Module-2
Modeling Concepts

The software requirements are **description of features and functionalities of the target system**. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.
Requirement Engineering is the process of defining, documenting and maintaining the requirements. 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

1. **Requirements Elicitation:**

   It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project.
   The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Elicitation does not produce formal models of the requirements understood. Instead, it widens the domain knowledge of the analyst and thus helps in providing input to the next stage.

2. **Requirements specification:**

   This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process.The models used at this stage include ER diagrams, data flow diagrams(DFDs), function decomposition diagrams(FDDs), data dictionaries, etc.

3. **Requirements verification and validation:**
   Verification: It refers to the set of tasks that ensures that the software correctly implements a specific function.
   Validation: It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements.
   If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework.
   The main steps for this process include:

- The requirements should be consistent with all the other requirements i.e no two requirements should conflict with each other.
- The requirements should be complete in every sense.
- The requirements should be practically achievable.

4. **Requirements management:**

   Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible so as to incorporate changes in requirements specified by the end users at later stages too. Being able to modify the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

Requirements Engineering:

Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

Requirement Engineering Process

It is a five-step process, which includes -

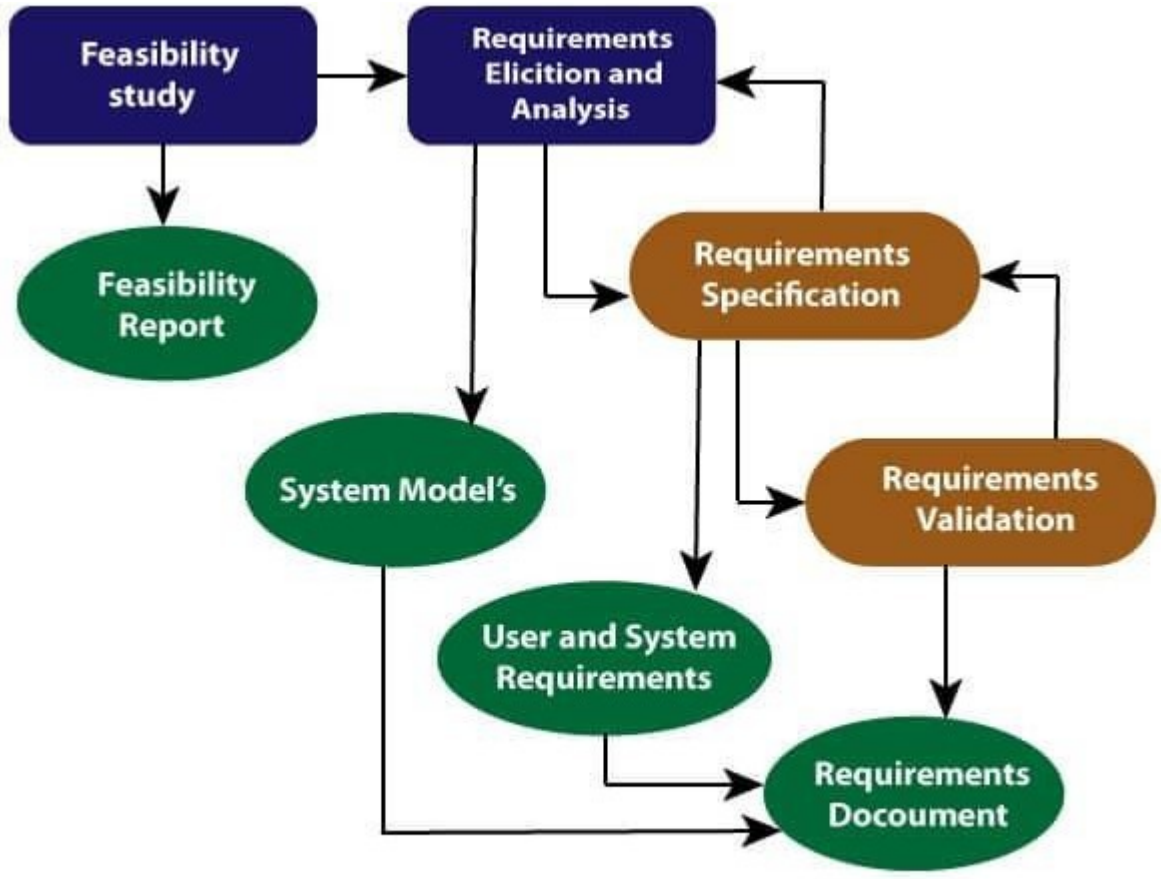
1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management

1. Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

Types of Feasibility:

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.



Requirement Engineering Process

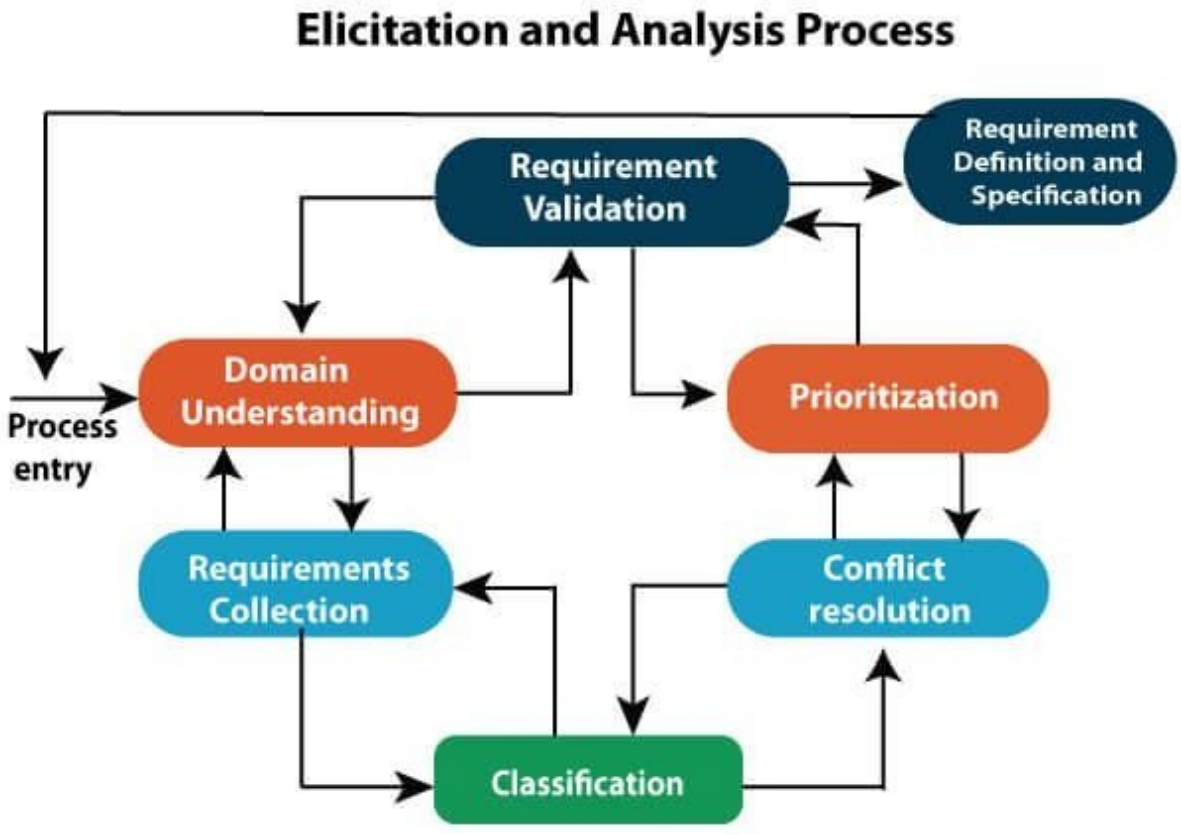
2. Requirement Elicitation and Analysis:

This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of Elicitation and Analysis

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements.



3. Software Requirement Specification:

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

- **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.
- **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
- **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "***E-R diagram***." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

4. Software Requirement Validation

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.
In the requirements validation process, we perform a different type of test to check the requirements mentioned in the Software Requirements Specifcation(SRS)which checks include:

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

There are several techniques which are used either individually or in conjunction with other techniques to check to check entire or part of the system:

1. **Test case generation:**
   Requirement mentioned in SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design than, this usually means that requirement will be difficult to implement and it should be reconsidered.
2. **Prototyping:**
   In this validation techniques the prototype of the system is presented before the end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is generally used to collect feedback about the requirement of the user.
3. **Requirements Reviews:**
   In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organizations and the client side, the reviewer systematically analyses the document to check error and ambiguity.
4. **Automated Consistency Analysis:**
   This approach is used for automatic detection of an error, such as non determinism, missing cases, a type error, and circular definitions, in requirements specifications. First, the requirement is structured in formal notation then CASE tool is used to check in-consistency of the system, The report of all inconsistencies is identified and corrective actions are taken.

5. Walk-through:

A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.

- Checking early whether the idea is feasible or not.
- Obtaining the opinions and suggestion of other people.
- Checking the approval of others and reaching an agreement.

5. Software Requirements Management:

The purpose of requirements management is to ensure product development goals are successfully met. It is a set of techniques for documenting, analyzing, prioritizing, and agreeing on requirements so that engineering teams always have current and approved requirements. Requirements management provides a way to avoid errors by keeping track of changes in requirements and fostering communication with stakeholders from the start of a project throughout the engineering lifecycle.

A typical requirements management process complements the systems engineering V model through these steps:

- Collect initial requirements from stakeholders
- Analyze requirements
- Define and record requirements

- Prioritize requirements
- Agree on and approve requirements
- Trace requirements to work items
- Query stakeholders after implementation on needed changes to requirements
- Utilize test management to verify and validate system requirements
- Assess impact of changes
- Revise requirements
- Document changes

Developing Use Cases:

The first step in writing a use case is to define the set of "actors" that will be involved in the story. *Actors are the different people (or devices) that use the system* or product within the context of the function and behavior that is to be described.

• 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?

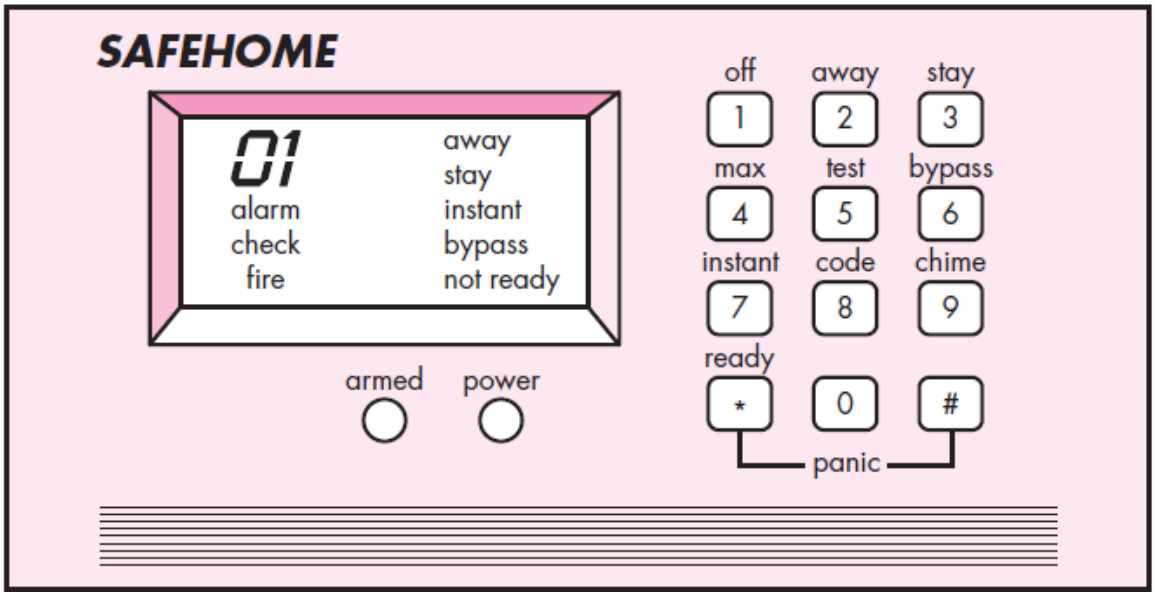
• What system information will the actor acquire, produce, or change?

• 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?

Safehome Control Panel:



we have to define four actors:

1. homeowner

2. setup manager

3. sensors

4. monitoring and response subsystem

Different ways using either the alarm control panel or a PC:

• Enters a password to allow all other interactions.

• Inquires about the status of a security zone.

• Inquires about the status of a sensor.

• Presses the panic button in an emergency.

• Activates/deactivates the security system.

Example:

Use case: InitiateMonitoring

Primary actor: Homeowner.

Goal in context: To set the system to monitor sensors when the homeowner leaves the house or remains inside.

Preconditions: System has been programmed for a password and to recognize various sensors.

Trigger: The homeowner decides to “set” the system, i.e., to turn on the alarm functions.

Scenario:

1. Homeowner: observes control panel

2. Homeowner: enters password

3. Homeowner: selects “stay” or “away”

4. Homeowner: observes read alarm light to indicate that *SafeHome has been armed*

Exceptions:

1. Control panel is not ready: homeowner checks all sensors to determine which are open; closes them.

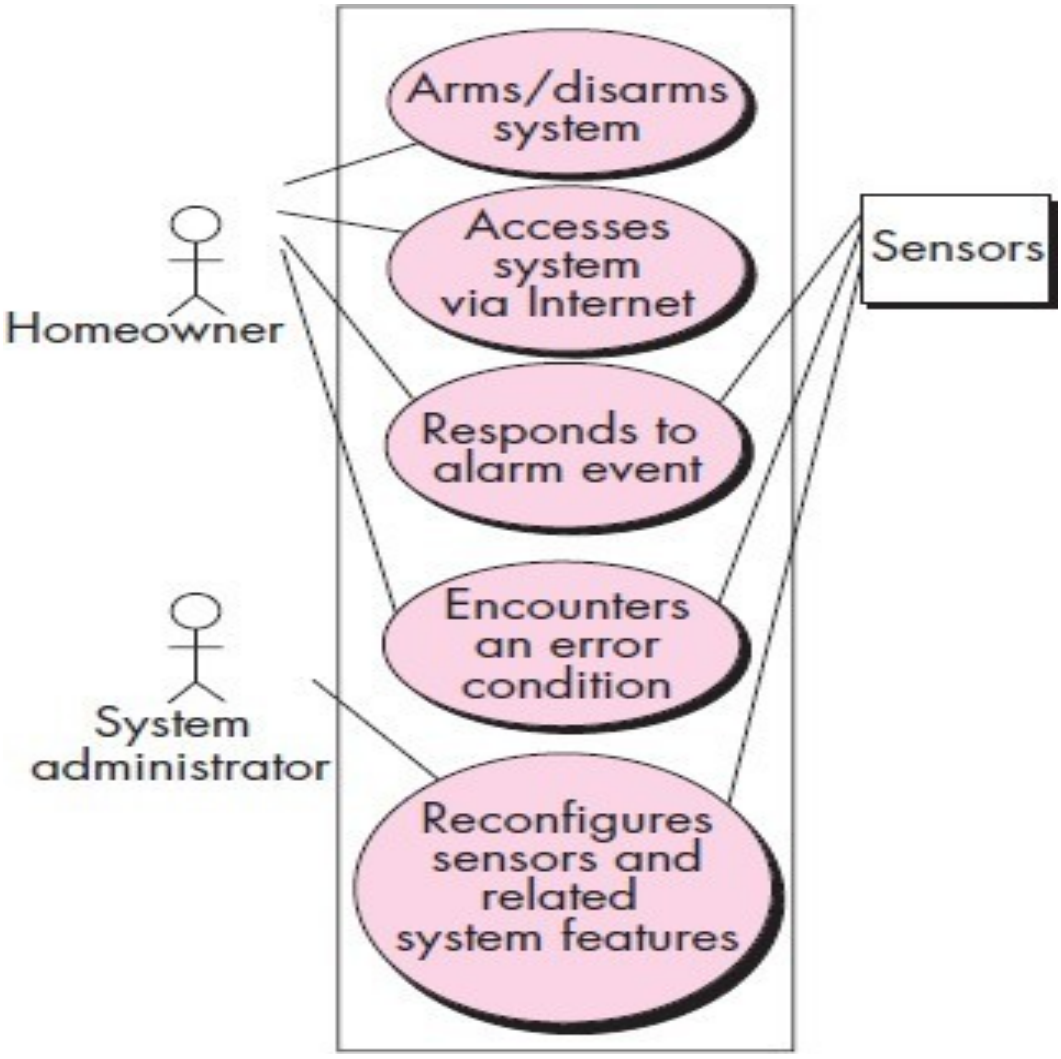
2. Password is incorrect (control panel beeps once): homeowner reenters correct password.

3. Password not recognized: monitoring and response subsystem must be contacted to reprogram password.

4. Stay is selected: control panel beeps twice and a stay light is lit; perimeter sensors are activated.

5. Away is selected: control panel beeps three times and an away light is lit; all sensors are activated.

Use case diagram for *SafeHome*



Building the requirements model:

The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system.

Elements of the Requirements Model:

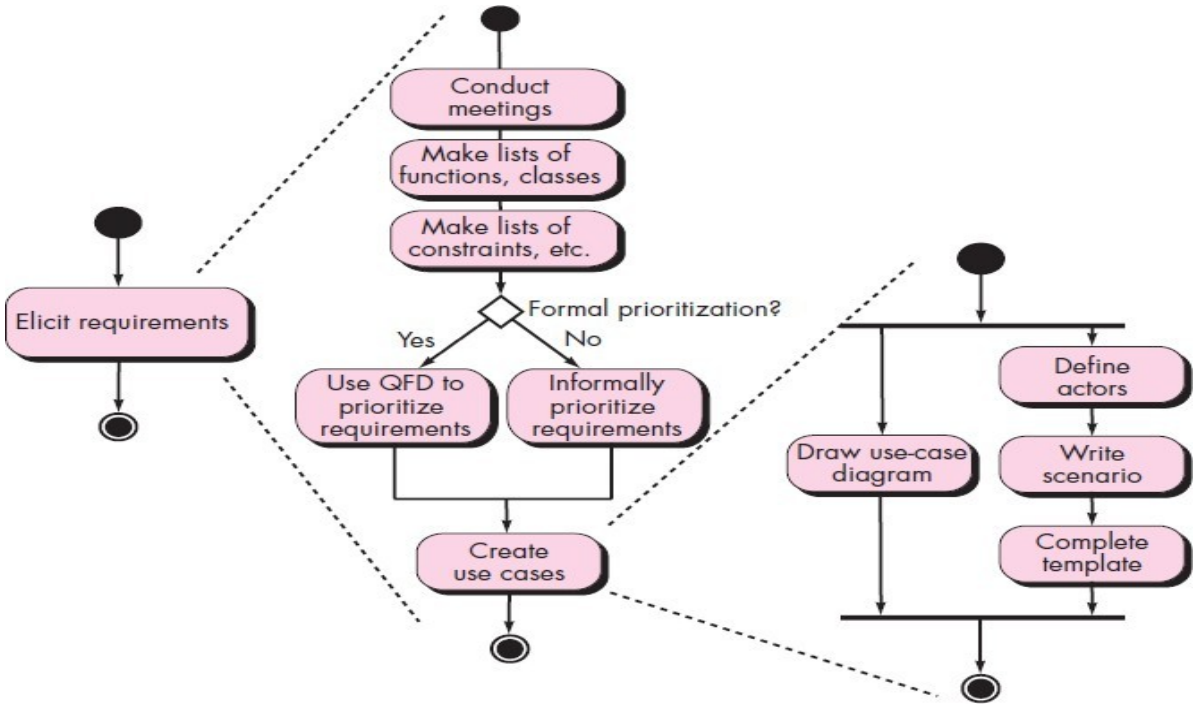
1. Scenario-based elements: The system is described from the user's point of view using a scenario-based approach.

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

3. Behavioral elements: The behavior of a computer-based system can have a profound effect on the design that is chosen and the implementation approach that is applied.

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

UML activity diagrams for eliciting requirements



Negotiating Requirements:

To negotiate the requirements of a system to be developed, it is necessary to identify conflicts and to resolve those conflicts. Conflicts can arise during all requirements engineering activities. For example, different stakeholders can utter contradicting requirements during elicitation. Requirements negotiations are about software that is to be developed. Thus, the specifics of software itself and of the process of designing and developing it have a strong influence on the medium that used in the negotiation process. Requirements negotiation should be used early on and repeated in later stages.

The main elements of Negotiating Requirements are as follows :

1. Identification of system key stakeholders
2. Determination of stakeholders' "win conditions"
3. Negotiate to reconcile stakeholders' win conditions into "win-win" result for all stakeholders including developers

The goal of requirement negotiation is to produce a win-win result before proceeding to subsequent software engineering activities.

The inception, elicitation, and elaboration tasks determine customer requirements in sufficient detail to proceed to subsequent software engineering activities.

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).

Validating Requirements:

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?

• 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?

• Do any requirements conflict with other requirements?

- Is each requirement achievable in the technical environment that will use 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?

Requirements Modeling identifies the **requirements** that a **software** application or system must meet in order to solve the business problem. **Requirements** are divided into functional and non-functional.

- ***Scenario-based models*** *of requirements from the point of view of various* system "actors"

- ***Data models*** *that depict the information domain for the problem*

- ***Class-oriented models*** *that represent object-oriented classes (attributes and* operations) and the manner in which classes collaborate to achieve system requirements

- ***Flow-oriented models*** *that represent the functional elements of the system* and how they transform data as it moves through the system

- ***Behavioral models*** *that depict how the software behaves as a consequence of* external "events"

Requirement modeling strategies

1. Flow Oriented Modeling
2. Class-based Modeling

   The flow oriented modeling represents how data objects are transformed at they move through the system. Derived from structured analysis, flow models use the data flow diagram, a modeling notation that depicts how input is transformed into output as data objects move through the system. Each software function that transforms data is described by a process specification or narrative.

Data Flow Diagram:

The data flow diagram represents the flows of data between different process in a business. It is a graphical technique that depicts information flow and transforms that are applied as data from input to output. It provides a simple, intuitive method for describing business processes without focusing on the details of computer systems. DFDs are attractive techniques because they provide what users do rather than what computers do.

In DFD, there are four symbols are used:

1. Process:
The circle represents the process. An activity that changes or transforms data flows. Since they transform incoming data to outgoing data, all processes must have inputs and outputs on a DFD.

2. Data Flow:
The labeled arrows indicate incoming and outgoing data flow. Movement of data between external entities, processes and data stores is represented with an arrow symbol, which indicates the direction of flow.

3. Data Store:
The rectangle represents an external entity. A data store does not generate any operations but simply holds data for later access.

4. External Entity:
In Data Flow Diagrams external entities produce and consume data that flows between the entity and the system being diagrammed.

These data flows are the inputs and outputs of the DFD. Data objects are represented by labeled arrows, and transformations are represented by circles. The DFD is presented in a hierarchical fashion. That is, the first data flow model (sometimes called a level 0 DFD or context diagram) represents the system as a whole. Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequent level.

2. Class-based Modeling

* Class based modeling represents the object. The system manipulates the operations.

* The elements of the class based model consist of classes and object,

attributes, operations, class – responsibility - collaborator (CRS) models.

- Classes are determined using underlining each noun or noun clause and enter it into the simple table.

Classes are found in following forms:

External entities: The system, people or the device generates the information that is used by the computer based system.

Things: The reports, displays, letter, signal are the part of the information domain or the problem.

Occurrences or events: A property transfer or the completion of series or robot movements occurs in the context of the system operation.

Roles: The people like manager, engineer, salesperson are interacting with the system.

Organizational units: The division, group, team are suitable for an application.

Places: The manufacturing floor or loading dock from the context of the problem and the overall function of the system.

Structures: The sensors, computers are defined a class of objects or related classes of objects.

* DFD shown in a hierarchical fashion. The DFD is split into different levels. It

also called as 'context level diagram'.

b) Control flow model

* Large class applications require a control flow modeling.

* The application creates control information instated of reports or displays.

* The applications process the information in specified time.

* An event is implemented as a Boolean value.

c) **Control Specification**

* A short term for control specification is CSPEC.

* It represents the behavior of the system.

* The state diagram in CSPEC is a sequential specification of the behavior.

* The state diagram includes states, transitions, events and activities.

- State diagram shows the transition from one state to another state if a particular event has occurred.

d) **Process Specification**

***** A short term for process specification is PSPEC.

* The process specification is used to describe all flow model processes.

* The content of process specification consists narrative text, Program Design Language(PDL) of the process algorithm, mathematical equations, tables or UML activity diagram.

Analysis Rules of Thumb:

1. *The model should focus on requirements that are visible within the problem or business domain.*

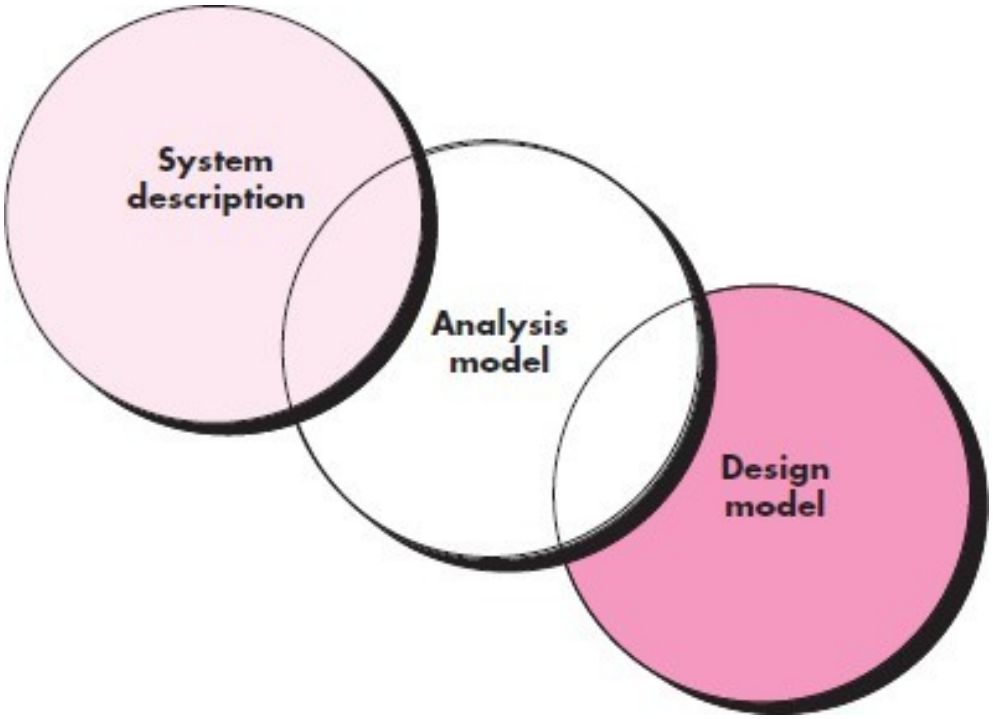
   2. Each element of the requirements model should add to an overall understanding of software requirements.

   3. Delay consideration of infrastructure and other nonfunctional models until design. That is, a database may be required.

   4. Minimize coupling throughout the system.

   5. Be certain that the requirements model provides value to all stakeholders.

   6. Keep the model as simple as it can be.

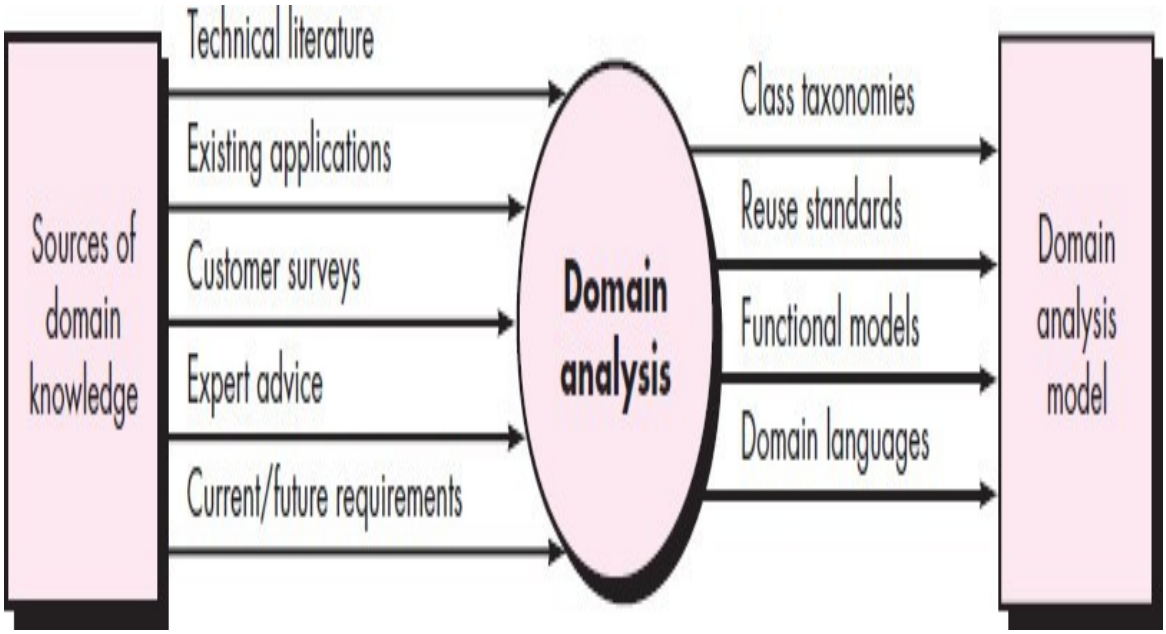


Input and output for domain analysis:

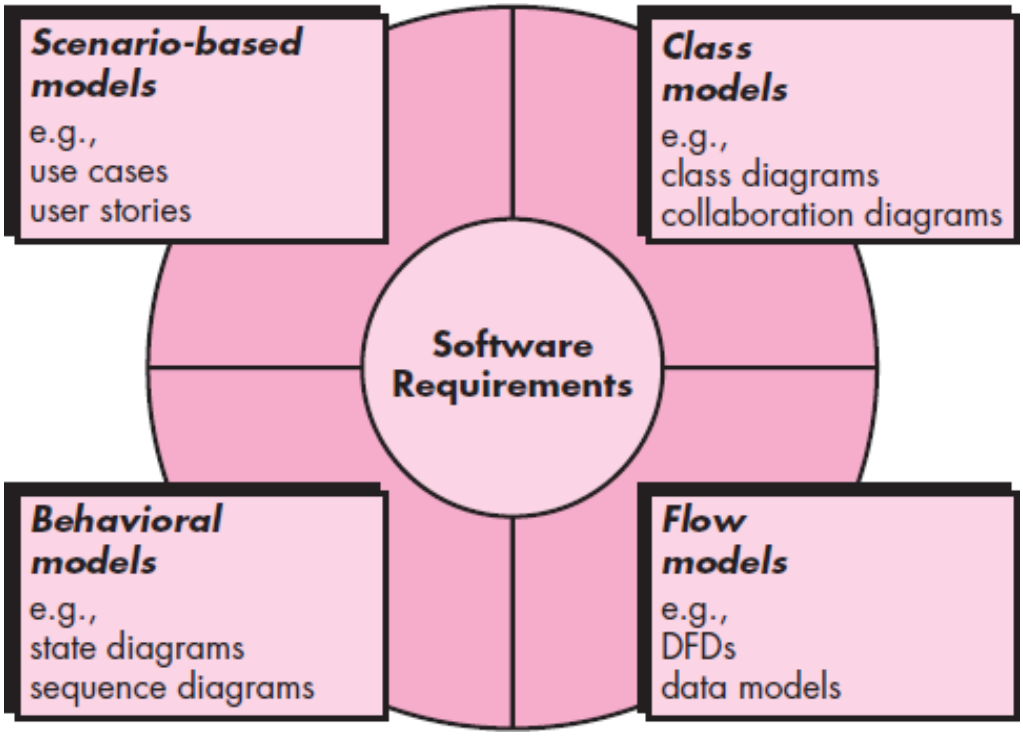
Domain analysis is the process by which a software engineer learns background information. He or she has to learn sufficient information so as to be able to understand the problem and make good decisions during requirements analysis and other stages of the software engineering process. The word 'domain' in this case means the general field of business or technology in which the customers expect to be using the software.

Some domains might be very broad, such as 'airline reservations', 'medical diagnosis', and 'financial analysis'. Others are narrower, such as 'the manufacturing of paint' or 'scheduling meetings'. People who work in a domain and who have a deep knowledge of it (or part of it), are called *domain experts*. Many of these people may become customers or users.

- **Faster development:** You will be able to communicate with the stakeholders more effectively; hence you will be able to establish requirements more rapidly. Having performed domain analysis will help you to focus on the most important issues.
- **Better system:** Knowing the subtleties of the domain will help ensure that the solutions you adopt will more effectively solve the customer's problem. You will make fewer mistakes, and will know which procedures and standards to follow. The analysis will give you a global picture of the domain of application; this will lead to better abstractions and hence improved designs.
- **Anticipation of extensions:** Armed with domain knowledge, you will obtain insights into emerging trends and you will notice opportunities for future development. This will allow you to build a more adaptable system.



Elements of the analysis model:



Scenario-Based Modeling:

Scenario-based modeling is one of the sub-stages of requirements *modeling*. It is also typically the first stage of requirements *modeling*, as it identifies the primary use cases for the proposed *software* system or application

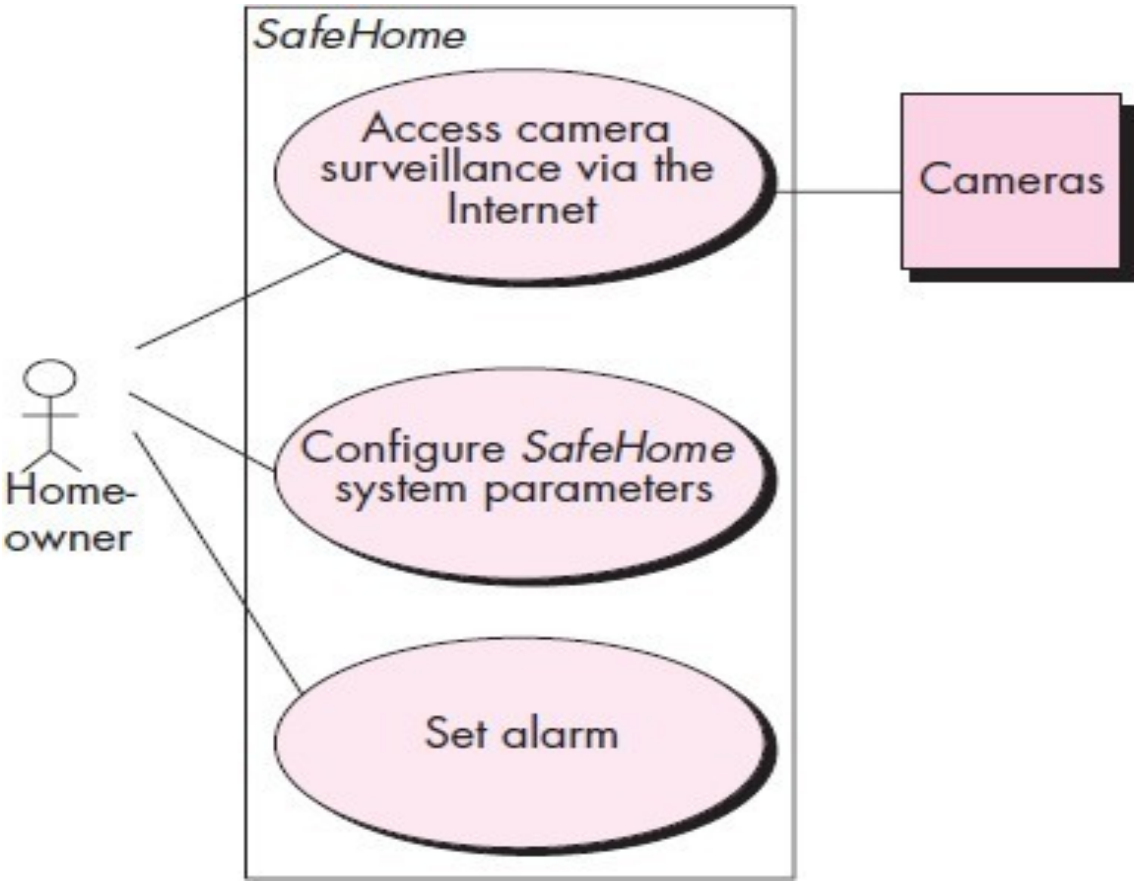
Following functions that are performed by the homeowner actor:

• Select camera to view.
• Request thumbnails from all cameras.
• Display camera views in a PC window.
• Control pan and zoom for a specific camera.
• Selectively record camera output.
• Replay camera output.
• Access camera surveillance via the Internet.

The use-cases are simply an aid to defining what exists outside the system and what should be performed by the system. And for this use cases, we need to have the answers for below questions :

1. What should we write about ?
2. How much should we write about it ?
3. How detailed should we make our description?
4. How should we organize the description?

Use-case diagram for the *SafeHome* system:



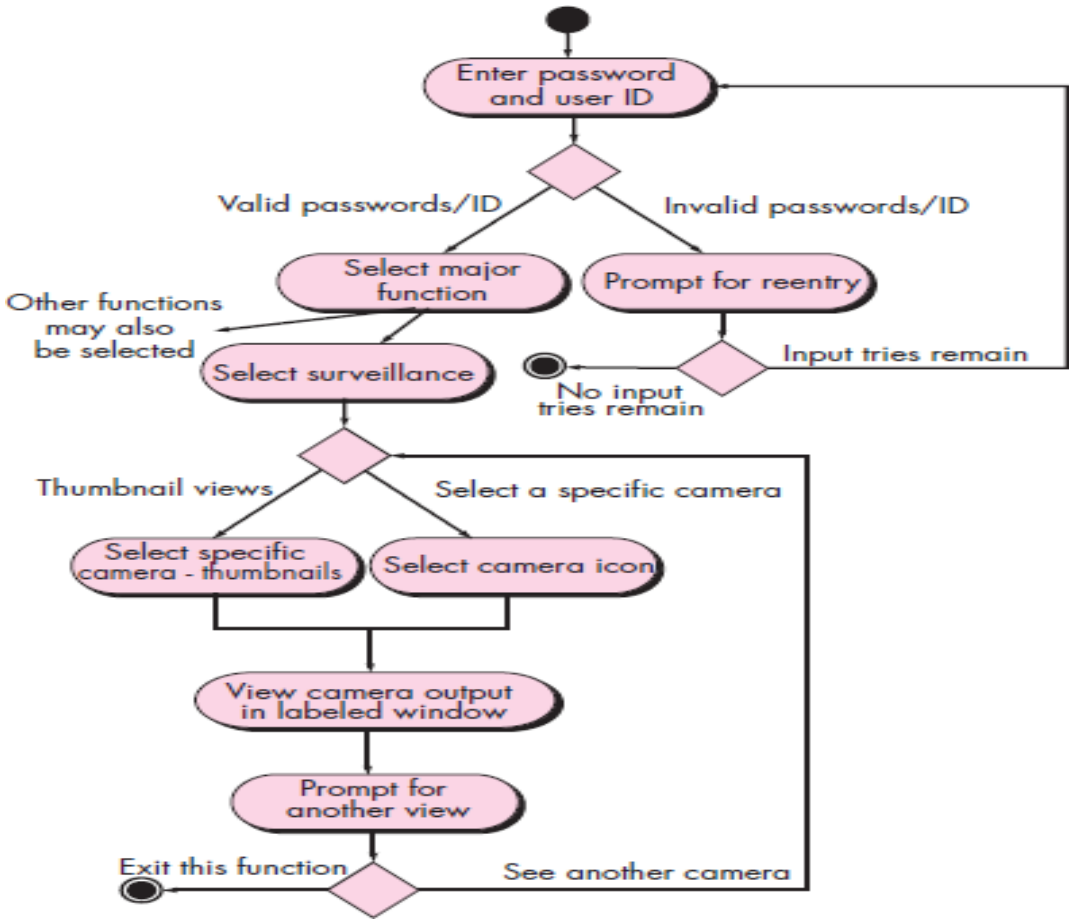
Actor: homeowner

1. The homeowner logs onto the *SafeHome Products website.*
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the “surveillance” from the major function buttons.
6. The homeowner selects “pick a camera.”
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the “view” button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

UML Models that Supplement the Use Case:

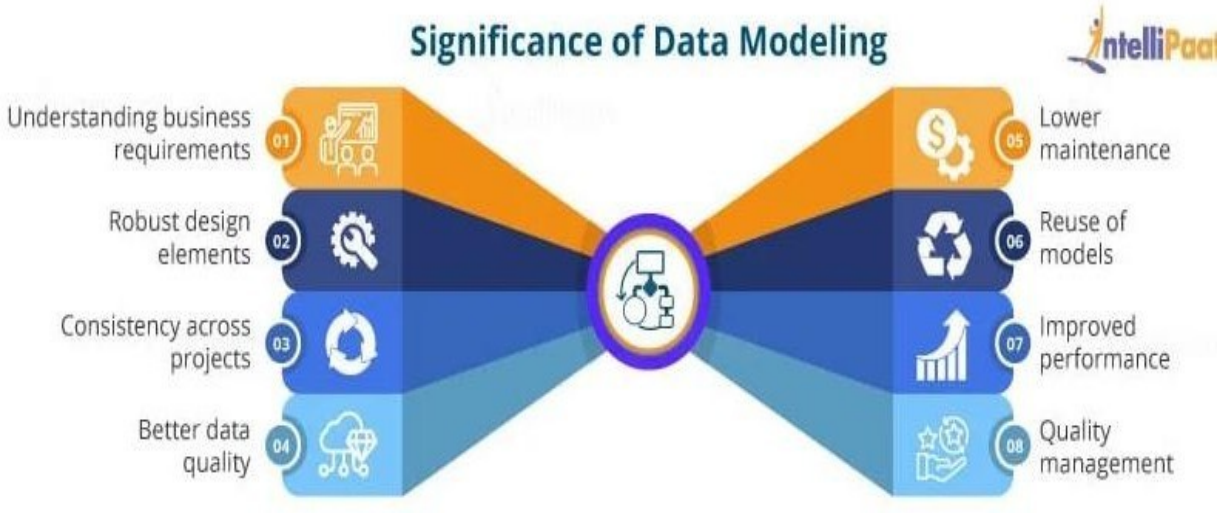
1. There are many requirements modeling situations in which a text-based model—even one as simple as a use case
2. The UML activity diagram supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario.
3. Similar to the flowchart, an activity diagram uses rounded rectangles to imply a specific system function, arrows to represent flow through the system, decision diamonds to depict a branching decision and solid horizontal lines to indicate that parallel activities are occurring.

Activity diagram for Access camera surveillance:



Data Modeling Concepts:

Data modeling, sometimes also called information **modeling**, is the process of visually representing what **data** the application or system will use, and how it will flow. The resulting diagram or other visual representation is meant to be designed in a way that is as easy to understand as possible.



Three Perspectives of a Data Model:

1. Conceptual Model

This Conceptual model defines what needs to be present in the structure of the model in order to define and organize business concepts. It mainly focuses on business-oriented entries, attributes, and relations. It is basically designed by Data Architects and Business Stakeholders.

2. Logical Model

The logical model defines how the model should be implemented. It broadly includes all kinds of data that need to be captured such as tables, columns, etc. This model is generally designed by Business Analysts and Data Architects.

3. Physical Model

The physical model defines how to implement a data model with the help of the database management system. It outlines the implementation methodology in terms of tables, operations, indexes, partitioning, etc. It is created by Database Administrators and Developers.

Class-Based Modeling:

In the context of **software engineering**, requirements **modeling** examines the requirements a proposed **software** application or system must meet in order to be successful. **Class-based modeling** takes the use case and extracts from it the **classes**, attributes, and operations the application will use.

How do you Identify Classes?

• *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.

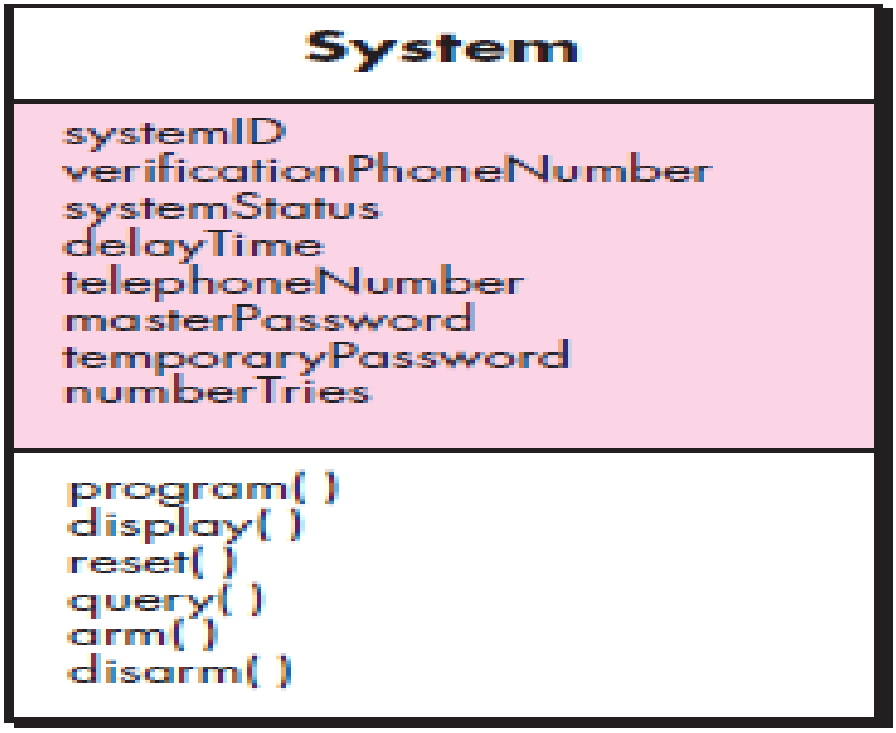
• *Occurrences or events (e.g., a property transfer or the completion of a series* of robot movements) that occur within the context of system operation.

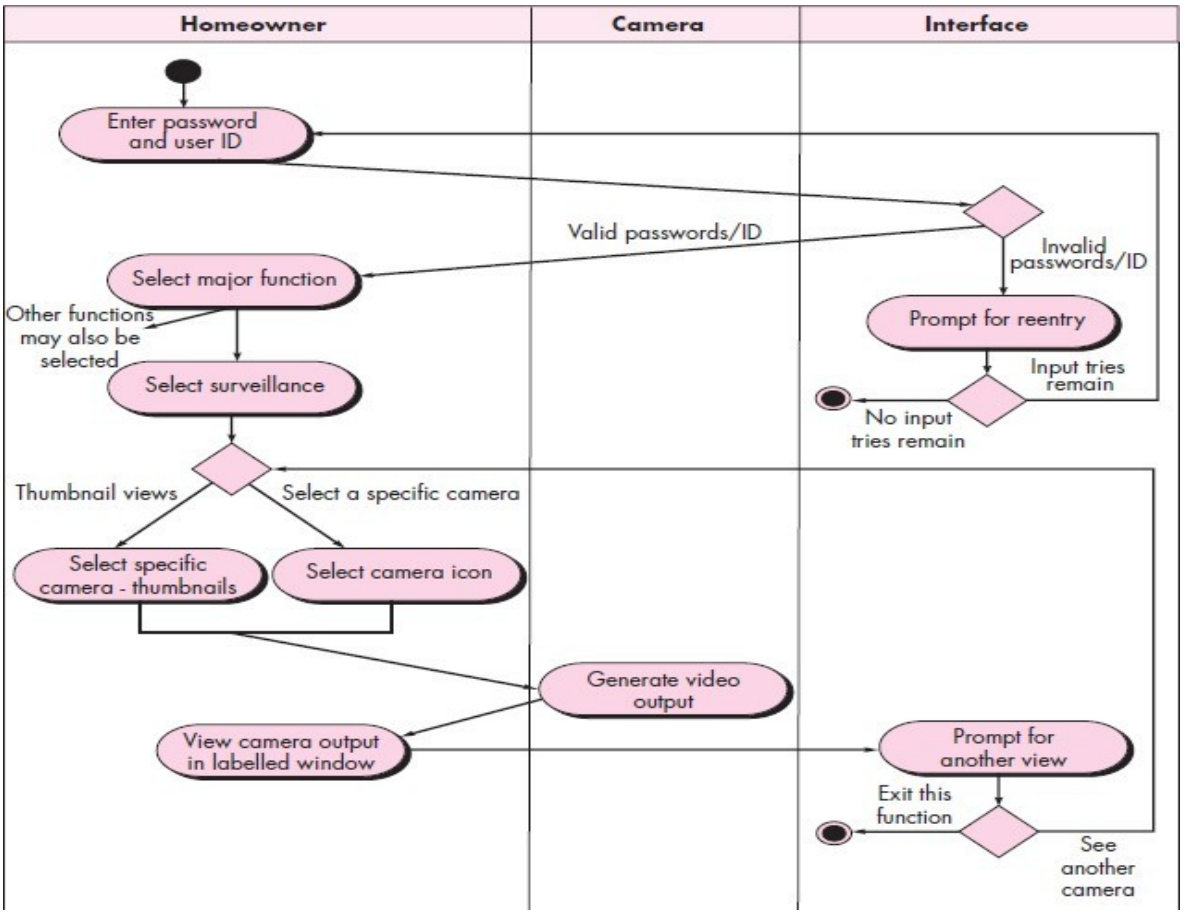
• *Roles (e.g., manager, engineer, salesperson) played by people who interact* with the system.

• *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.





Flow-Oriented Modeling:

Data **flow** diagram (DFD) represents the **flows** of data between different processes in a business. It is a graphical technique that depicts information **flow** and the transforms that are applied as data move form input to output.

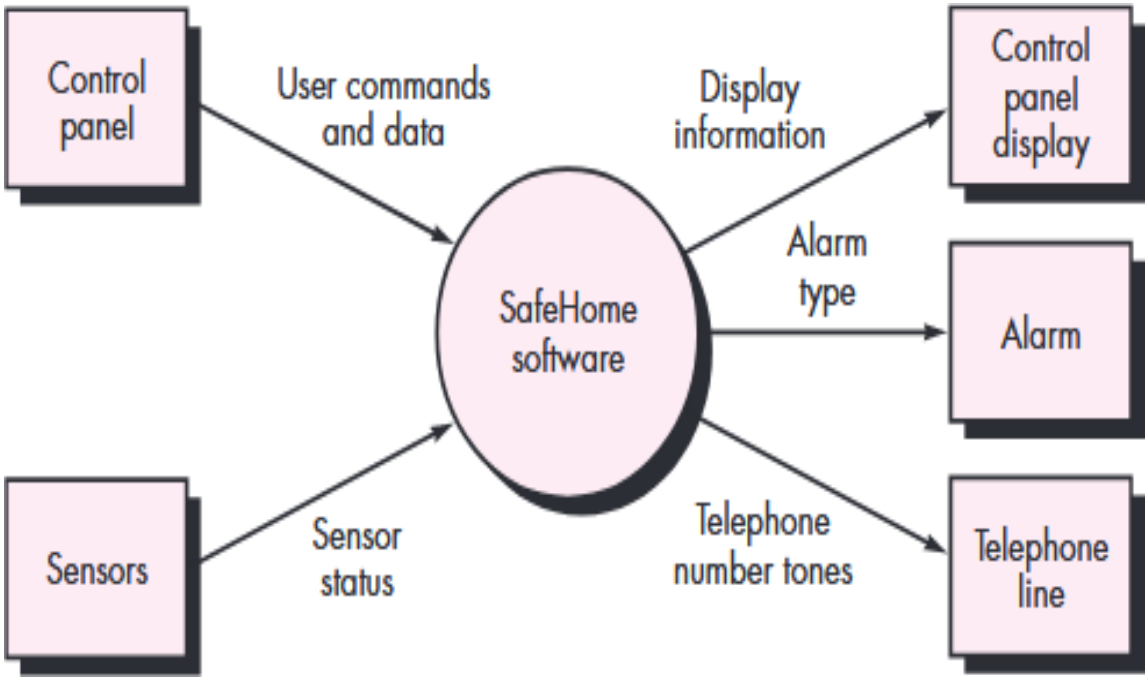
The flow oriented modeling represents how data objects are transformed at they move through the system. Derived from structured analysis, flow models use the data flow diagram, a modeling notation that depicts how input is transformed into output as data objects move through the system. Each software function that transforms data is described by a process specification or narrative.

1. Level 0 DFD : This shows an overall view of the system. Level 0 DFD is also known as context diagram.

2. Level 1 DFD : This elaborates level a DFD and splits the process into a detailed form.2

3. **Level 2 DFD** : This elaborates level 1 DFD and displays the process(s) in a detailed form.

4. **Level 3 DFD** : This elaborates level 2 DFD and displays the process(s) in a detailed form.



State diagram for the Control Panel:

A state diagram shows the behavior of classes in response to external stimuli. Specifically a state diagram describes the behavior of a single object in response to a series of events in a system. Sometimes it's also known as a Harel state chart or a state machine diagram. This UML diagram models the dynamic flow of control from state to state of a particular object within a system

States

States represent situations during the life of an object. You can easily illustrate a state in SmartDraw by using a rectangle with rounded corners.

Transition

A solid arrow represents the path between different states of an object. Label the

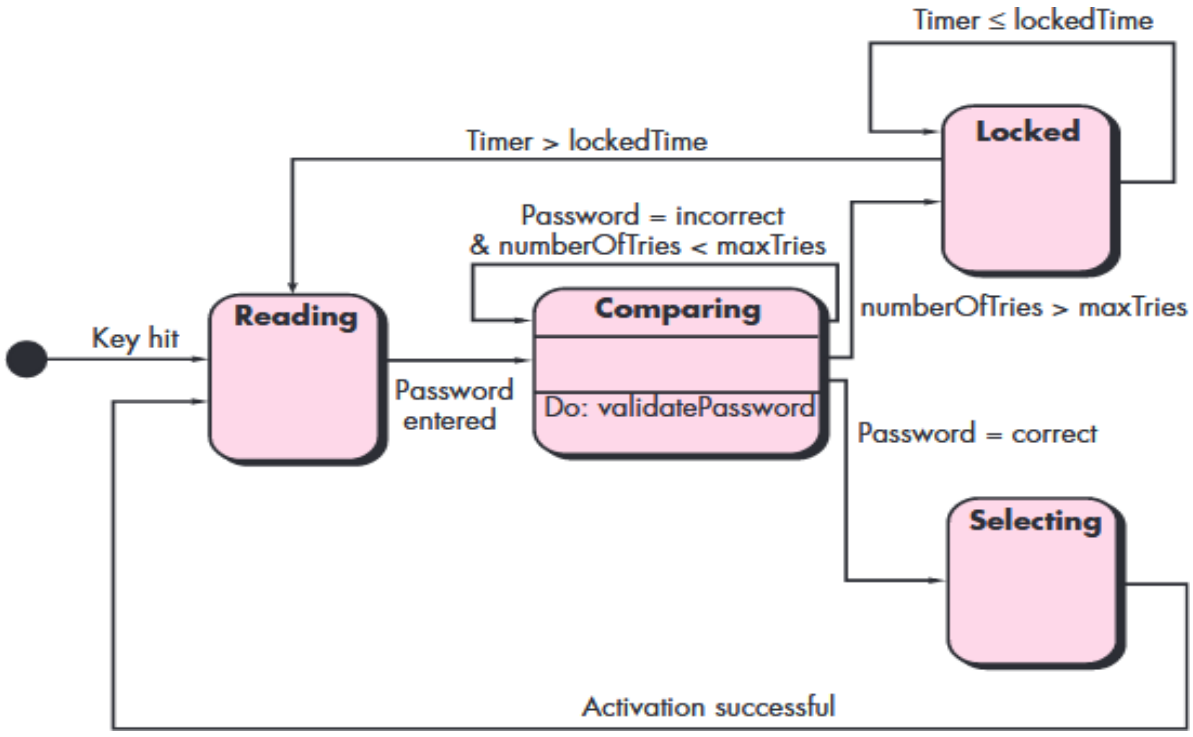
transition with the event that triggered it and the action that results from it. A state can have a transition that points back to itself.

Initial State

A filled circle followed by an arrow represents the object's initial state.

Final State

An arrow pointing to a filled circle nested inside another circle represents the object's final state.



Requirements Modeling for WebApps:

Requirements analysis does take time, but solving the wrong problem takes even more time. The question for every WebApp developer is simple—are you sure you understand the requirements of the problem? If the answer is an unequivocal "yes," then it may be possible to skip requirements modeling, but if the answer is "no," then requirements modeling should be performed.

The degree to which requirements modeling for WebApps is emphasized depends on the following factors:

•Size and complexity of WebApp increment.

•Number of stakeholders.

•Size of the WebApp team.

•Degree to which members of the WebApp team have worked together before.

•Degree to which the organization's success is directly dependent on the success of the WebApp.

There are five main classes of models:

1. **Content model**—identifies the full spectrum of content to be provided by the WebApp. Content includes text, graphics and images, video, and audio data.
2. **Interaction model**—describes the manner in which users interact with the WebApp.
3. **Functional model**—defines the operations that will be applied to WebApp content and describes other processing functions that are independent of content but necessary to the end user.
4. **Navigation model**—defines the overall navigation strategy for the WebApp.
5. **Configuration model**—describes the environment and infrastructure in which the WebApp resides.