Software Engineering: A Practitioner's Approach, 6/e

Chapter 7
Requirements Engineering

Requirements Engineering-I

- 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
- Elaboration—create an analysis model that identifies data, function and behavioral requirements
- Negotiation—agree on a deliverable system that is realistic for developers and customers

Requirements Engineering-II

- Specification—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—a review mechanism that looks for
  - errors in content or interpretation
  - areas where clarification may be required
  - missing information
  - inconsistencies (a major problem when large products or systems are engineered)
  - conflicting or unrealistic (unachievable) requirements.
- Requirements management

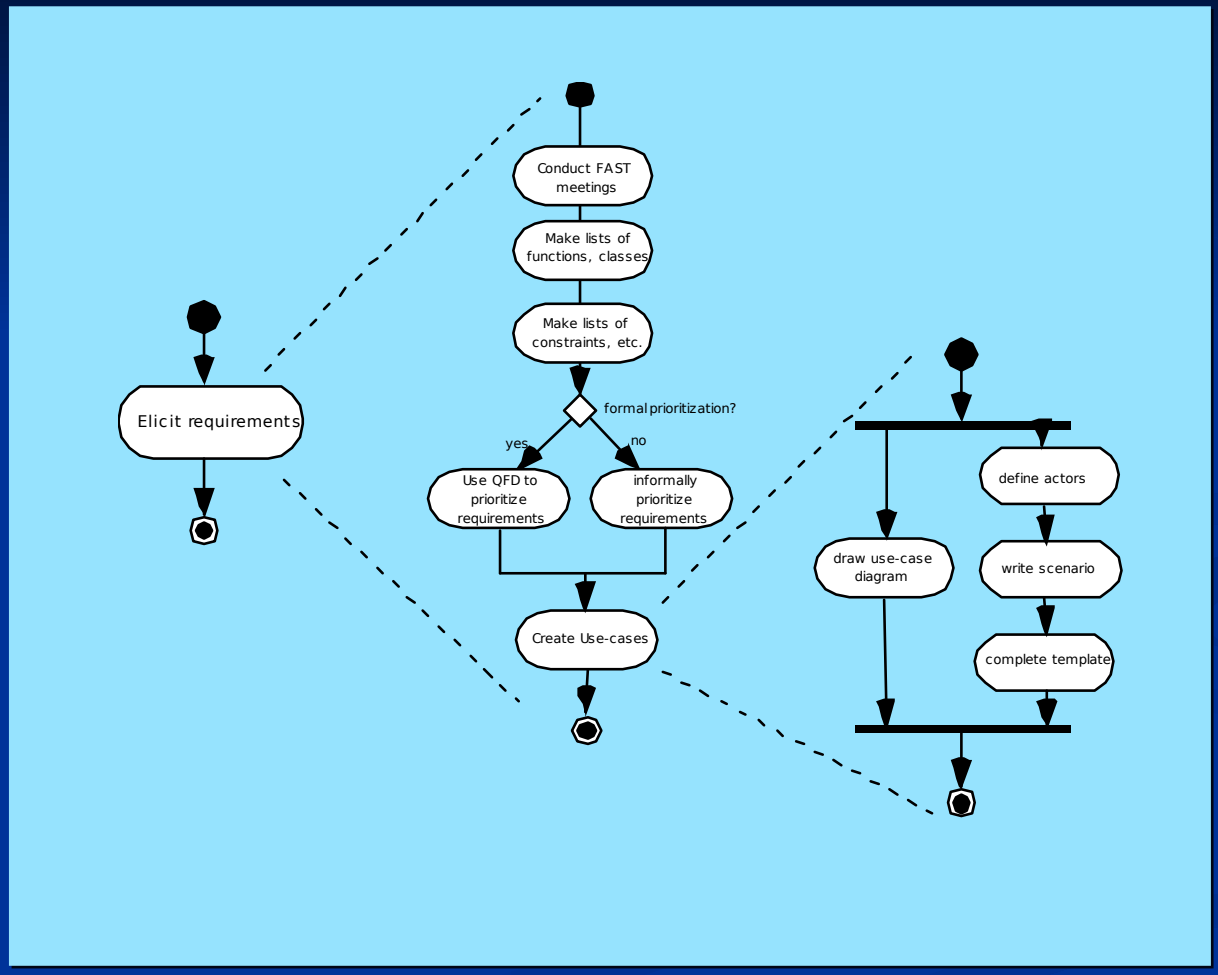
Inception

- Identify stakeholders
  - "who else do you think I should talk to?"
- Recognize multiple points of view
- Work toward collaboration
- 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?

Eliciting Requirements

- meetings are conducted and attended by both software engineers and customers
- rules for preparation and participation are established
- an agenda is suggested
- a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- a "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
- the goal is
  - to identify the problem
  - propose elements of the solution
  - negotiate different approaches, and
  - specify a preliminary set of solution requirements

Eliciting Requirements



Quality Function Deployment

- Function deployment determines the "value" (as perceived by the customer) of each function required of the system
- Information deployment identifies data objects and events
- Task deployment examines the behavior of the system
- Value analysis determines the relative priority of requirements

Quality Function Deployment

- QFD : emphasizes an understanding of what is valuable to customer & then deploys these values throughout the engineering process.

3 types of requirements

- Normal requirements
- Expected requirements
- Exciting requirements

QFD

- QFD techniques applicable to requirements elicitation.
- QFD uses customer interviews , surveys & examination of historical data for requirement gathering activity.
- Data is then translated into a table of requirements called customer ***Voice table***.
- Voice table is reviewed by customer & stakeholders.

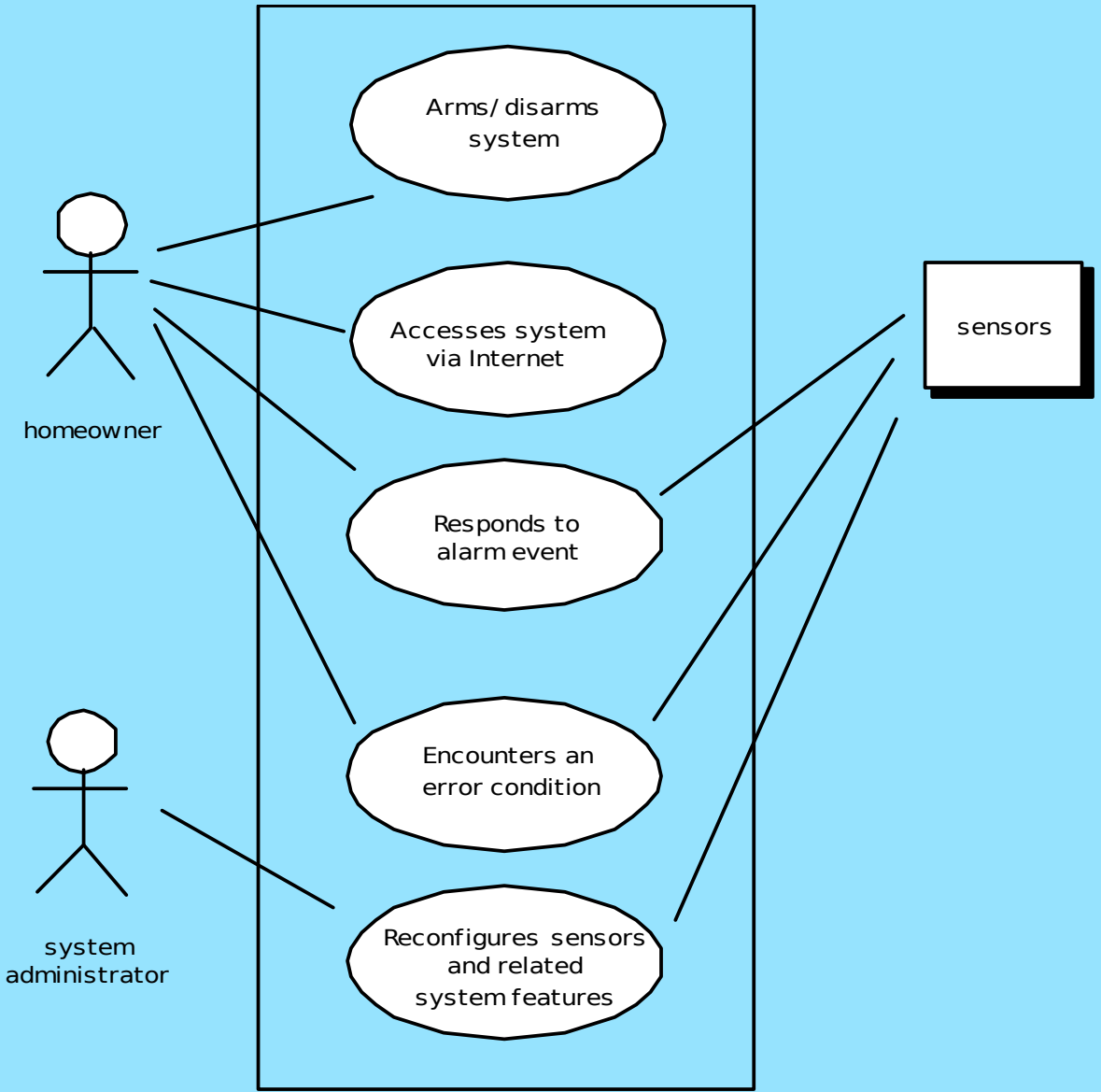
Elicitation Work Products

- 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.

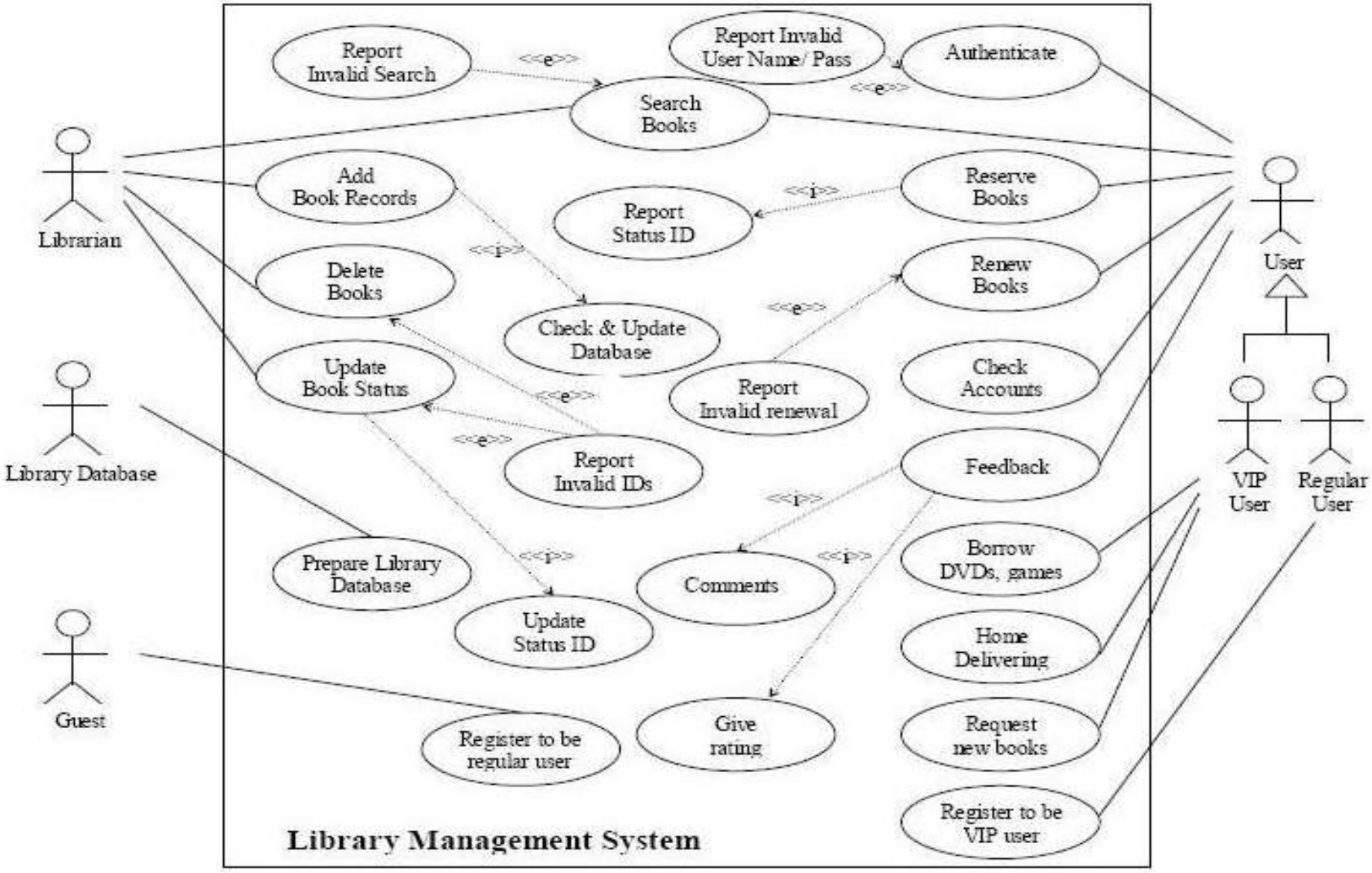
Use-Cases

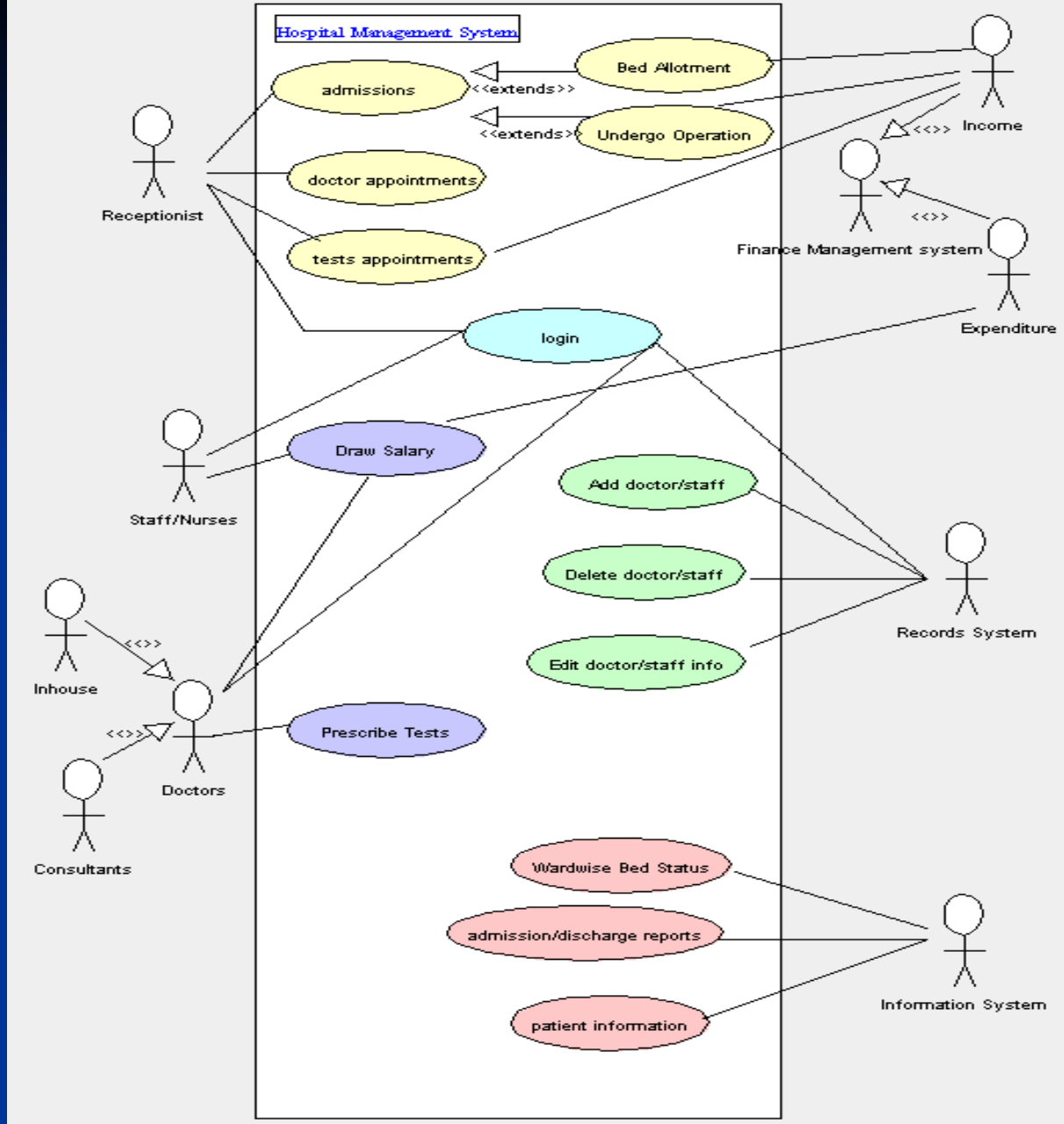
- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an "actor"—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
  - **Who is the primary actor, the secondary 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 extensions 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?**

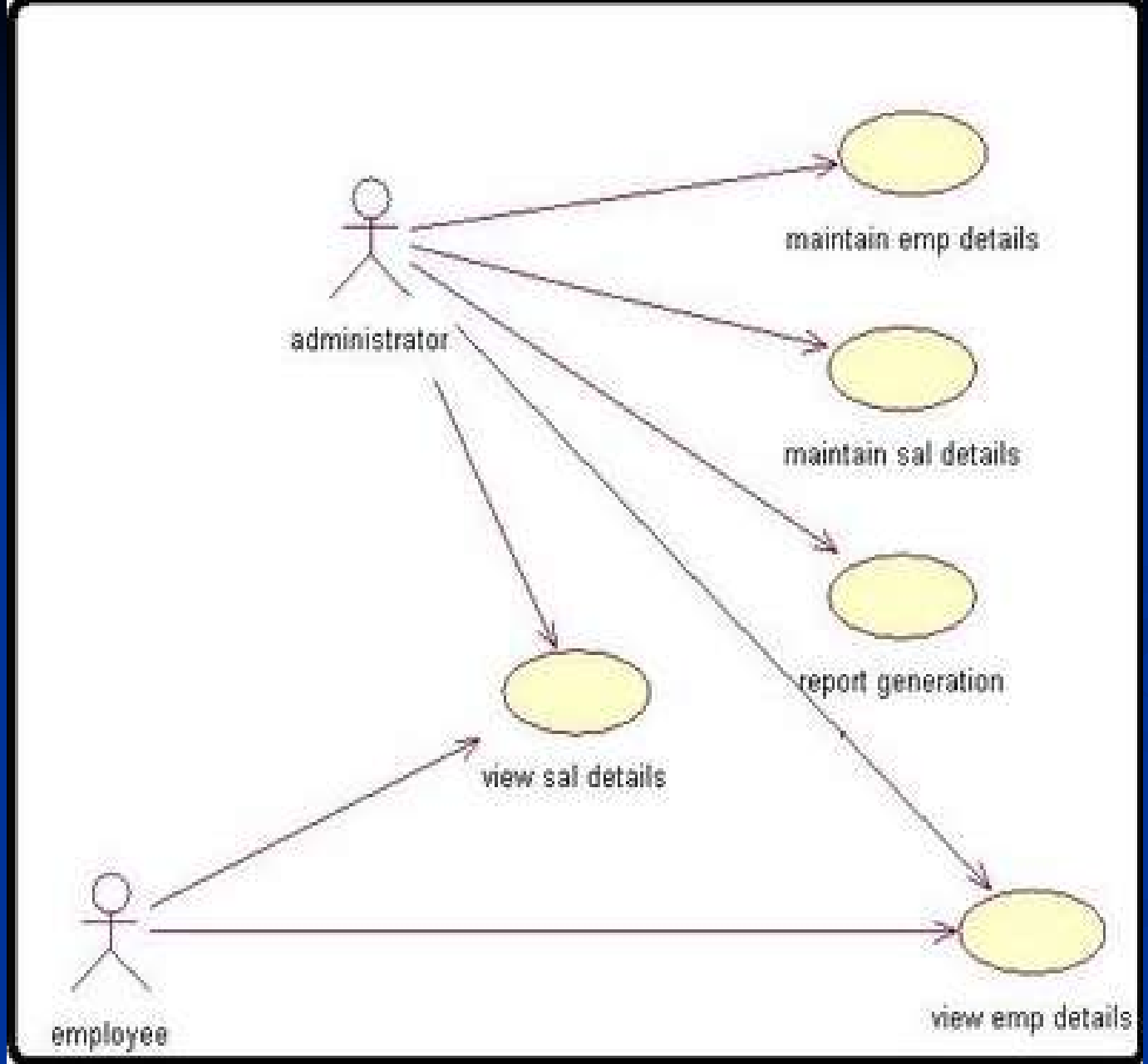
Use-Case Diagram



These
are pro


Report Invalid Search
<<e>>
Report Invalid User Name/ Pass
Authenticate
<<e>>
Search Books
Librarian
Add Book Records
Report Status ID
<<i>>
Reserve Books
<<i>>
Delete Books
Renew Books
<<e>>
Check & Update Database
User
Update Book Status
Report Invalid renewal
Check Accounts
<<e>>
<<e>>
Report Invalid IDs
Feedback
Library Database
VIP User
Regular User
<<i>>
Prepare Library Database
<<i>>
Comments
<<i>>
Borrow DVDs, games
Update Status ID
Home Delivering
Guest
Give rating
Request new books
Register to be regular user
Register to be VIP user
Library Management System


Hospital Management System
admissions
Bed Allotment
<<extends>>
<<extends>>
Undergo Operation
doctor appointments
tests appointments
Receptionist
Income
<<>>
Finance Management system
<<>>
Expenditure
login
Draw Salary
Staff/Nurses
Add doctor/staff
Delete doctor/staff
Edit doctor/staff info
Records System
Inhouse
<<>>
Prescribe Tests
<<>>
Doctors
Consultants
Wardwise Bed Status
admission/discharge reports
patient information
Information System


maintain emp details
administrator
maintain sal details
report generation
view sal details
employee
view emp details

Building the Requirement Model

- Elements of the requirement model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an "actor" and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram

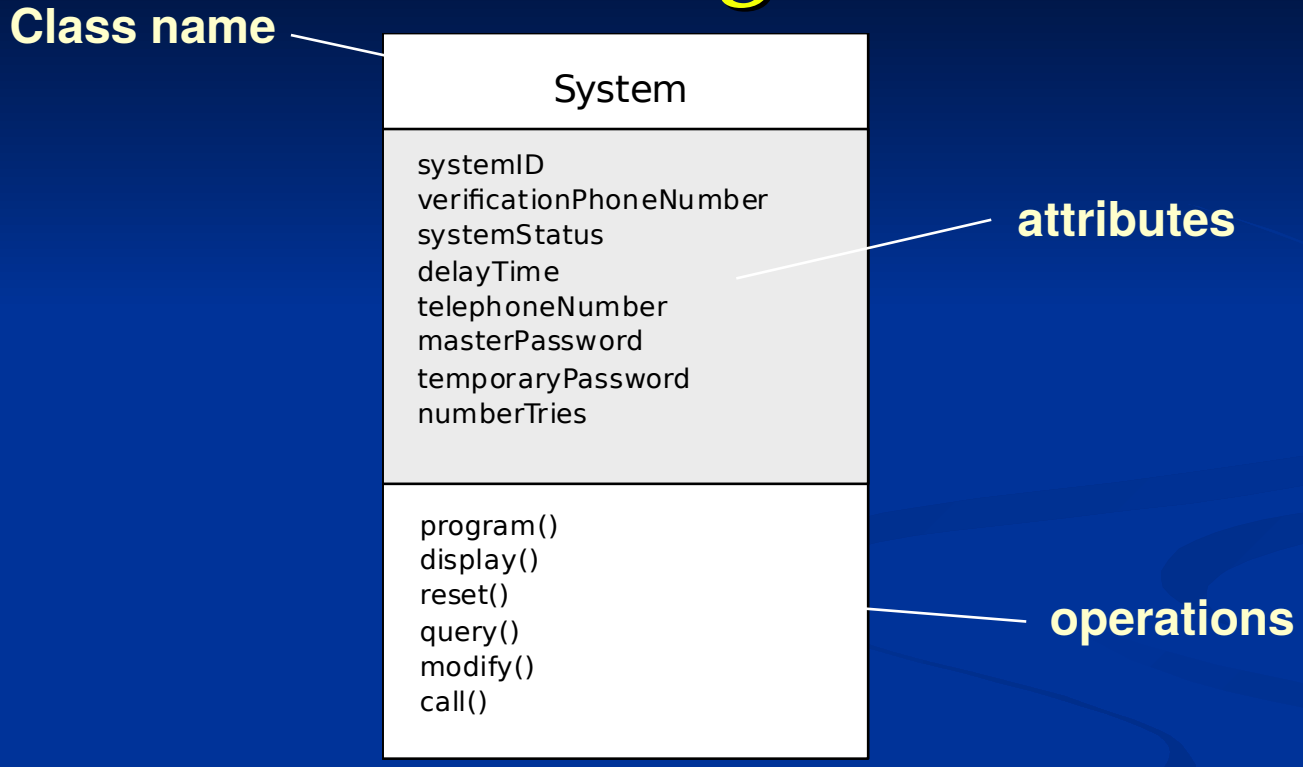
Analysis Classes

- ***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.

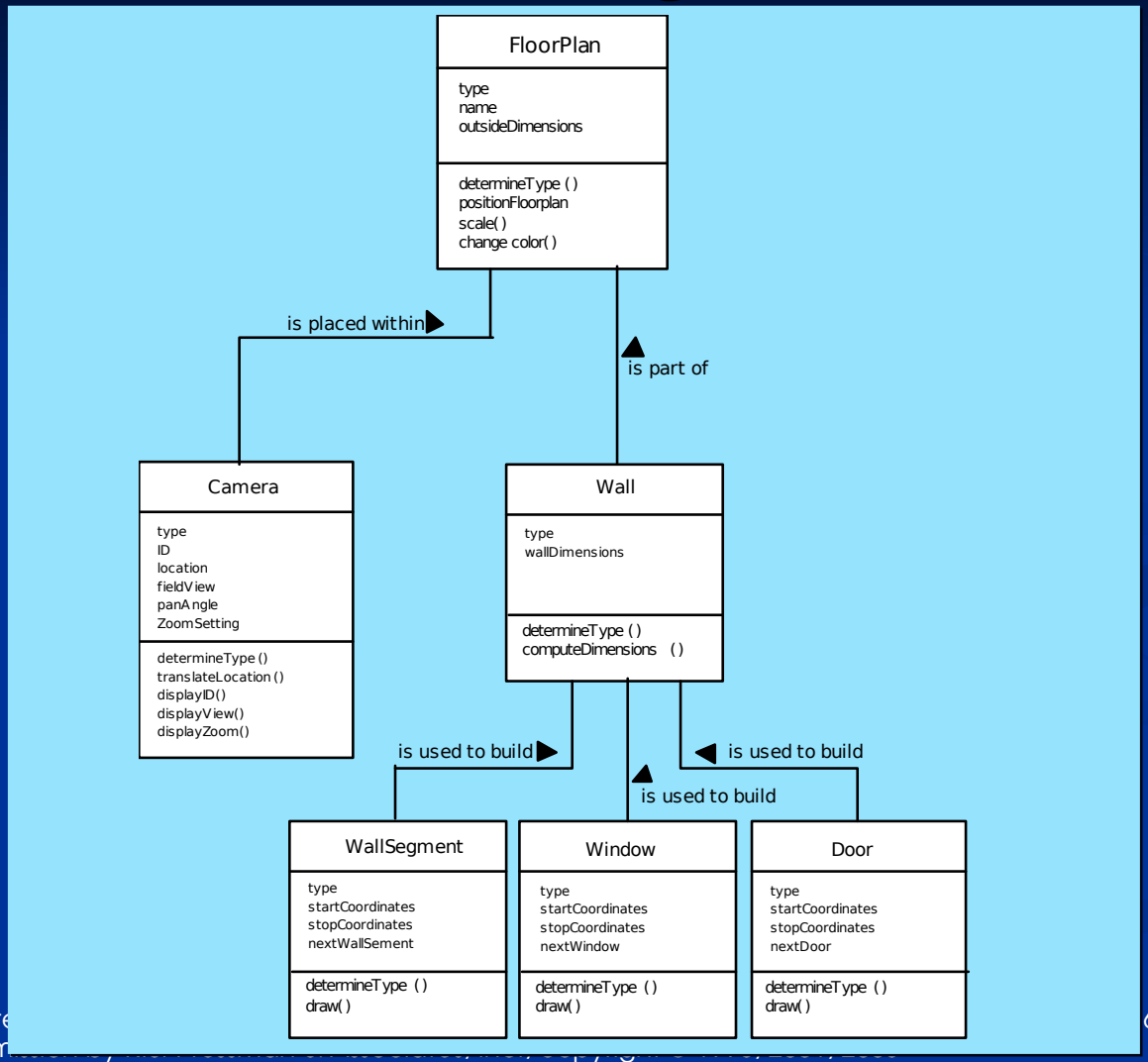
Selecting Classes—Criteria

- Retained information
- Needed services
- Multiple attributes
- Common attributes
- Common operations
- Essential requirements

Class Diagram



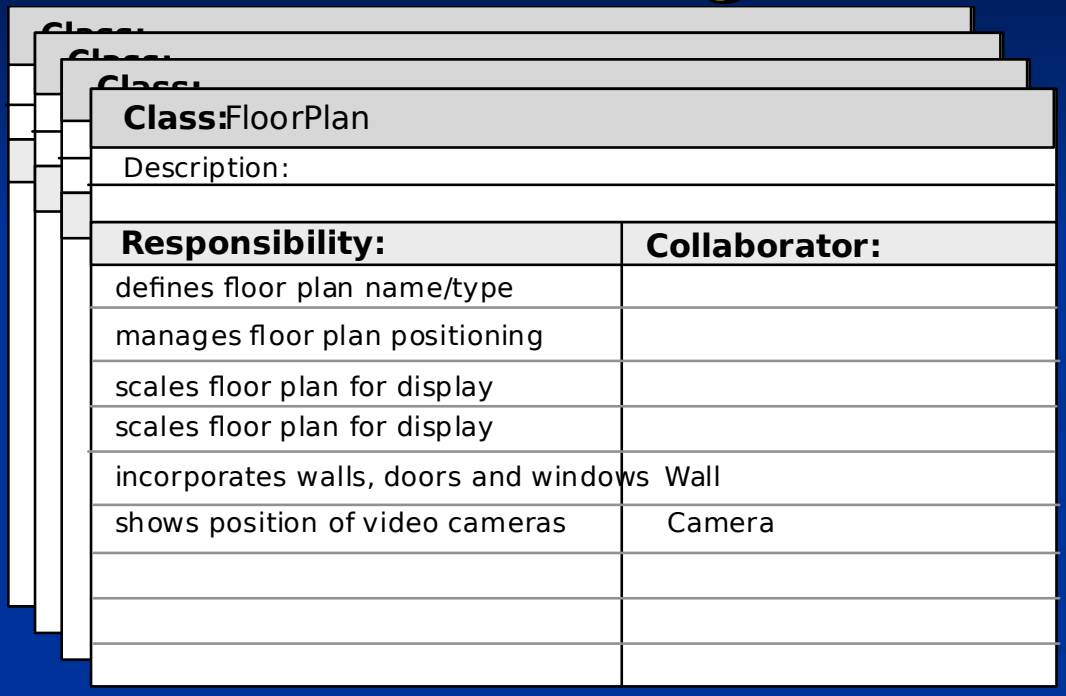
Class Diagram



CRC Modeling
(Class_Responsibility-Collaborator)

- Analysis classes have "responsibilities"
  - *Responsibilities* are the attributes and operations encapsulated by the class
- Analysis classes collaborate with one another
  - *Collaborators* are those classes that are required to provide a class with the information needed to complete a responsibility.
  - In general, a collaboration implies either a request for information or a request for some action.

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

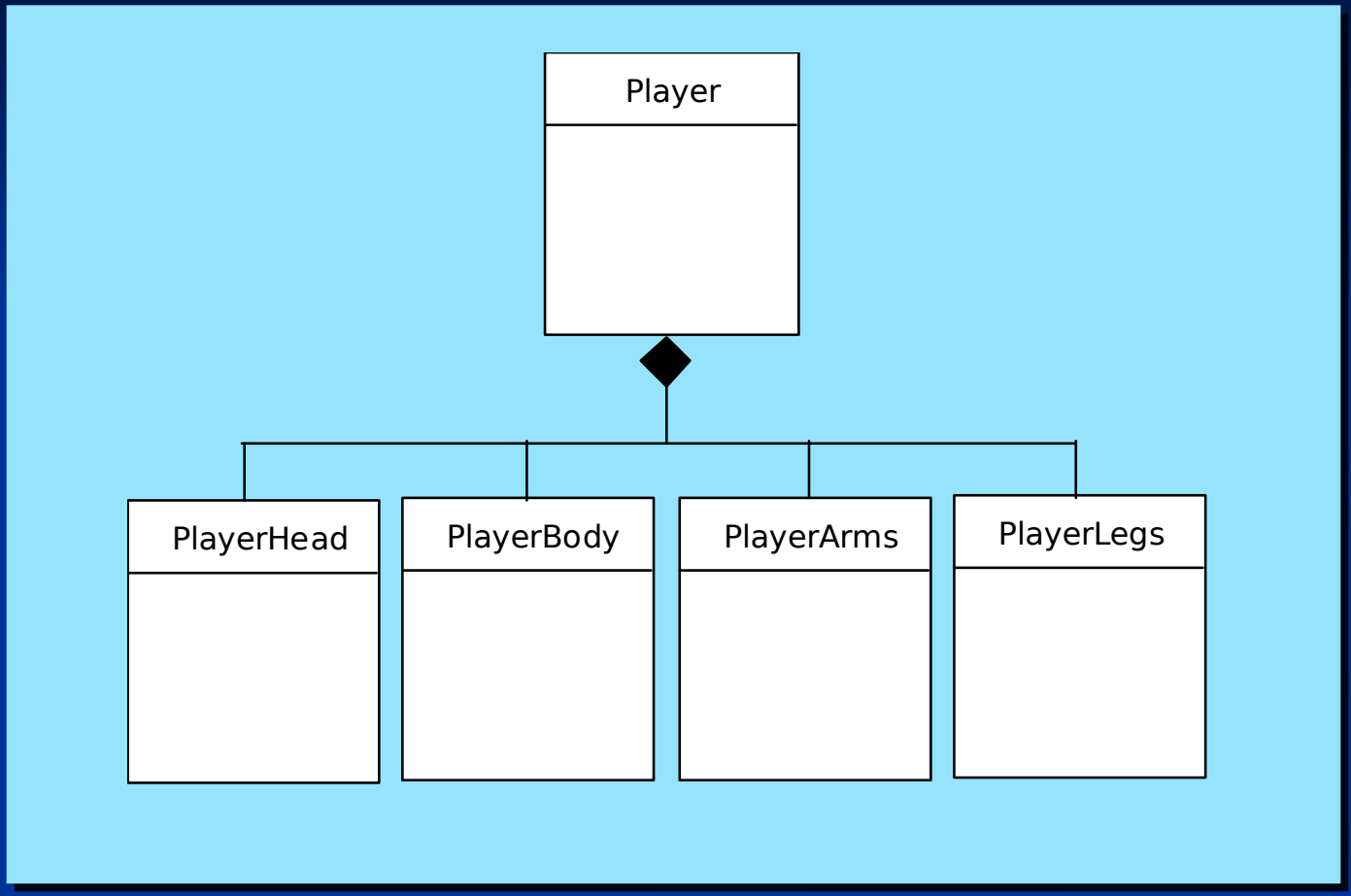
(Attributes and methods relevant to the class)

- 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

Composite Aggregate Class



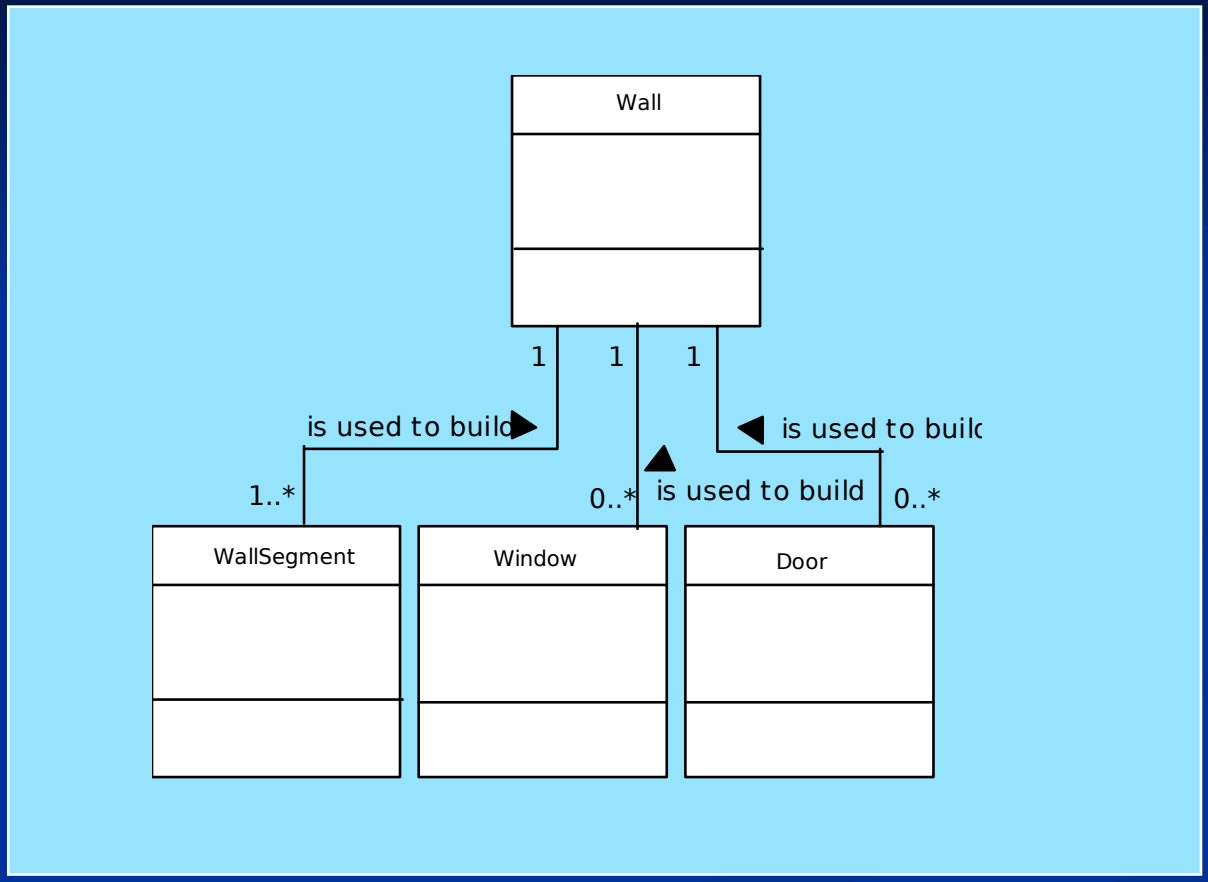
Reviewing the CRC Model

- 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.
- 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.

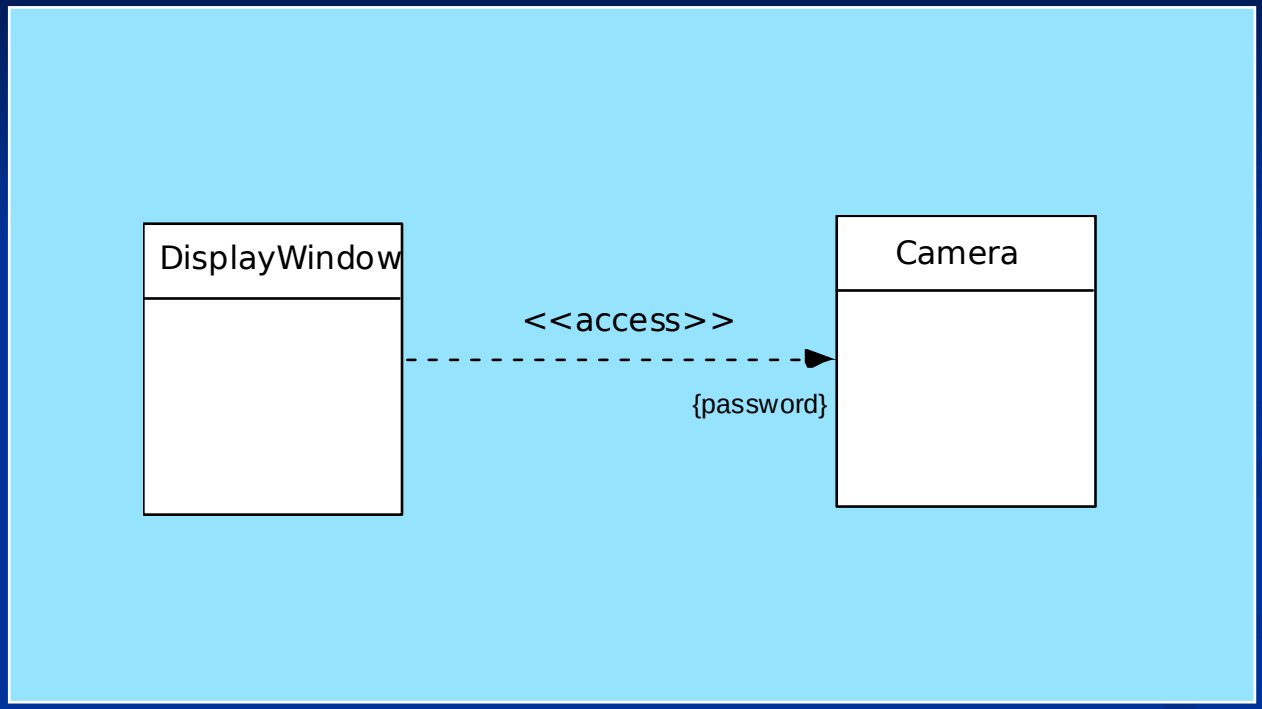
Associations and Dependencies

- Two analysis classes are often related to one another in some fashion
  - In UML these relationships are called *associations*
  - Associations can be refined by indicating *multiplicity* (the term *cardinality* is used in data modeling
- In many instances, a client-server relationship exists between two analysis classes.
  - In such cases, a client-class depends on the server-class in some way and a *dependency relationship* is established

Multiplicity



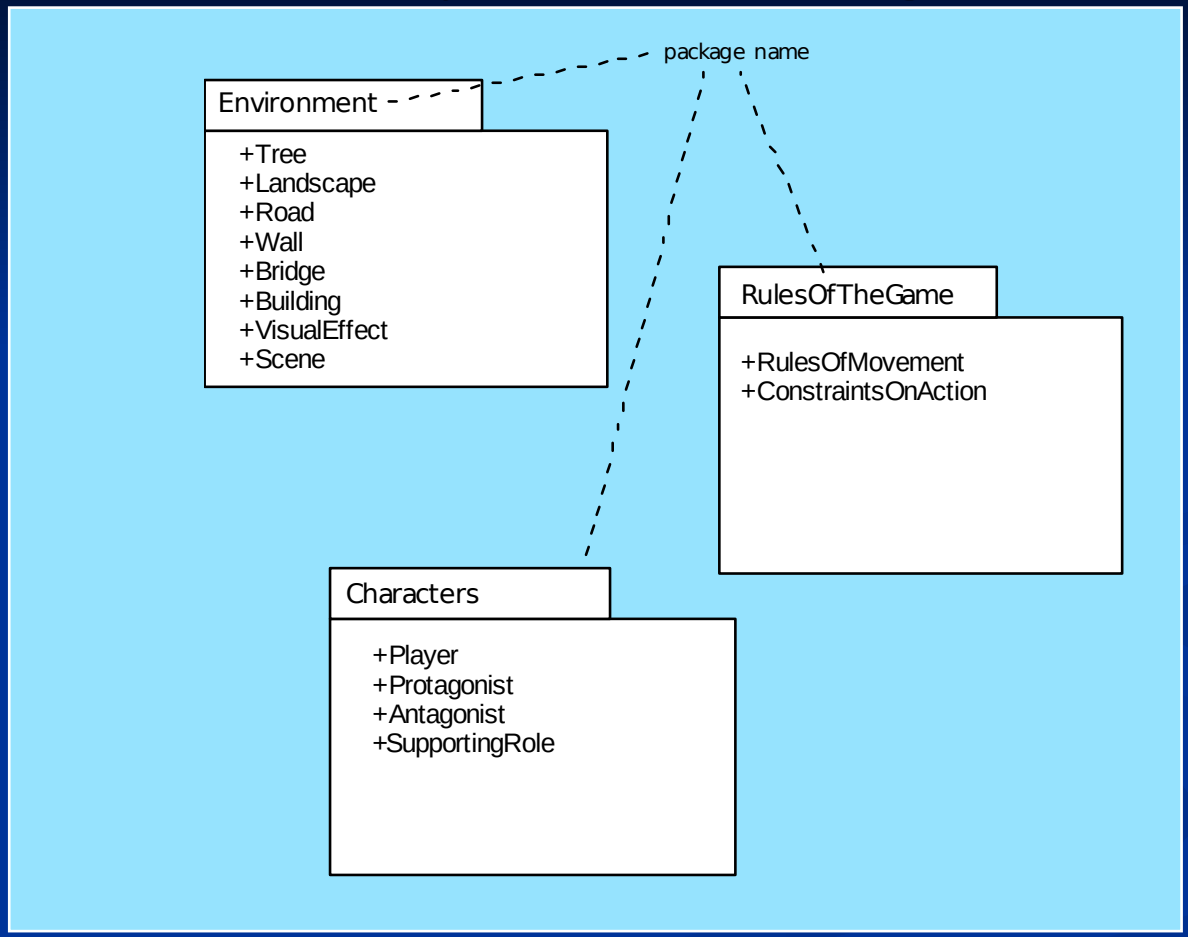
Dependencies



Analysis Packages

- Various elements of the analysis model (e.g., use-cases, analysis classes) are categorized in a manner that packages them as a grouping
- The plus sign preceding the analysis class name in each package indicates that the classes have public visibility and are therefore accessible from other packages.
- Other symbols can precede an element within a package. A minus sign indicates that an element is hidden from all other packages and a # symbol indicates that an element is accessible only to packages contained within a given package.

Analysis Packages



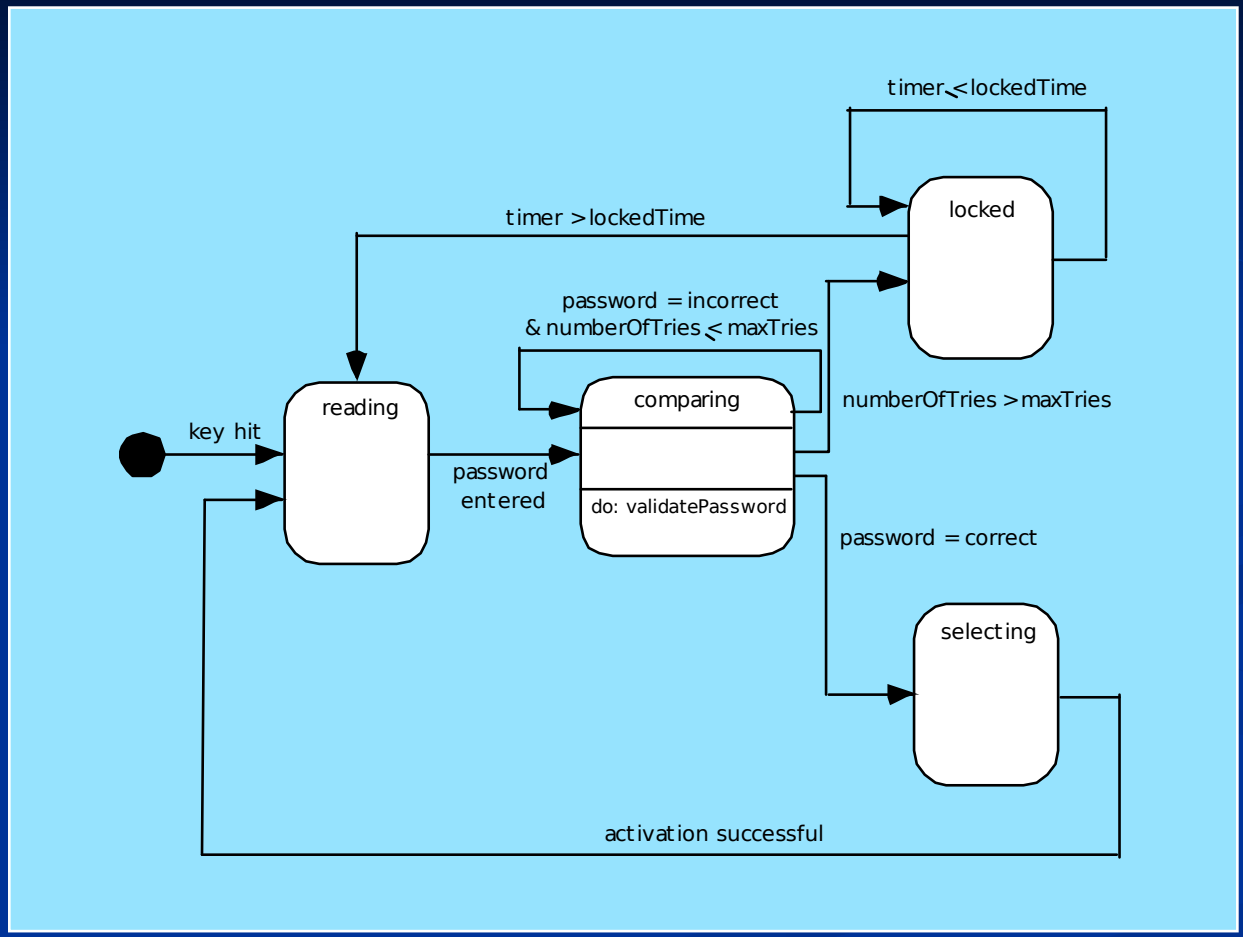
Behavioral Modeling

- 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.

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 and
  - 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.

State Diagram for the ControlPanel Class



The States of a System

- state—a set of observables 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

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

Sequence Diagram

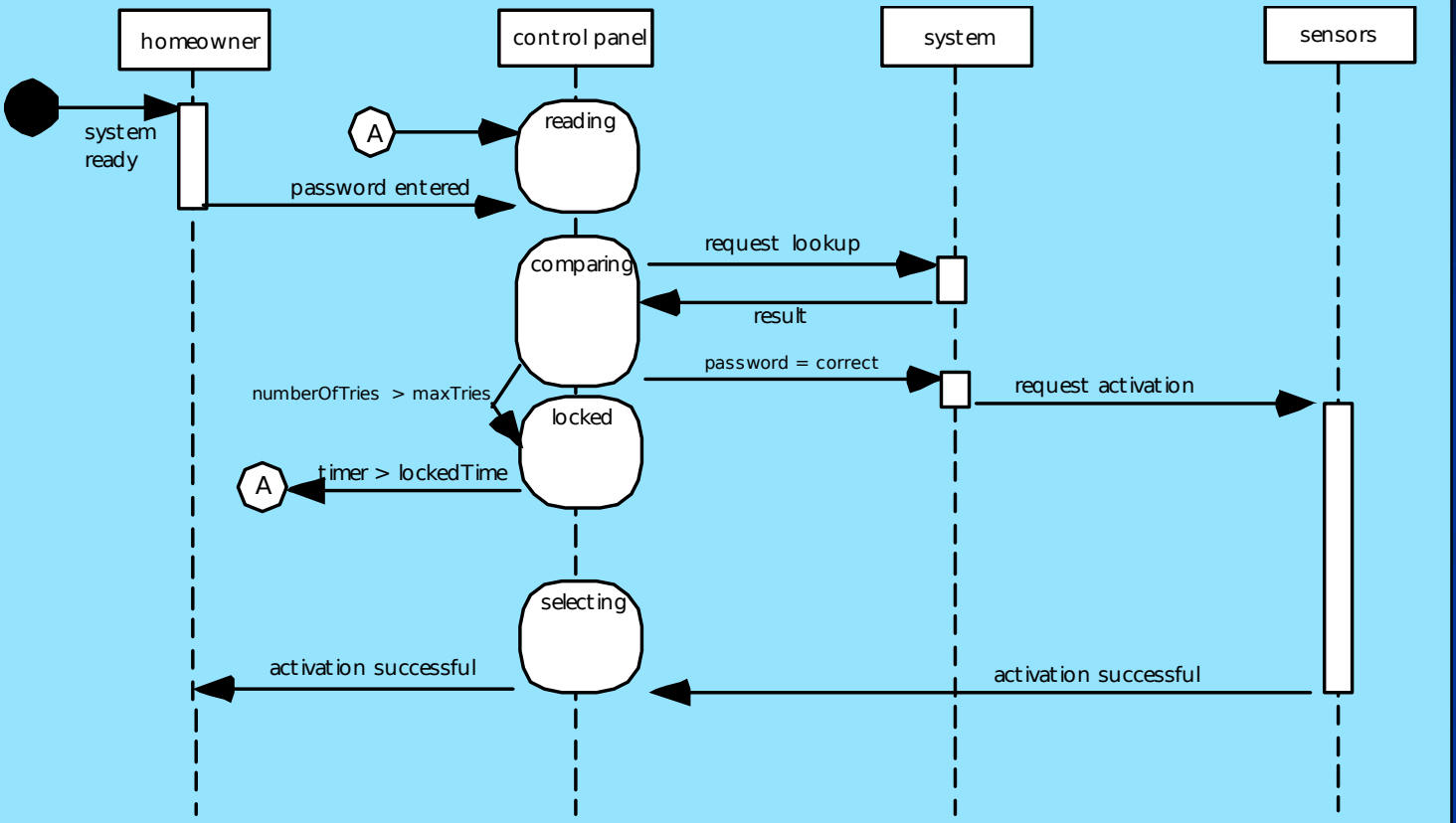
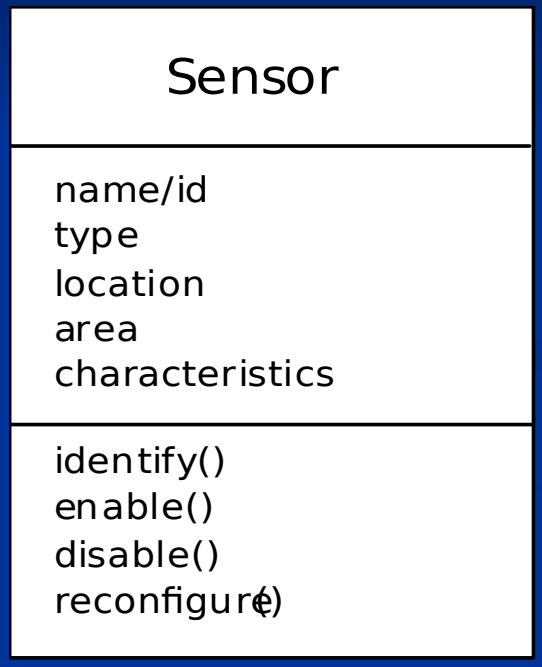


Figure 8.27 Sequence diagram (partial) for *SafeHome* security function

Writing the Software Specification

Class Diagram

From the *SafeHome* system ...



State Diagram

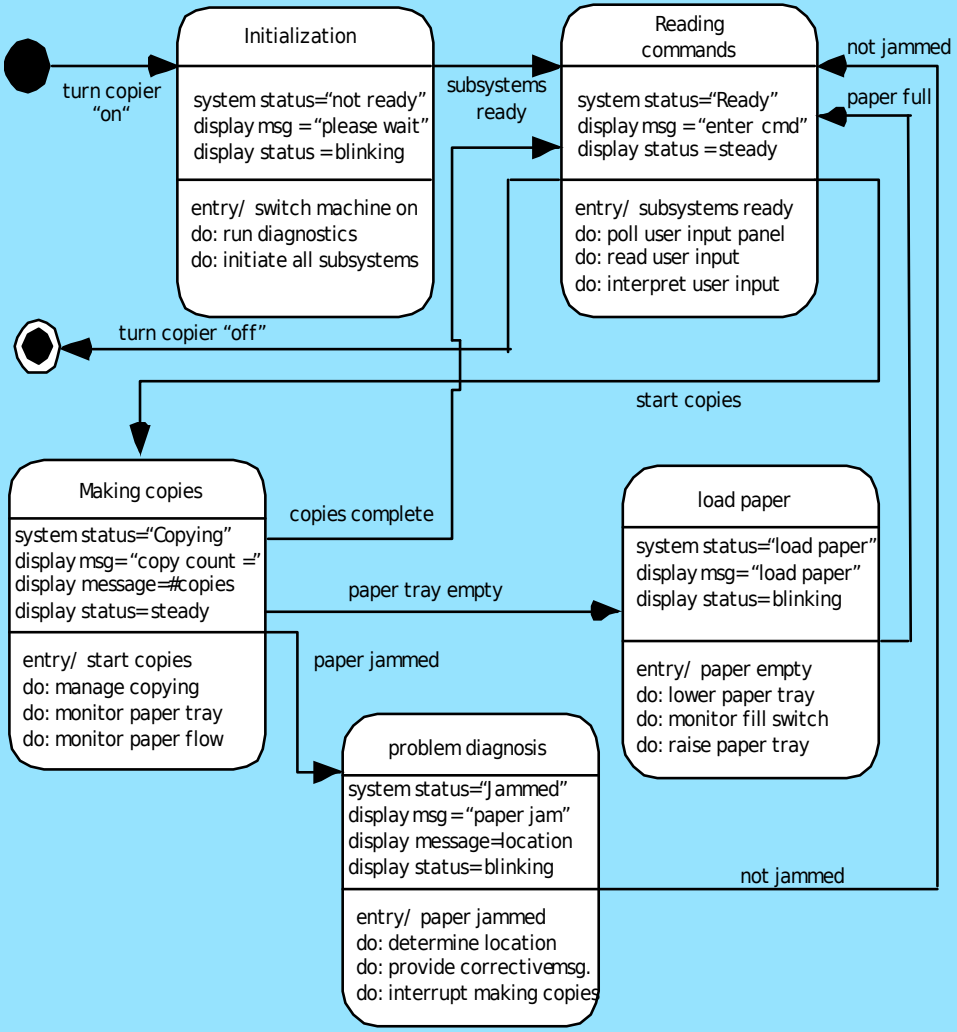


Figure 7.6 Preliminary UML state diagram for a photocopier

Analysis Patterns

Pattern name: A descriptor that captures the essence of the pattern.

Intent: Describes what the pattern accomplishes or represents

Motivation: A scenario that illustrates how the pattern can be used to address the problem.

Forces and context: A description of external issues (forces) that can affect how the pattern is used and also the external issues that will be resolved when the pattern is applied.

Solution: A description of how the pattern is applied to solve the problem with an emphasis on structural and behavioral issues.

Consequences: Addresses what happens when the pattern is applied and what trade-offs exist during its application.

Design: Discusses how the analysis pattern can be achieved through the use of known design patterns.

Known uses: Examples of uses within actual systems.

Related patterns: On e or more analysis patterns that are related to the named pattern because (1) it is commonly used with the named pattern; (2) it is structurally similar to the named pattern; (3) it is a variation of the named pattern.

Negotiating Requirements

- Identify the key stakeholders
  - These are the people who will be involved in the negotiation
- Determine each of the stakeholders "win conditions"
  - Win conditions are not always obvious
- Negotiate
  - Work toward a set of requirements that lead to "win-win"

Validating Requirements-I

- Is each requirement consistent with the overall objective 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?

Validating Requirements-II

- 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?