Chapter 8

- **Requirements Modeling**
  — A Recommended Approach

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 9/e

by Roger S. Pressman and Bruce R. Maxim

Requirements Analysis

- Requirements analysis
  - specifies software's operational characteristics
  - indicates software's interface with other system elements
  - establishes constraints that software must meet

- Requirements analysis allows the software engineer (called an *analyst* or *modeler* in this role) to:
  - elaborate on basic requirements established during earlier requirement engineering tasks
  - build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.

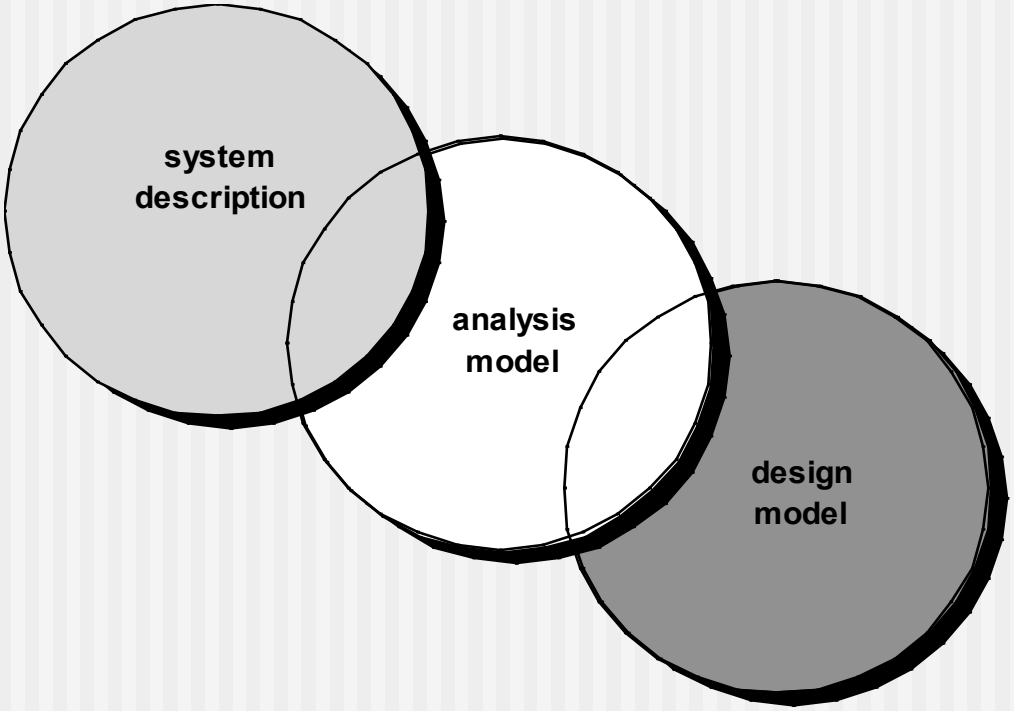
Requirements Modeling

- Scenario-based
  - system from the user's point of view
- Class-oriented
  - defines objects, attributes, and relationships
- Behavioral
  - show the impact of events on the system states
- Data (Structure Analysis Method)
  - shows how data are transformed inside the system
- Flow-oriented (Structure Analysis Method)
  - shows how data are transformed inside the system

Overall Objectives & Philosophy

- Requirements are not framework, not design, not interface.
- Requirements are need.
- Three primary objectives of requirements:
  - To describe what the customer requires?
  - To establish a basis for the creation of a software design.
  - To define a set of requirements that can be validated once the software is built.

A Bridge



Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure and other non-functional models until design.
- Minimize coupling throughout the system.
- Be certain that the analysis model provides value to all stakeholders.
- Keep the model as simple as it can be.

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 . . .

Donald Firesmith

Domain Analysis Process

- Define the domain to be investigated.
- Collect a representative sample of applications in the domain.
- Analyze each application in the sample.
- Develop an analysis model for the objects.
- Validate this model by your customer.

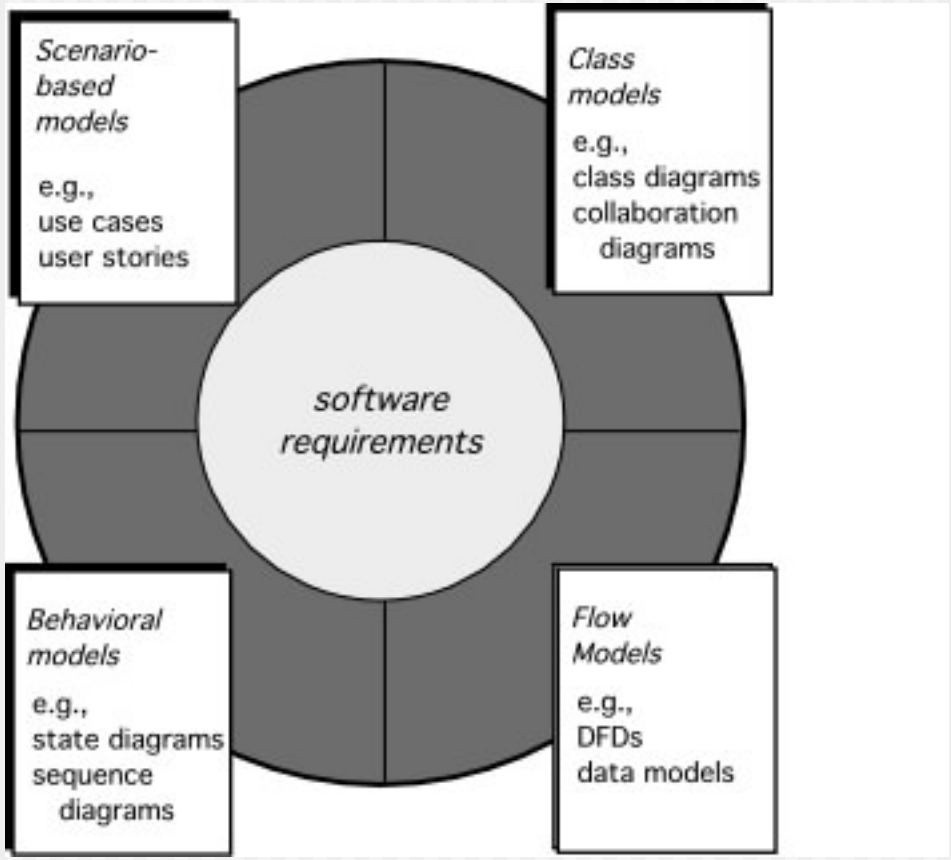
Requirements Modeling Approaches

- Structured analysis
  - Data and 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 manner that shows how they transform data as data objects flow through the system.

Requirements Modeling Approaches

- Object-oriented analysis
  - Each element of the requirements model present the problem from a different point of view.
  - 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(some hierarchical) 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.

Elements of Requirements Analysis



Scenario-Based Modeling

"[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases)." Ivar Jacobson

(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?

What to Write About?

- Inception and elicitation—provide you with the information you'll need to begin writing use cases.
- Requirements gathering meetings, QFD, and other requirements engineering mechanisms are used to
  - identify stakeholders
  - define the scope of the problem
  - specify overall operational goals
  - establish priorities
  - outline all known functional requirements, and
  - describe the things (objects) that will be manipulated by the system.
- To begin developing a set of use cases, list the functions or activities performed by a specific actor.

How Much to Write About?

- As further conversations with the stakeholders progress, the requirements gathering team develops use cases for each of the functions noted.
- In general, use cases are written first in an informal narrative fashion.
- If more formality is required, the same use case is rewritten using a structured format similar to the one proposed.

Use-Cases

- a scenario that describes a “thread of usage” for a system
- *actors* represent roles people or devices play as the system functions
- *users* can play a number of different roles for a given scenario

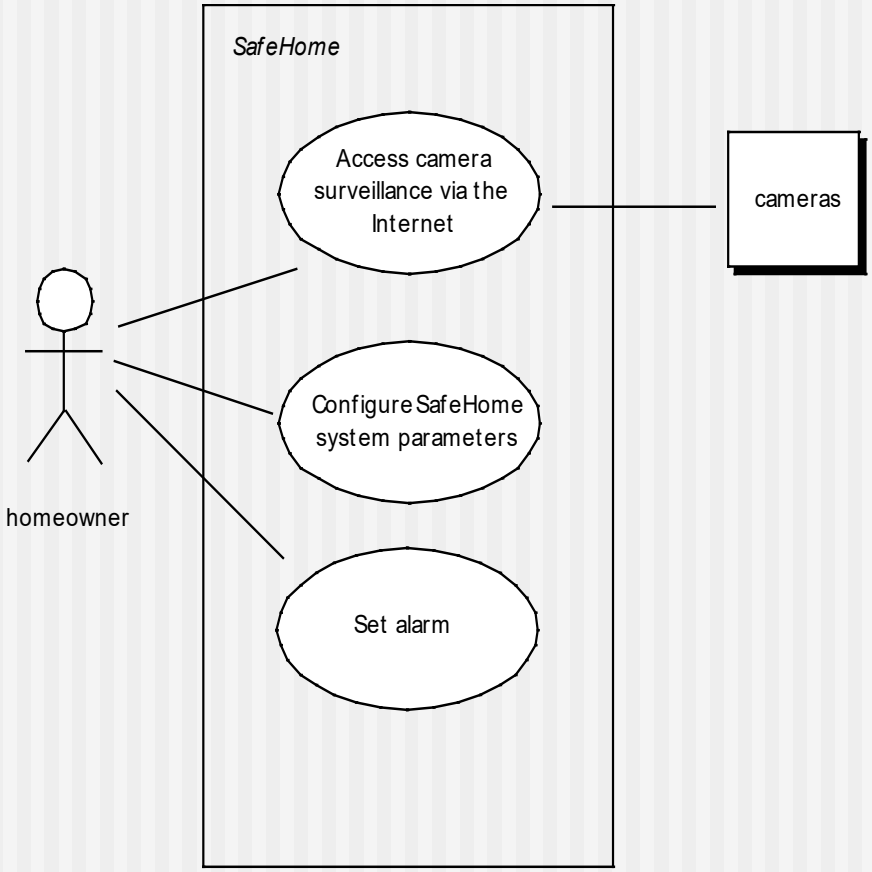
Developing a Use-Case

- What are the main tasks or functions that are performed by the actor?
- What system information will the 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?

Reviewing a Use-Case

- Use-cases are written first in narrative form and mapped to a template if formality is needed
- Each primary scenario should be reviewed and refined to see if alternative interactions are possible
  - Can the actor take some other action at this point?
  - Is it possible that the actor will encounter an error condition at some point? If so, what?
  - Is it possible that the actor will encounter some other behavior at some point? If so, what?

Use-Case Diagram

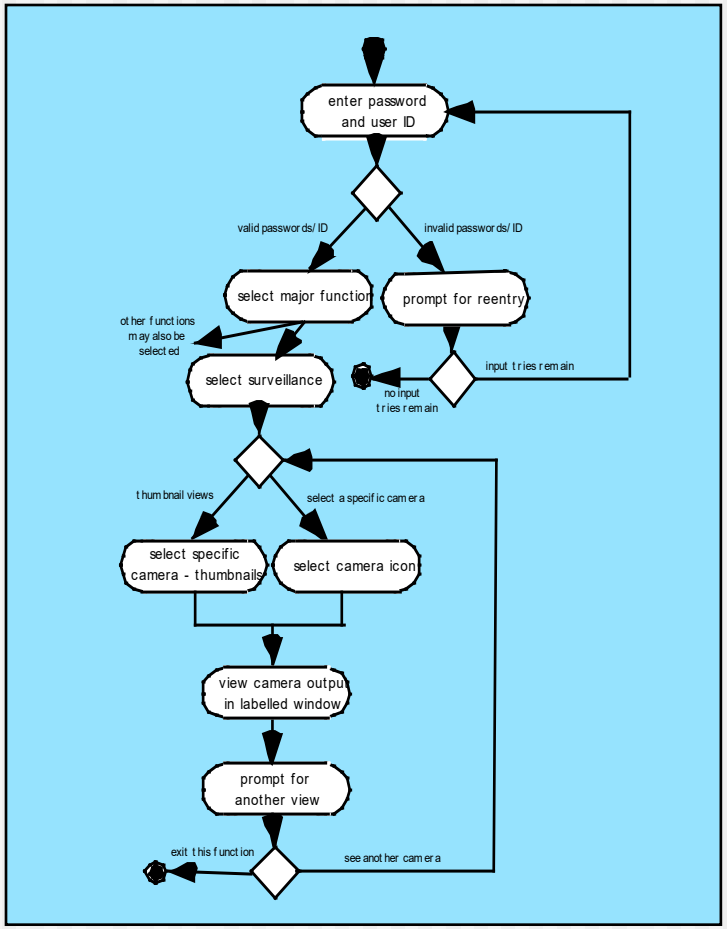


Exceptions

- Describe situations (failures or user choices) that cause the system to exhibit unusual behavior
- Brainstorming should be used to derive a reasonably complete set of exceptions for each use case
- Are there cases where a validation function occurs for the use case?
  - Are there cases where a supporting function (actor) fails to respond appropriately?
  - Can poor system performance result in unexpected or improper use actions?
- Handling exceptions may require the creation of additional use cases

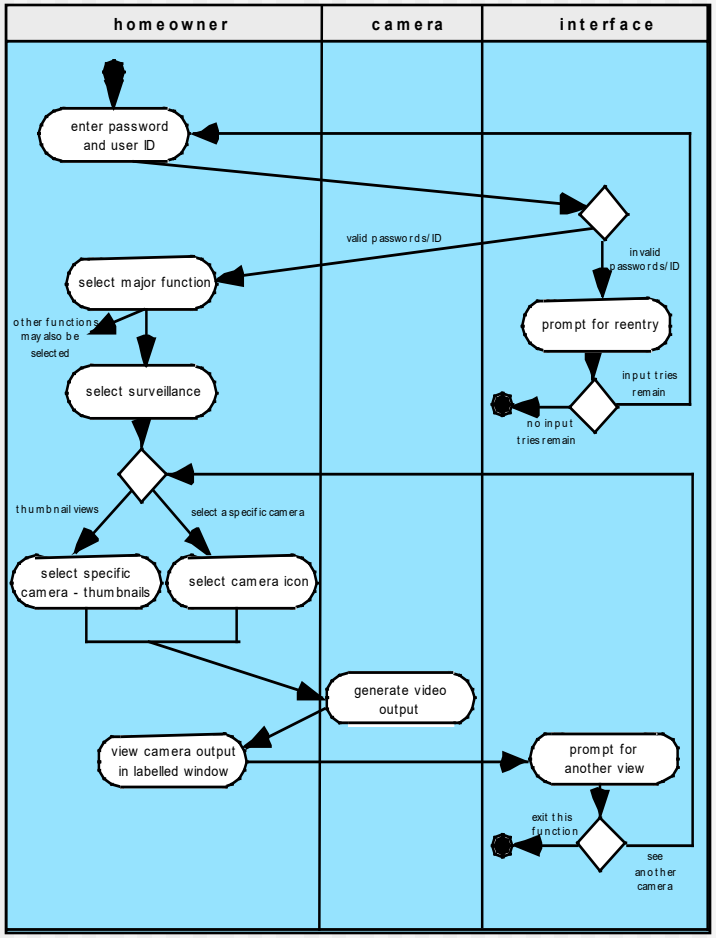
Activity Diagram

Supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario



Swimlane Diagrams

Allows the modeler to represent the flow of activities described by the use-case and at the same time indicate which actor (if there are multiple actors involved in a specific use-case) or analysis class has responsibility for the action described by an activity rectangle



Class-Based Modeling

- Class-based modeling represents:
  - objects that the system will manipulate
  - operations (also called methods or services) that will be applied to the objects to effect the manipulation
  - relationships (some hierarchical) between the objects
  - collaborations that occur between the classes that are defined.
- The elements of a class-based model include classes and objects, attributes, operations, CRC models, collaboration diagrams and packages.

Identifying Analysis Classes

- Examining the usage scenarios developed as part of the requirements model and perform a "grammatical parse" [Abb83]
  - Classes are determined by underlining each noun or noun phrase and entering it into a simple table.
  - Synonyms should be noted.
  - If the class (noun) is required to implement a solution, then it is part of the solution space; otherwise, if a class is necessary only to describe a solution, it is part of the problem space.
- But what should we look for once all of the nouns have been isolated?

Manifestations of Analysis Classes

- *Analysis classes* manifest themselves in one of the following ways:
  - *External entities* (e.g., other systems, devices, people) that produce or consume information
  - *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
  - *Structures* (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects

Potential Classes

- *Retained information.* The potential class will be useful during analysis only if information about it must be remembered so that the system can function.
- *Needed services.* The potential class must have a set of identifiable operations that can change the value of its attributes in some way.
- *Multiple attributes.* During requirement analysis, the focus should be on "major" information; a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity.
- *Common attributes.* A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.
- *Common operations.* A set of operations can be defined for the potential class and these operations apply to all instances of the class.
- *Essential requirements.* External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as classes in the requirements model.

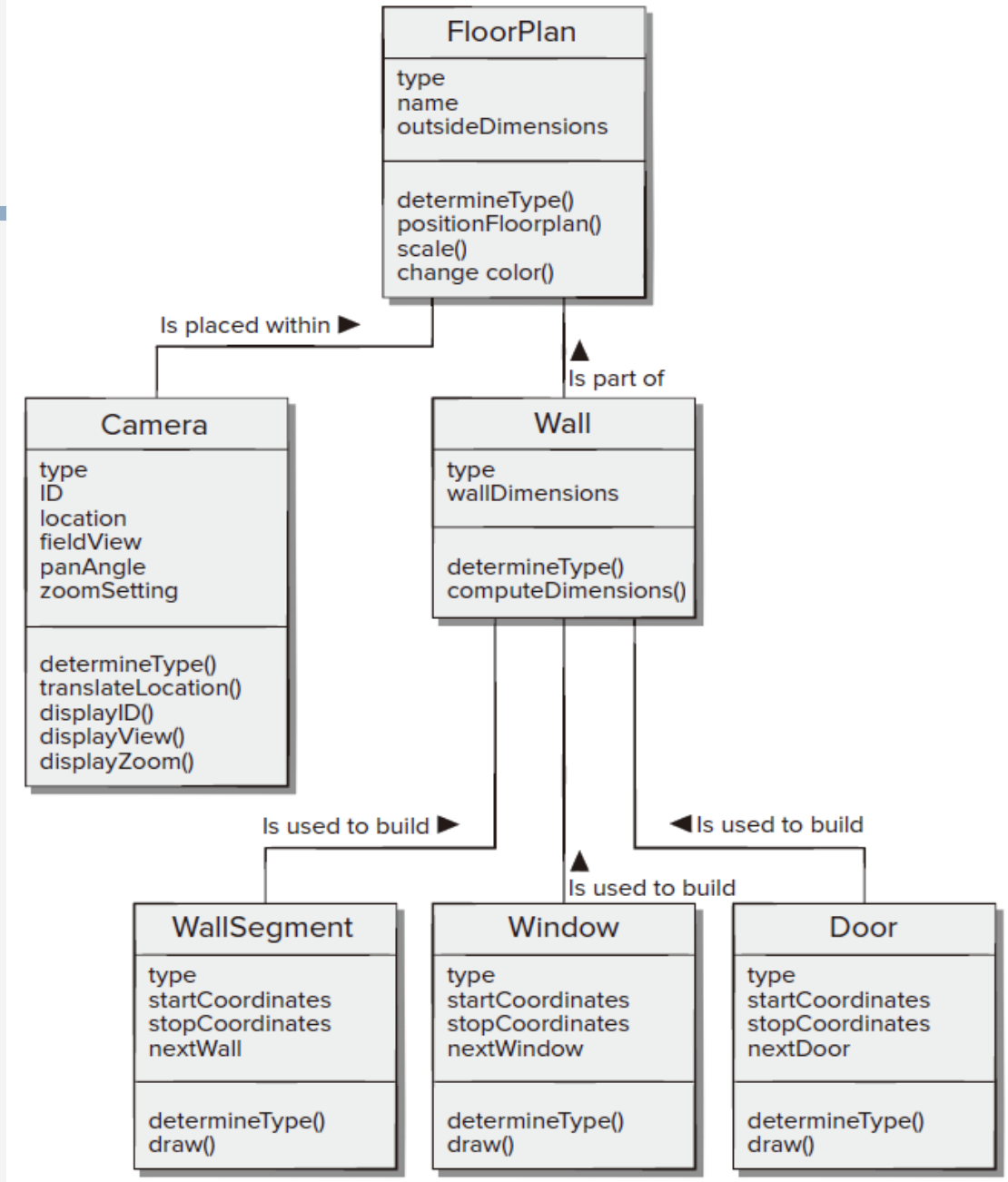
Defining Attributes

- *Attributes* describe a class that has been selected for inclusion in the analysis model.
  - build two different classes for professional baseball players
    - **For Playing Statistics software:** name, position, batting average, fielding percentage, years played, and games played might be relevant
    - **For Pension Fund software:** average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.

Defining Operations

- Do a grammatical parse of a processing narrative and look at the verbs
- Operations can be divided into four broad categories:
  - (1) operations that manipulate data in some way (e.g., adding, deleting, reformatting, selecting)
  - (2) operations that perform a computation
  - (3) operations that inquire about the state of an object, and
  - (4) operations that monitor an object for the occurrence of a controlling event.

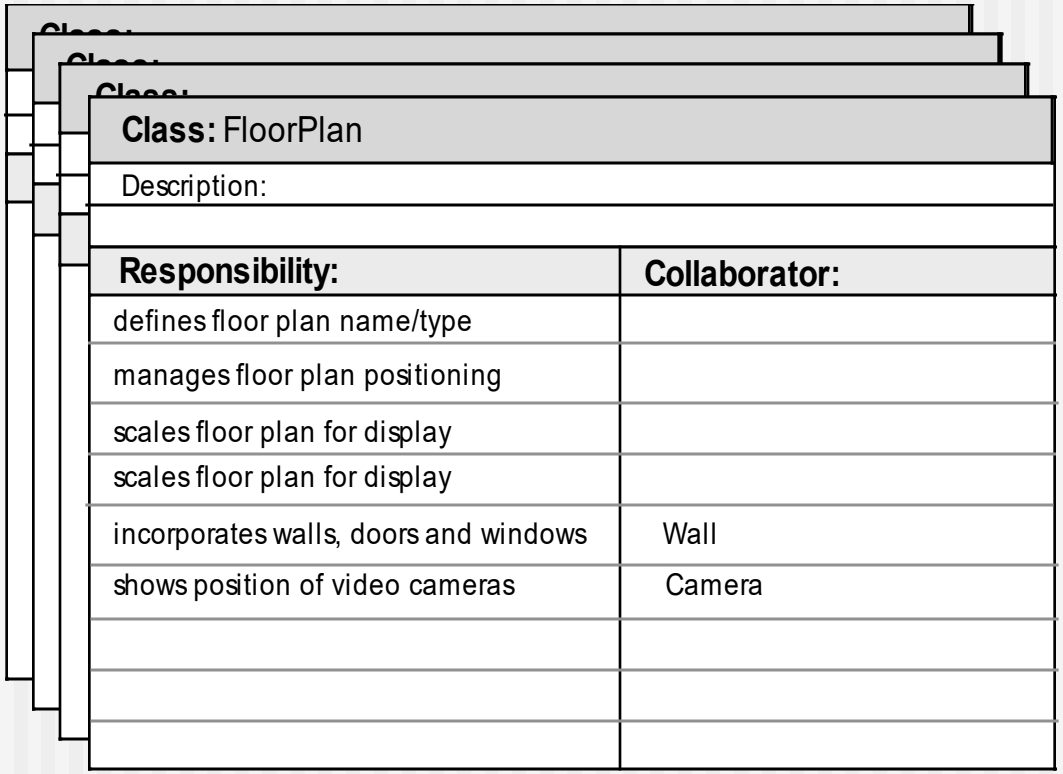
UML Class Models



CRC Models

- *Class-responsibility-collaborator (CRC) modeling* [Wir90] provides a simple means for identifying and organizing the classes that are relevant to system or product requirements. Ambler [Amb95] 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.

CRC Modeling



Class Types

- *Entity classes*, also called *model* or *business* classes, are extracted directly from the statement of the problem (e.g., FloorPlan and Sensor).
- *Boundary classes* are used to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used.
- *Controller classes* manage a "unit of work" [UML03] from start to finish. That is, controller classes can be designed to manage
  - the creation or update of entity objects;
  - the instantiation of boundary objects as they obtain information from entity objects;
  - complex communication between sets of objects;
  - validation of data communicated between objects or between the user and the application.

Responsibilities

- System intelligence should be distributed across classes to best address the needs of the problem
- Each responsibility should be stated as generally as possible
- Information and the behavior related to it should reside within the same class
- Information about one thing should be localized with a single class, not distributed across multiple classes.
- Responsibilities should be shared among related classes, when appropriate.

Collaborations

- Classes fulfill their responsibilities in one of two ways:
  - A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
  - a class can collaborate with other classes.
- Collaborations identify relationships between classes
- Collaborations are identified by determining whether a class can fulfill each responsibility itself
- three different generic relationships between classes [WIR90]:
  - the *is-part-of* relationship
  - the *has-knowledge-of* relationship
  - the *depends-upon* relationship

Reviewing the CRC Model

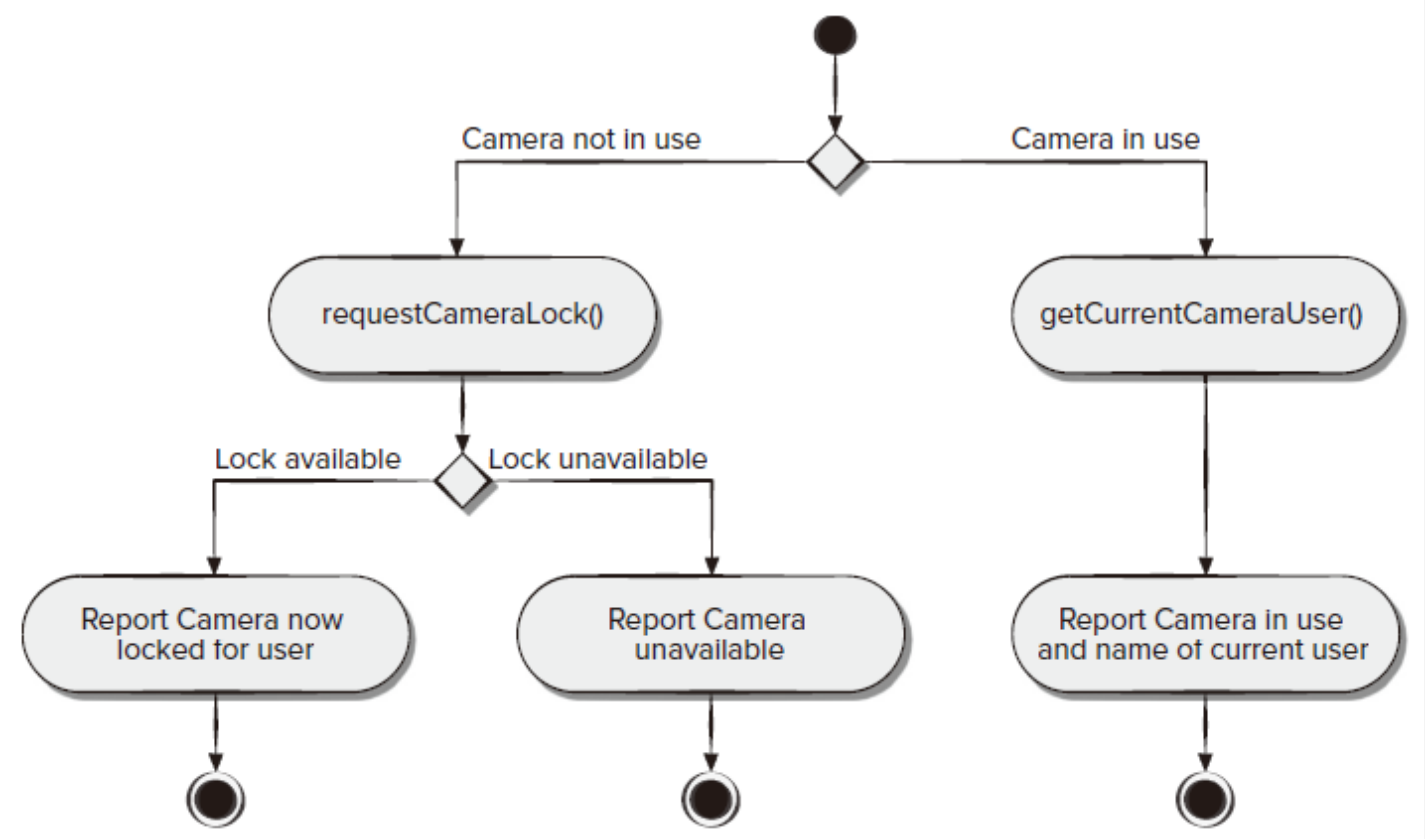
- All participants in the review (of the CRC model) are given a subset of the CRC model index cards.
  - Cards that collaborate should be separated (i.e., no reviewer should have two cards that collaborate).
- All use-case scenarios (and corresponding use-case diagrams) should be organized into categories.
- The review leader reads the use-case deliberately.
  - As the review leader comes to a named object, she passes a token to the person holding the corresponding class index card.

Reviewing the CRC Model

- When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card.
  - The group determines whether one (or more) of the responsibilities satisfies the use-case requirement.
- If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, modifications are made to the cards.
  - This may include the definition of new classes (and corresponding CRC index cards) or the specification of new or revised responsibilities or collaborations on existing cards.

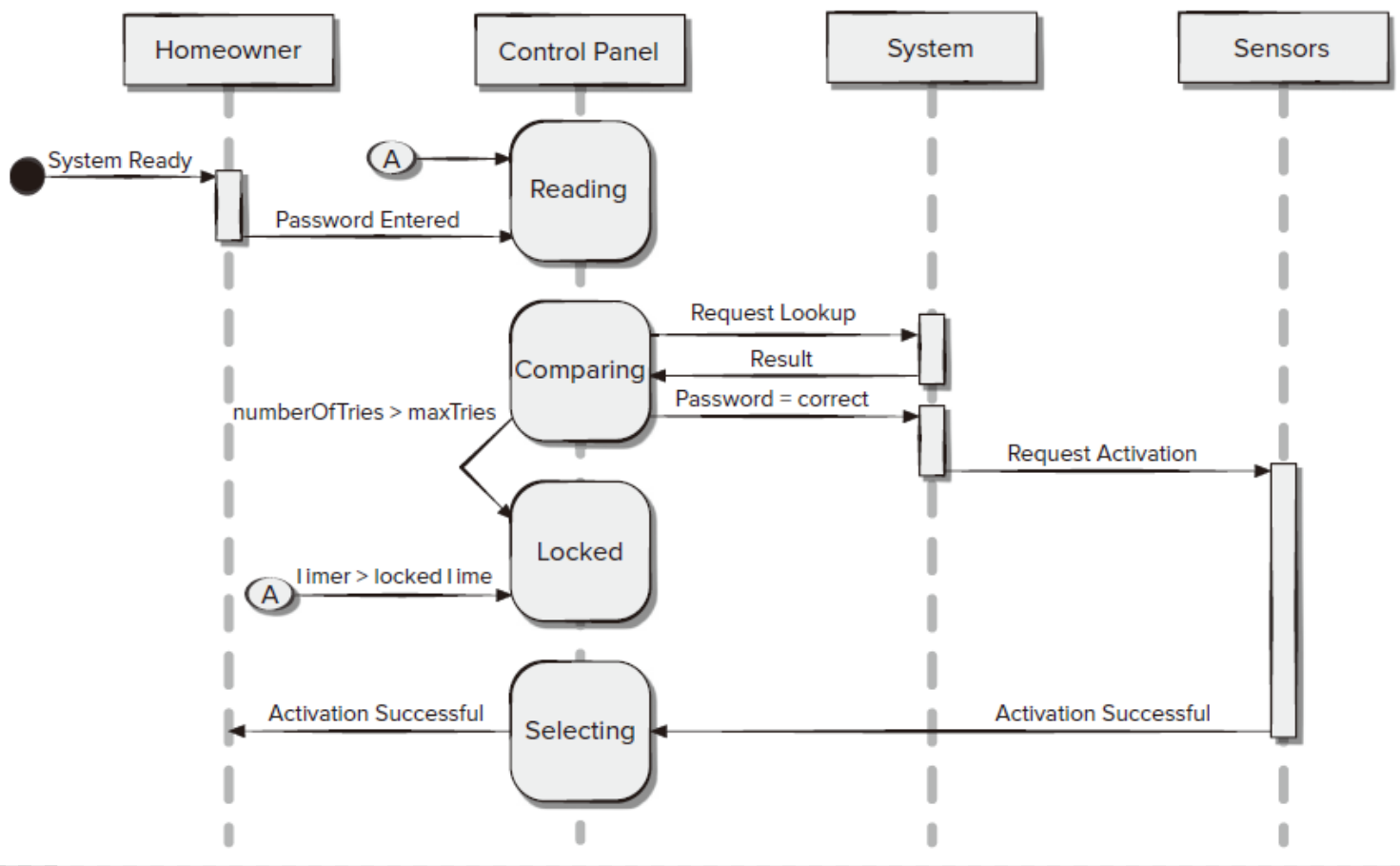
Functional Modeling

- A Procedural View

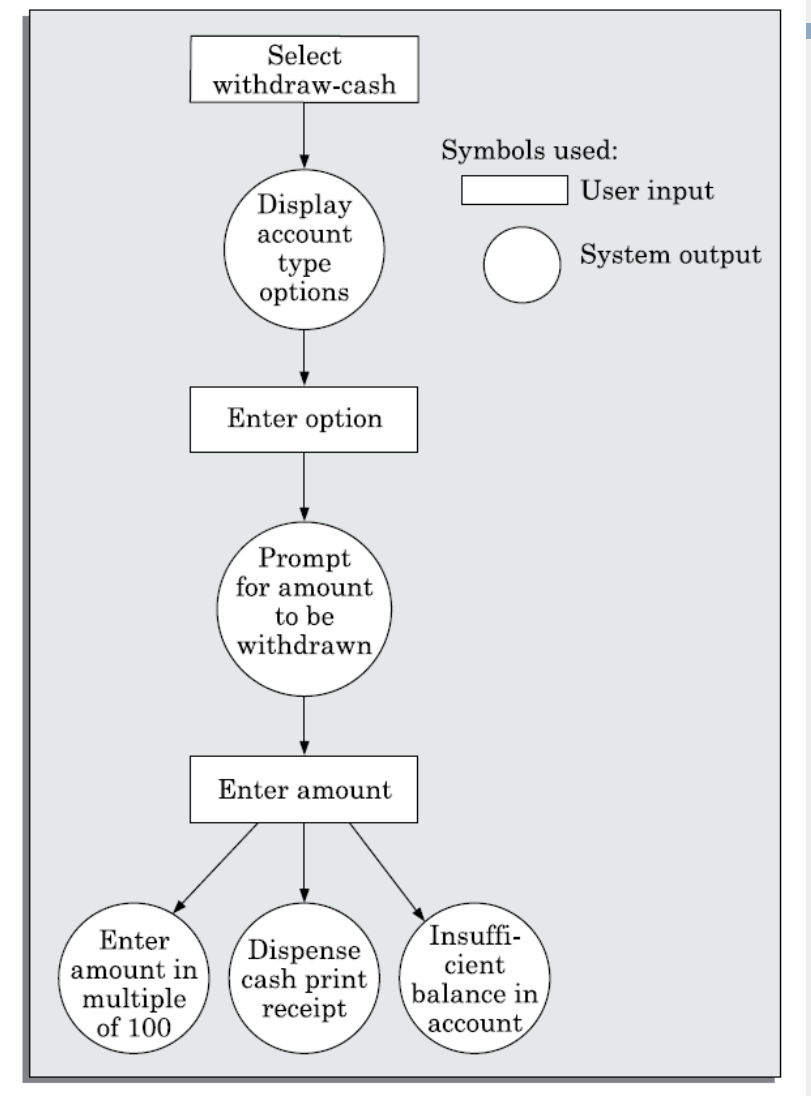


Functional Modeling

- UML Sequence Diagrams



Sequence Diagram for ATM



Behavioral Modeling

- The behavioral model indicates how software will respond to external events or stimuli. To create the model, the analyst must perform the following steps:
  - Evaluate all use-cases to fully understand the sequence of interaction within the system.
  - Identify events that drive the interaction sequence and understand how these events relate to specific objects.
  - Create a sequence for each use-case.
  - Build a state diagram for the system.
  - Review the behavioral model to verify accuracy and consistency.

The States of a System

- state—a set of observable circum-stances that characterizes the behavior of a system at a given time
- state transition—the movement from one state to another
- event—an occurrence that causes the system to exhibit some predictable form of behavior
- action—process that occurs as a consequence of making a transition

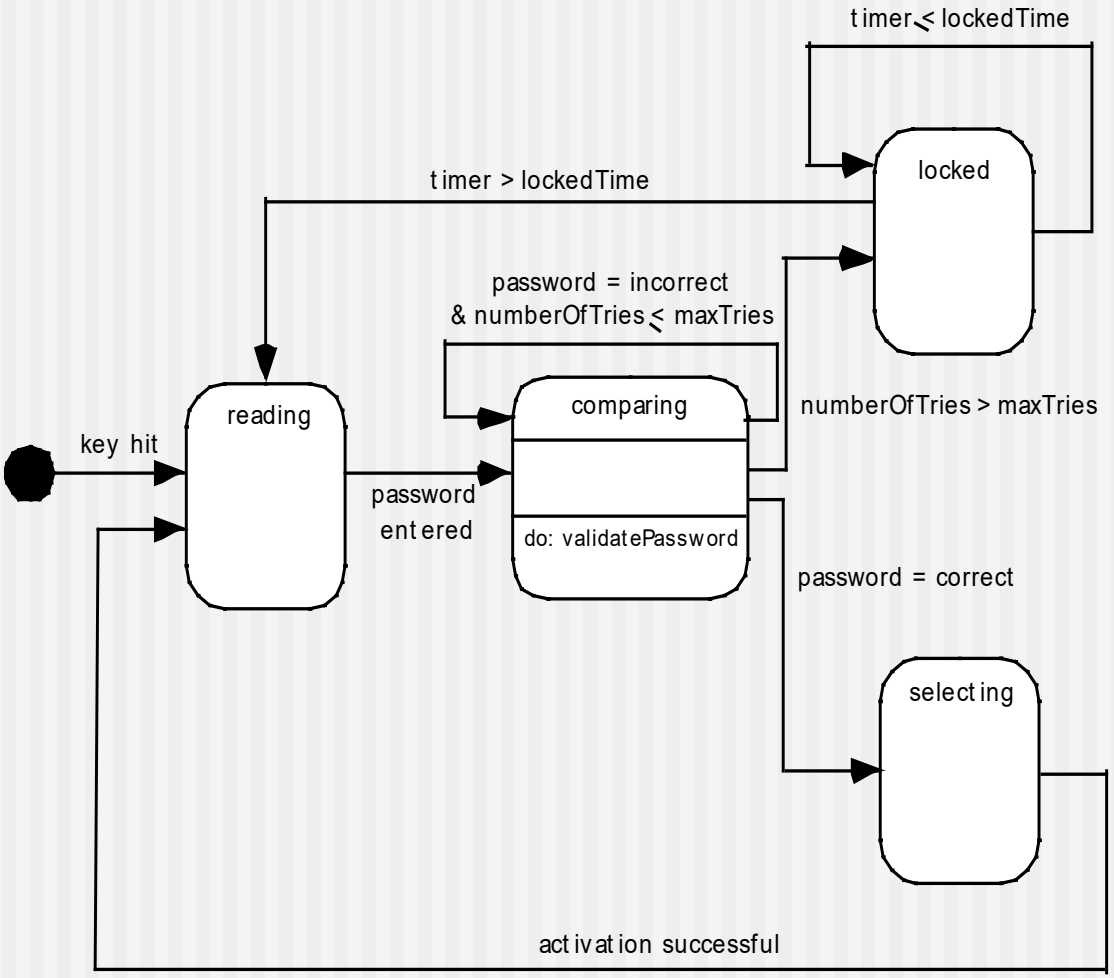
State Representations

- In the context of behavioral modeling, two different characterizations of states must be considered:
  - the state of each class as the system performs its function
  - the state of the system as observed from the outside as the system performs its function
- The state of a class takes on both passive and active characteristics [CHA93].
  - A *passive state* is simply the current status of all of an object's attributes.
  - The *active state* of an object indicates the current status of the object as it undergoes a continuing transformation or processing.

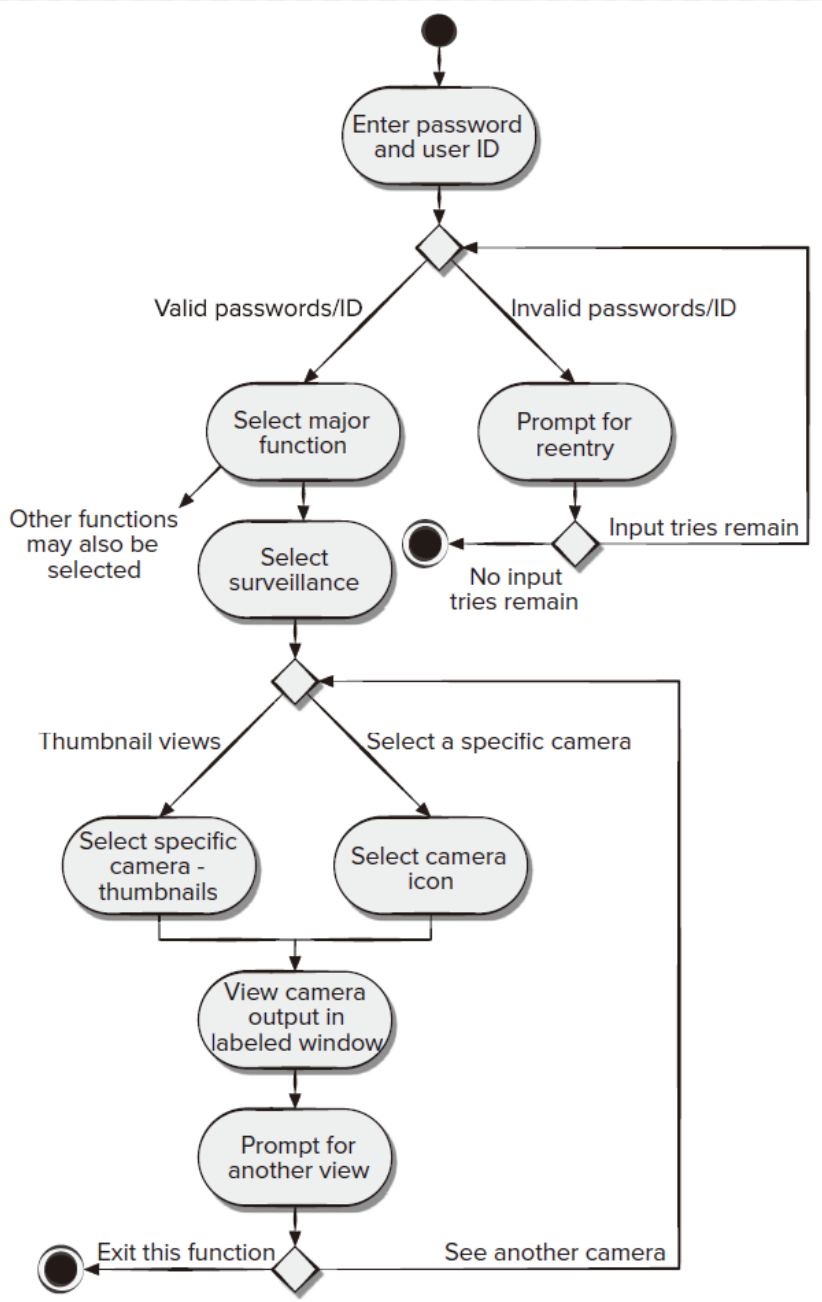
Behavioral Modeling

- make a list of the different states of a system (How does the system behave?)
- indicate how the system makes a transition from one state to another (How does the system change state?)
  - indicate event
  - indicate action
- draw a state diagram or a sequence diagram

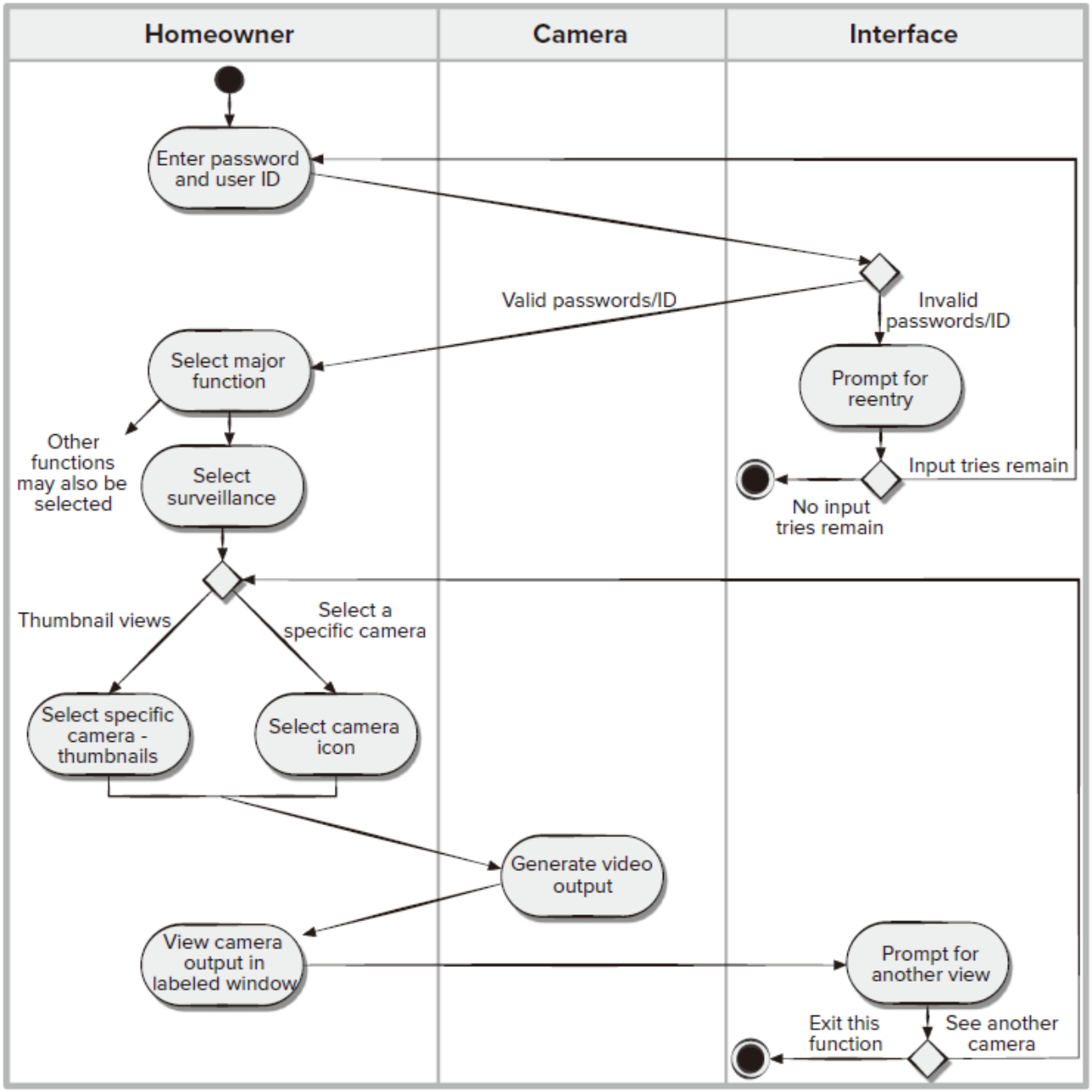
State Diagram for the ControlPanel Class



Activity Diagram



Swimlane Diagram



When Do We Perform Analysis?

- In some Web/Mobile App situations, analysis and design merge. However, an explicit analysis activity occurs when …
  - the Web or Mobile App to be built is large and/or complex
  - the number of stakeholders is large
  - the number of developers is large
  - the development team members have not worked together before
  - the success of the app will have a strong bearing on the success of the business

Requirements Modeling for WebApps

Content Analysis. The full spectrum of content to be provided by the WebApp is identified, including text, graphics and images, video, and audio data. Data modeling can be used to identify and describe each of the data objects.

Interaction Analysis. The manner in which the user interacts with the WebApp is described in detail. Use-cases can be developed to provide detailed descriptions of this interaction.

Functional Analysis. The usage scenarios (use-cases) created as part of interaction analysis define the operations that will be applied to WebApp content and imply other processing functions. All operations and functions are described in detail.

Configuration Analysis. The environment and infrastructure in which the WebApp resides are described in detail.

The Content Model

- Content objects are extracted from use-cases
  - examine the scenario description for direct and indirect references to content
- Attributes of each content object are identified
- The relationships among content objects and/or the hierarchy of content maintained by a WebApp
  - Relationships—entity-relationship diagram or UML
  - Hierarchy—data tree or UML

Data Tree

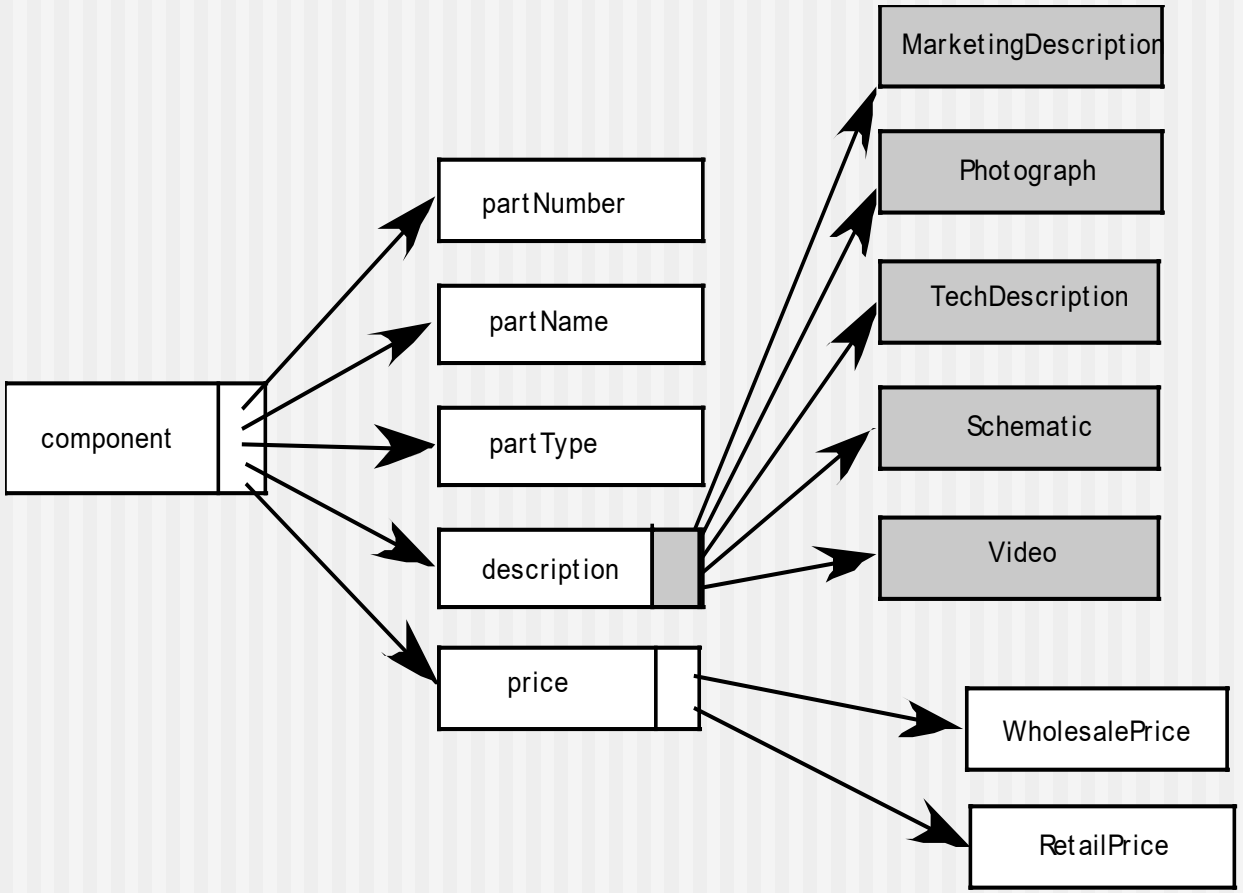
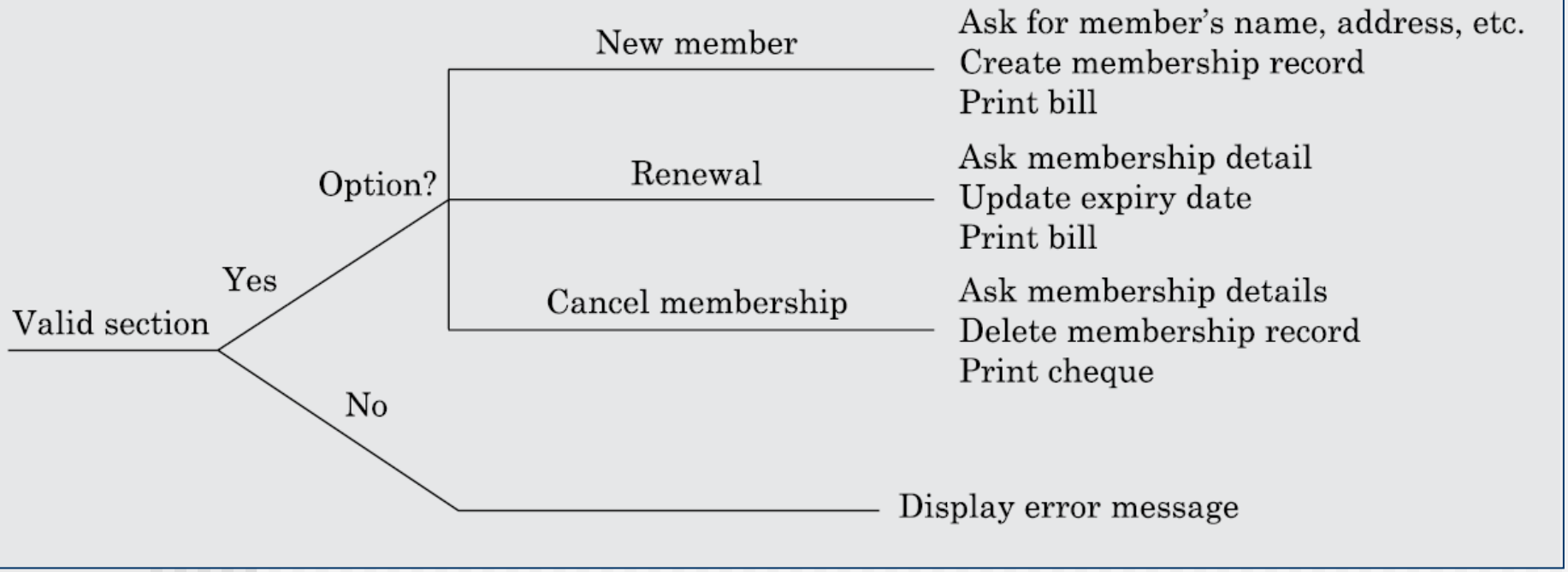


Figure 18.3 Data tree for a *SafeHome* component

Data Tree


Valid section
Yes
Option?
New member
Ask for member's name, address, etc.
Create membership record
Print bill
Renewal
Ask membership detail
Update expiry date
Print bill
Cancel membership
Ask membership details
Delete membership record
Print cheque
No
Display error message

Data Table

Conditions				
Valid selection	NO	YES	YES	YES
New member	-	YES	NO	NO
Renewal	-	NO	YES	NO
Cancellation	-	NO	NO	YES
Actions				
Display error message	×			
Ask member's name, etc.		×		
Build customer record		×		
Generate bill		×	×	
Ask membership details			×	×
Update expiry date			×	
Print cheque				×
Delete record				×

The Interaction Model

- Composed of four elements:
  - use-cases
  - sequence diagrams
  - state diagrams
  - a user interface prototype
- Each of these is an important UML notation.

Sequence Diagram

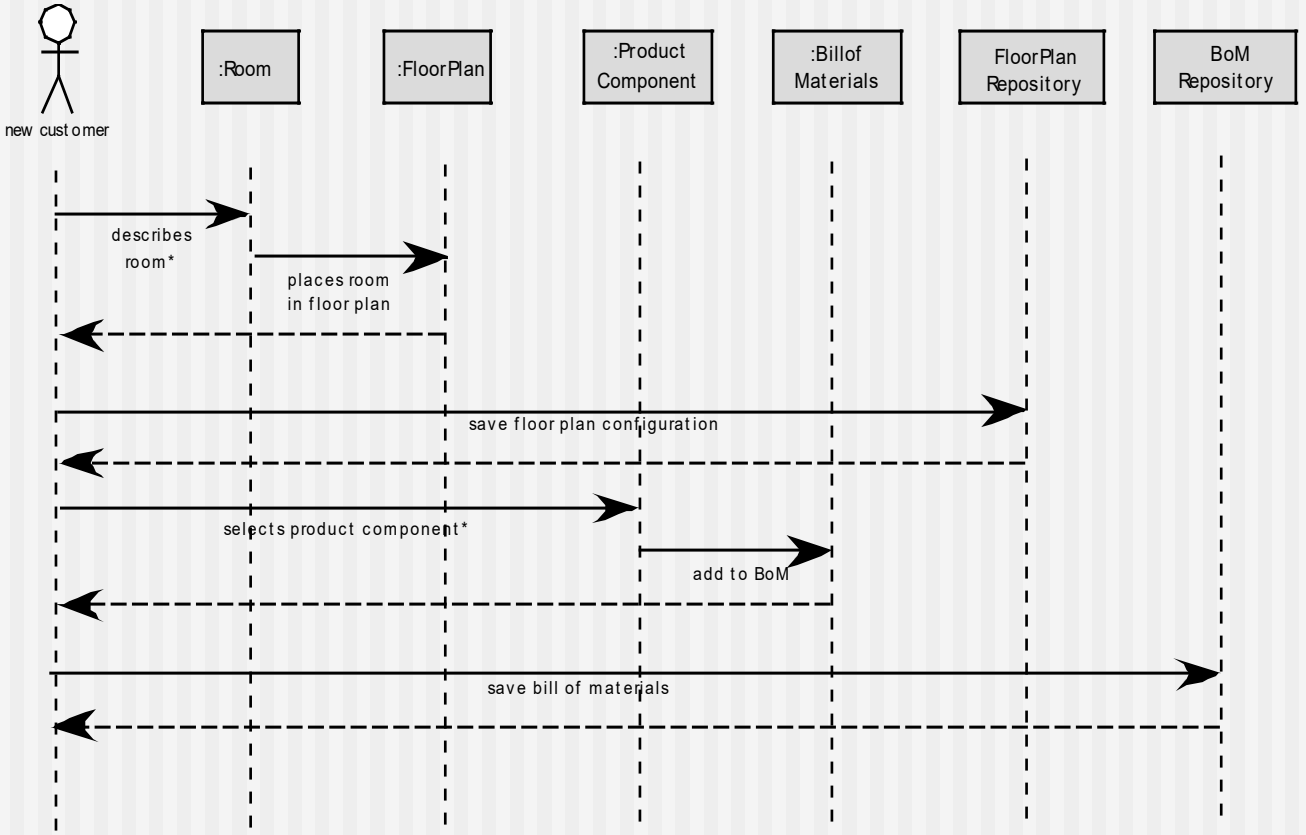


Figure 18.5 Sequence diagram for use-case: *select SafeHome components*

State Diagram

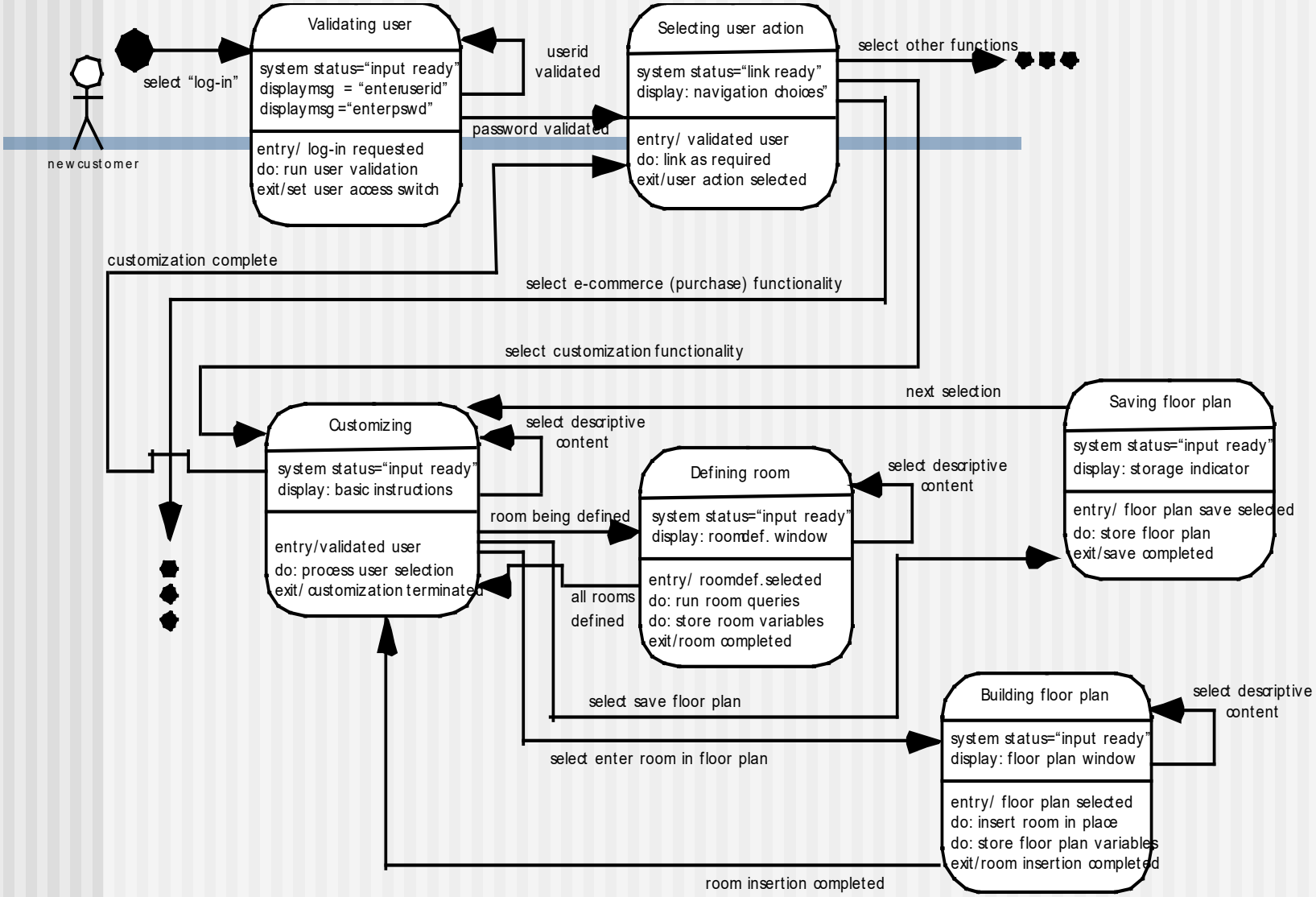


Figure 18.6 Partial state diagram for **new customer** interaction

The Functional Model

- The functional model addresses two processing elements of the WebApp
  - user observable functionality that is delivered by the WebApp to end-users
  - the operations contained within analysis classes that implement behaviors associated with the class.
- An activity diagram can be used to represent processing flow

Activity Diagram

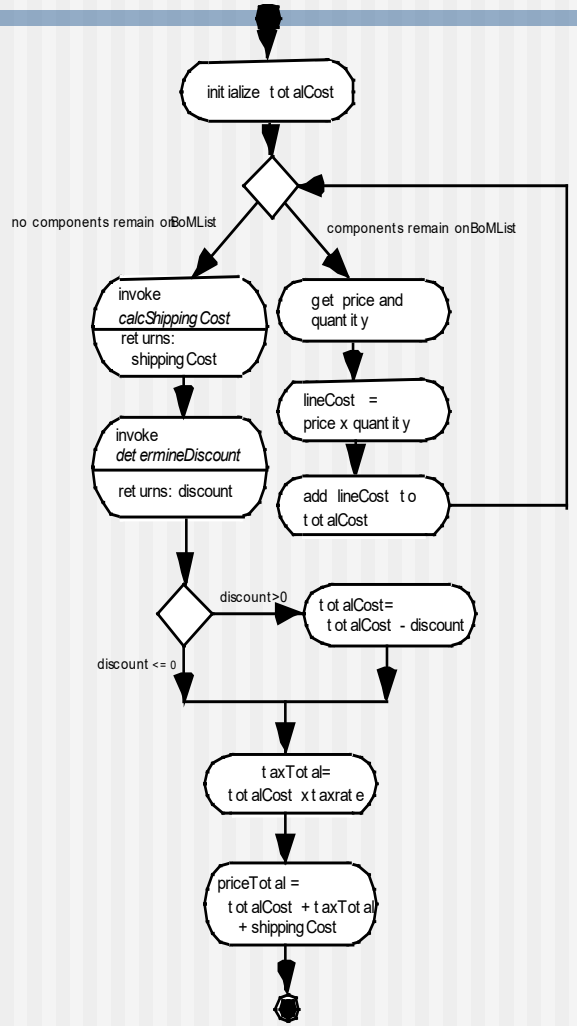
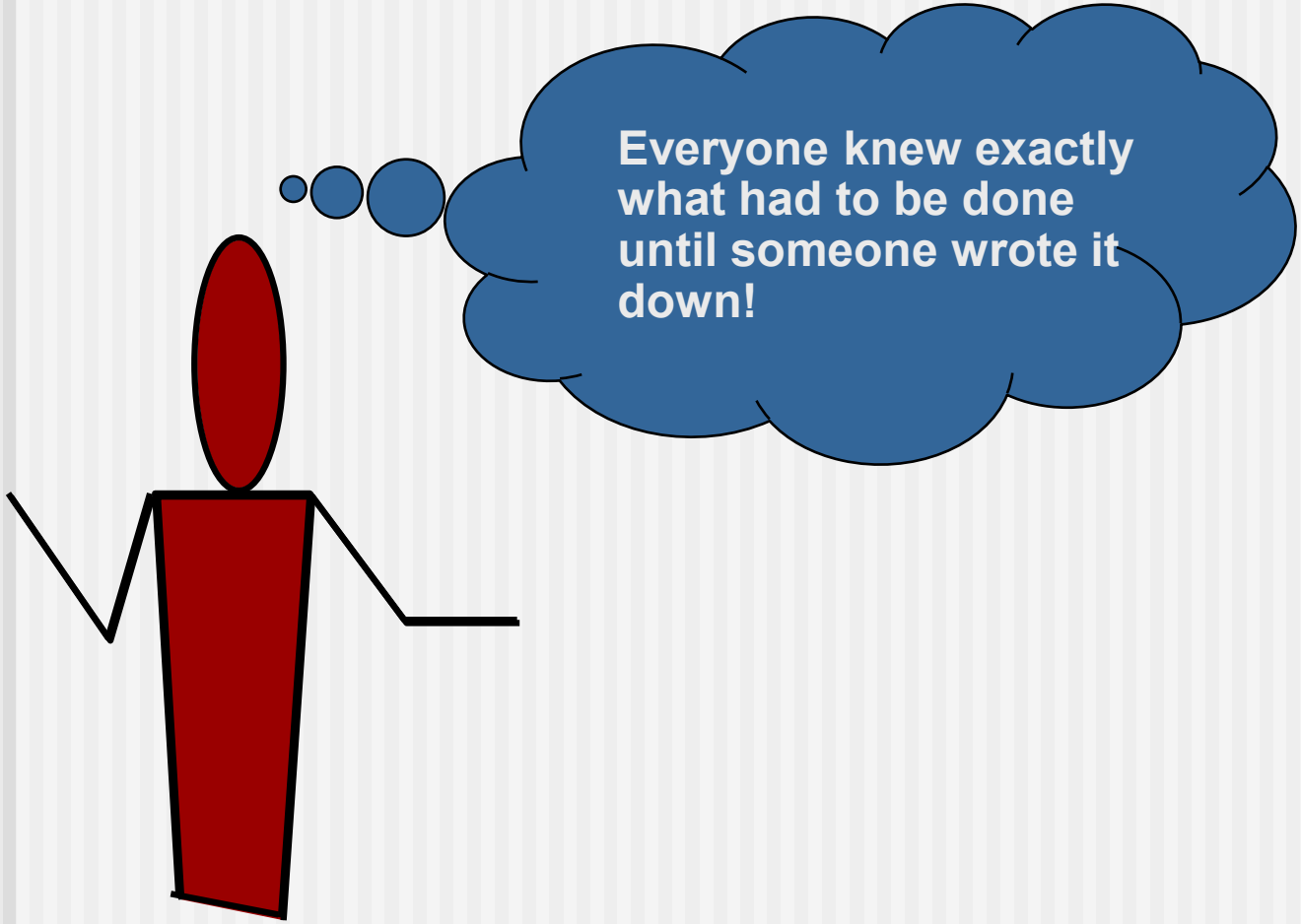


Figure 18.7 Activity diagram for ***computePrice()*** operatic

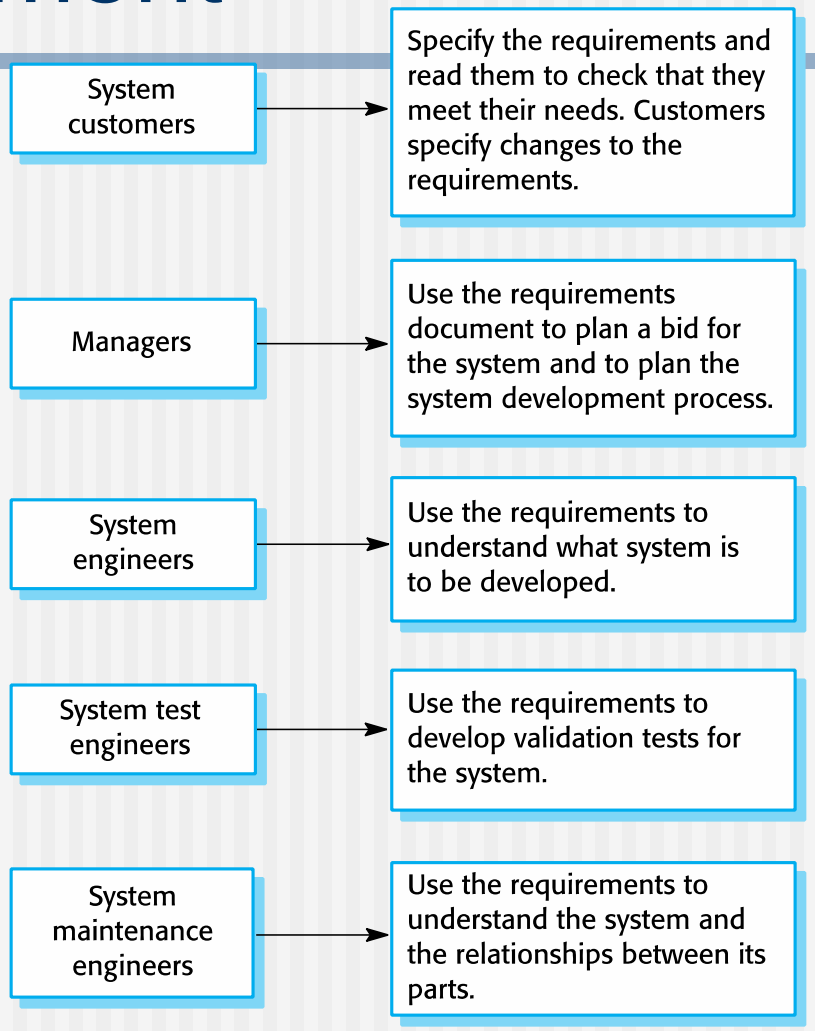
Writing the Software Specification



The software requirements document

- The software requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

Users of a requirements document



Requirements document variability

- Information in requirements document depends on type of system and the approach to development used.
- Systems developed incrementally will, typically, have less detail in the requirements document.
- Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

The structure of a requirements document

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

The structure of a requirements document

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements checking

- Validity. Does the system provide the functions which best support the customer's needs?
- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using an executable model of the system to check requirements. Covered in Chapter 2.
- Test-case generation
  - Developing tests for requirements to check testability.

Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

Review checks

- Verifiability
  - Is the requirement realistically testable?
- Comprehensibility
  - Is the requirement properly understood?
- Traceability
  - Is the origin of the requirement clearly stated?
- Adaptability
  - Can the requirement be changed without a large impact on other requirements?

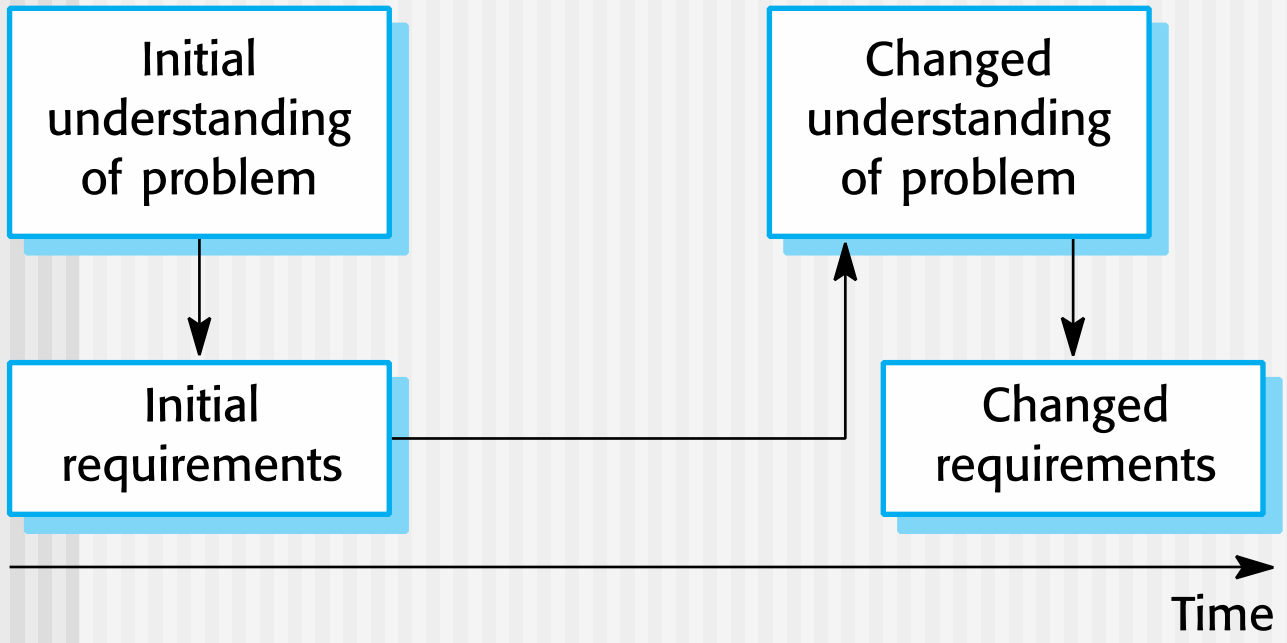
Changing requirements

- The business and technical environment of the system always changes after installation.
  - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.
- The people who pay for a system and the users of that system are rarely the same people.
  - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

Changing requirements

- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.
  - The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

Requirements evolution



Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

Requirements management planning

- Establishes the level of requirements management detail that is required.
- Requirements management decisions:
  - *Requirements identification* Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
  - *A change management process* This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following section.

Requirements management planning

- Requirements management decisions:
  - *Traceability policies* These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
  - *Tool support* Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

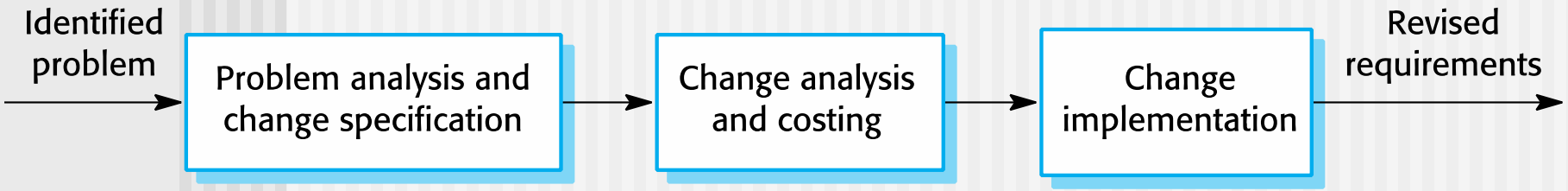
Requirements change management

- Deciding if a requirements change should be accepted
  - *Problem analysis and change specification*
    - During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
  - *Change analysis and costing*
    - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.

Requirements change management

- Deciding if a requirements change should be accepted
  - Change implementation
    - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.

Requirements change management



Summary

- The objective of requirements modeling is to create a variety of representations that describe what the customer requires, establish a basis for the creation of a software design, and define a set of requirements that can be validated once the software is built. The requirements model bridges the gap between a system-level description that describes overall system and business functionality and a software design that describes the software's application architecture, user interface, and component-level structure.

Summary

- Scenario-based models depict software requirements from the user's point of view. The use case — a narrative or template-driven description of an interaction between an actor and the software — is the primary modeling element. Derived during requirements elicitation, the use case defines the key steps for a specific function or interaction. The degree of use case formality and detail varies, but they can provide necessary input to all other analysis modeling activities. Scenarios can also be described using an activity diagram—a graphical representation that depicts the processing flow within a specific scenario. Temporal relations in a use case can be modeled using sequence diagrams.

Summary

- Class-based modeling uses information derived from use cases and other written application descriptions to identify analysis classes. A grammatical parse may be used to extract candidate classes, attributes, and operations from text-based narratives. Criteria for the definition of a class are defined using the parse results.
- A set of class-responsibility-collaborator index cards can be used to define relation-ships between classes. In addition, a variety of UML modeling notation can be applied to define hierarchies, relationships, associations, aggregations, and dependencies among classes.

Summary

- Behavioral modeling during requirements analysis depicts dynamic behavior of the software. The behavioral model uses input from scenario-based or class-based elements to represent the states of analysis classes. To accomplish this, states are identified, the events that cause a class (or the system) to make a transition from one state to another are defined, and the actions that occur as transition is accomplished are also identified. UML state diagrams, activity diagrams, swim lane diagrams, and sequence diagrams can be used for behavioral modeling.