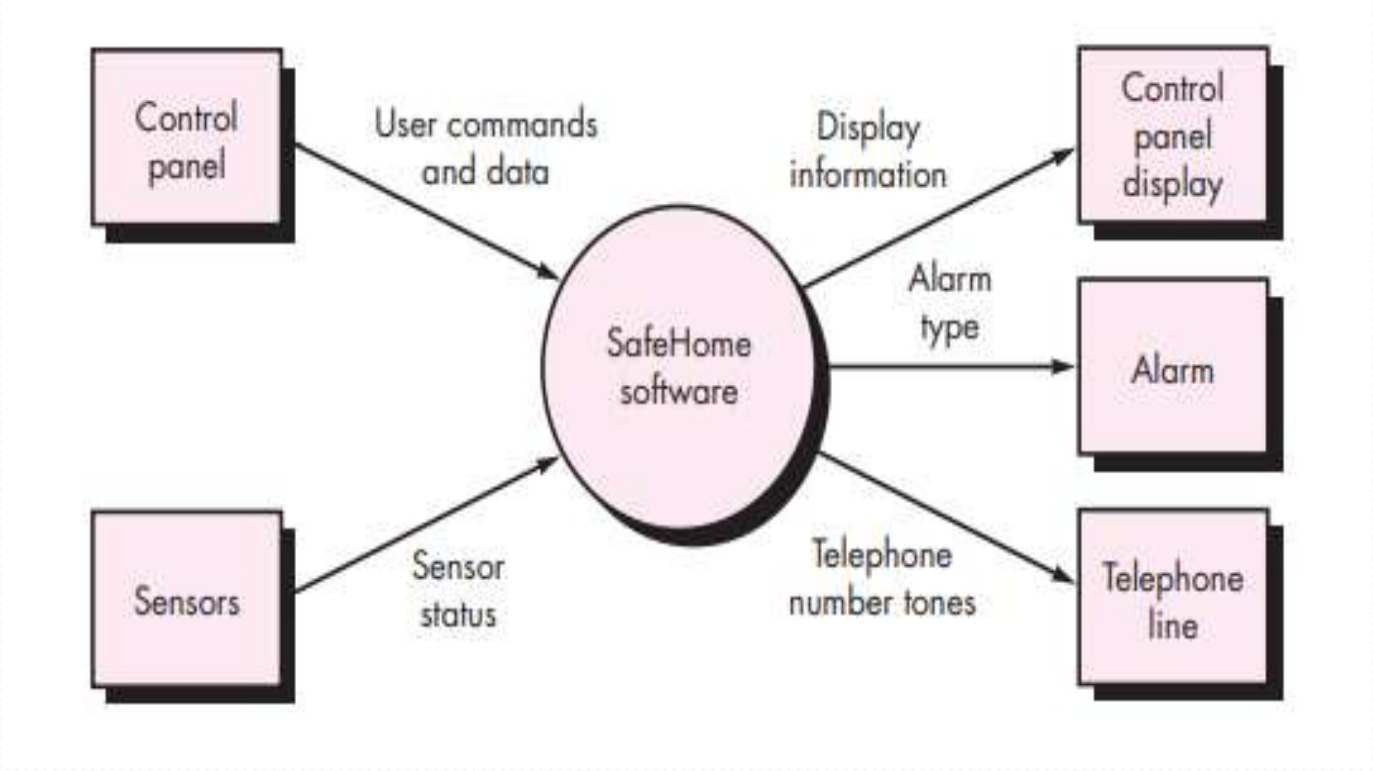
Behavioral elements: Requirements model must provide modeling elements that depict behavior.

- The ***state diagram*** *is one method for representing the behavior of a system by depicting* its states and the events that cause the system to change state.
- A *state is any* externally observable mode of behavior. In addition, the state diagram indicates actions (e.g., process activation) taken as a consequence of a particular event.

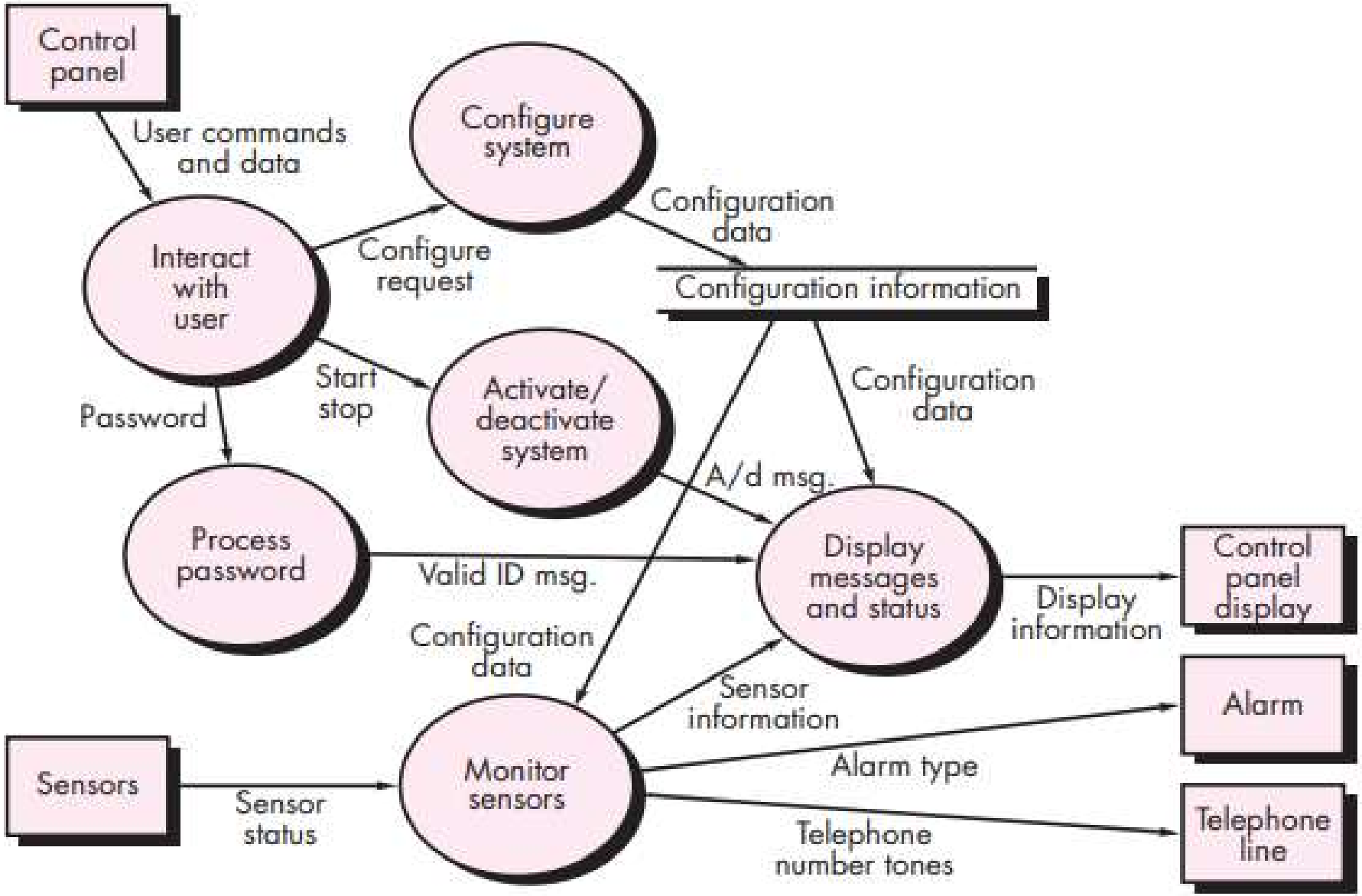


***Flow-oriented elements*:** Information is transformed as it flows through a computer-based system.

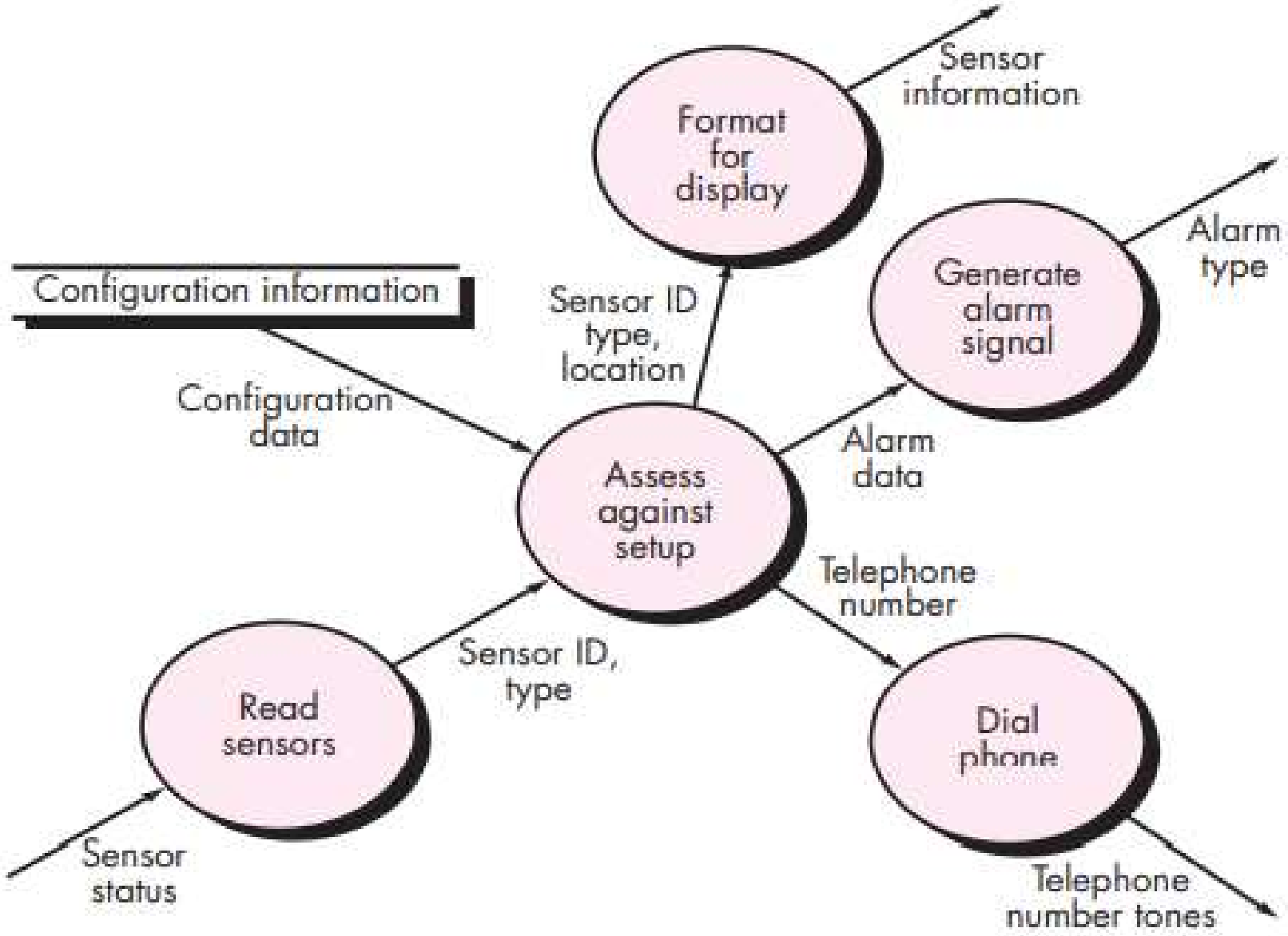
- The system accepts input in a variety of forms, applies functions to transform it, and produces output in a variety of forms.
- DFD (Data Flow Diagrams)



CONTEXT FLOW DIAGRAM – SAFEHOME SYSTEM



LEVEL 1 DFD – SECURITY FUNCTION IN SAFEHOME SYSTEM



LEVEL 2 DFD – refines the monitor sensors process

***Flow-oriented elements*:** Information is transformed as it flows through a computer-based system.

- The system accepts input in a variety of forms, applies functions to transform it, and produces output in a variety of forms.
- DFD (Data Flow Diagrams)

Analysis Patterns

- certain problems reoccur across all projects within a specific application domain
- Analysis patterns suggest solutions (e.g., a class, a function, a behavior) within the application domain that can be reused when modeling many applications.

Geyer-Schulz and Hahsler suggest two benefits that can be associated with the use of analysis patterns:

- First, analysis patterns speed up the development of abstract analysis models that capture the main requirements of the concrete problem by providing reusable analysis models
- Second, analysis patterns facilitate the transformation of the analysis model into a design model by suggesting design patterns and reliable solutions for common problems.

Negotiating Requirements

- In an ideal requirements engineering context, the inception, elicitation, and elaboration tasks determine customer requirements in sufficient detail to proceed to subsequent software engineering activities.
- Unfortunately, this rarely happens. In reality, you may have to enter into a negotiation with one or more stakeholders.
- In most cases, stakeholders are asked to balance functionality, performance, and other product or system characteristics against cost and time-to-market. The intent of this 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.

  - stakeholders win by getting the system or product that satisfies the majority of their needs

  - software team member 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.

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 (including the software team).

- 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?

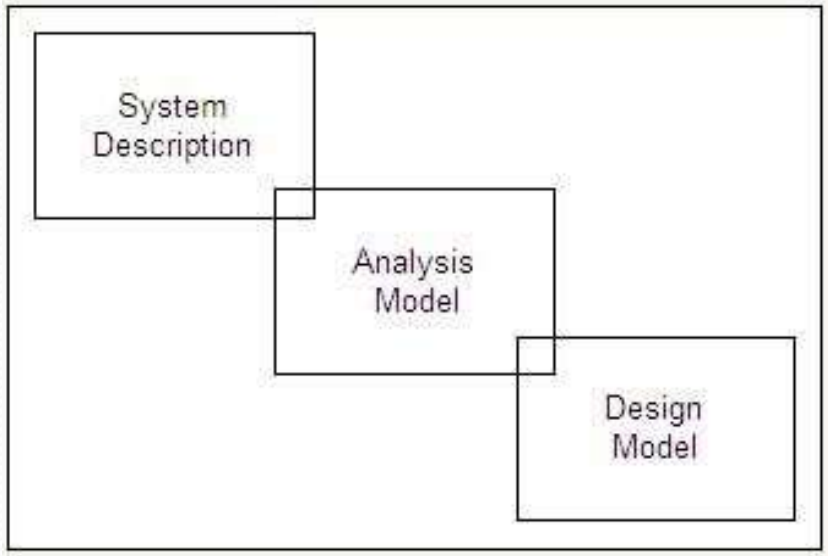
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
- Do any requirements conflict with other requirements?
- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function, and behavior of the system to be built?

- Has the requirements model been "partitioned" in a way that exposes progressively more detailed information about the system?
- Have requirements patterns been used to simplify the requirements model?
- Have all patterns been properly validated? Are all patterns consistent with customer requirements?

Requirements Analysis

- Requirements analysis allows you *to elaborate on* basic requirements established during the inception, elicitation, and negotiation tasks that are part of requirements engineering.
- **IEEE** defines requirements analysis as the process of studying user needs to arrive at a definition of a system, hardware or software requirements.
- Various other tasks performed using requirements analysis are listed below.
  - To detect and resolve conflicts that arise due to unclear and unspecified requirements
  - To determine operational characteristics of the software and how they interact with the environment
  - To understand the problem for which the software is to be developed
  - To develop an analysis model to analyze the requirements in the software.

- An analysis model created for the software facilitates the software development team to understand what is required in the software and then they develop it.



- The analysis model connects the system description and design model.
- System description provides information about the entire functionality of the system, which is achieved by implementing the software, hardware and data.

Design model provides information about the software's architecture, user interface, and component level structure.

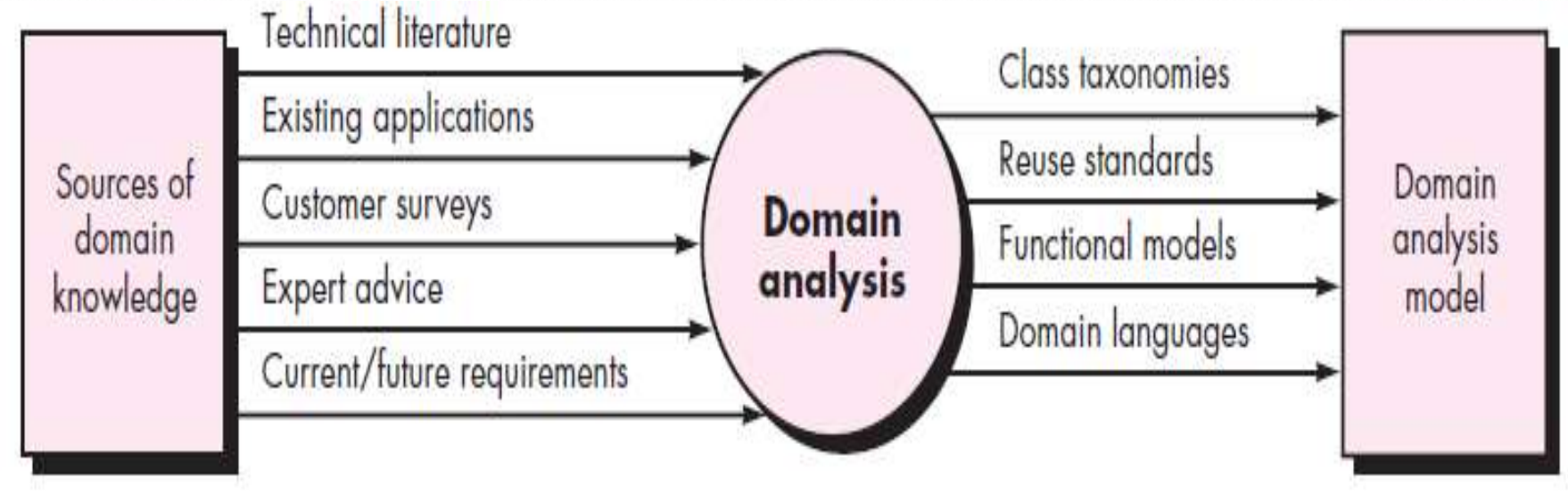
- The guidelines followed while creating an analysis model are listed below.
  - The model **should concentrate on requirements in the problem** domain that are to be accomplished. However, it **should not describe the procedure** to accomplish the requirements in the system.
  - Every element of the analysis model should help in **understanding the software requirements**. This model should also describe the information domain, function, and behavior of the system.
  - The analysis model **should be useful to all stakeholders** because every stakeholder uses this model in his own manner. For example, **business stakeholders use this model to validate requirements** whereas **software designers view this model as a basis for design.**
  - The analysis model **should be as simple as possible**. For this, additional diagrams that depict no new or unnecessary information should be avoided.

- Also, abbreviations and acronyms should be used instead of complete notations.

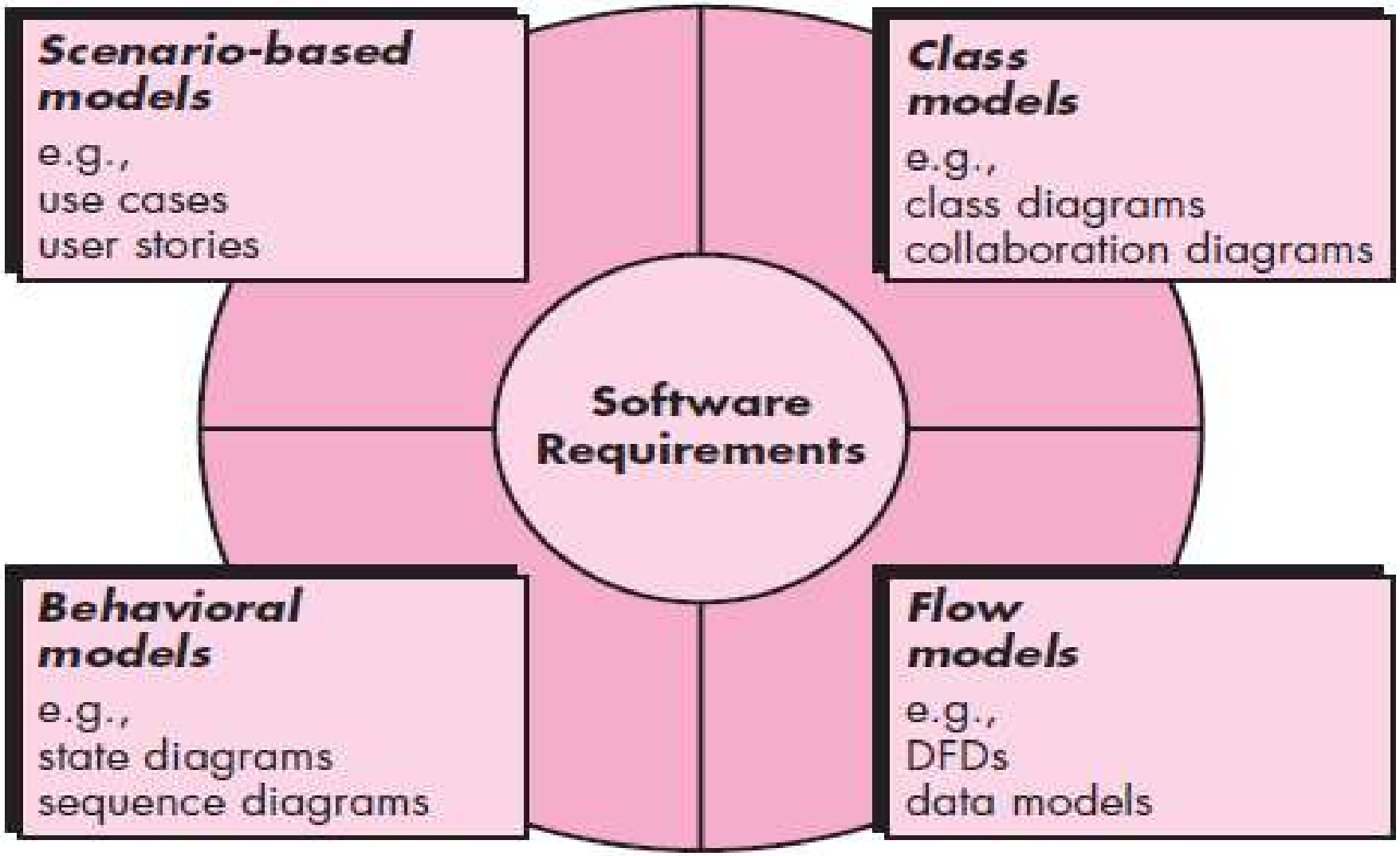
Domain Analysis:

- Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain. .
- Object-oriented domain analysis is the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks.

Input and output for domain analysis



Elements of the analysis model



Requirements Modeling Strategies

- One view of requirements modeling, called ***structured analysis***, *considers data and* the processes that transform the data as separate entities.
- Data objects are modeled in a way that defines their attributes and relationships.
- Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.
- A **second approach** to analysis modeling, called ***object-oriented analysis***, *focuses* on the definition of classes and the manner in which they collaborate with one another to effect customer requirements.


Thank You!