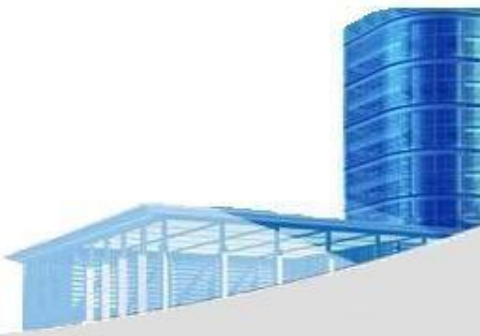
Ch.8 Understanding Requirements




概念设计
方案设计
施工图设计
竣工验收

Requirements Engineering

- **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 (cont.)

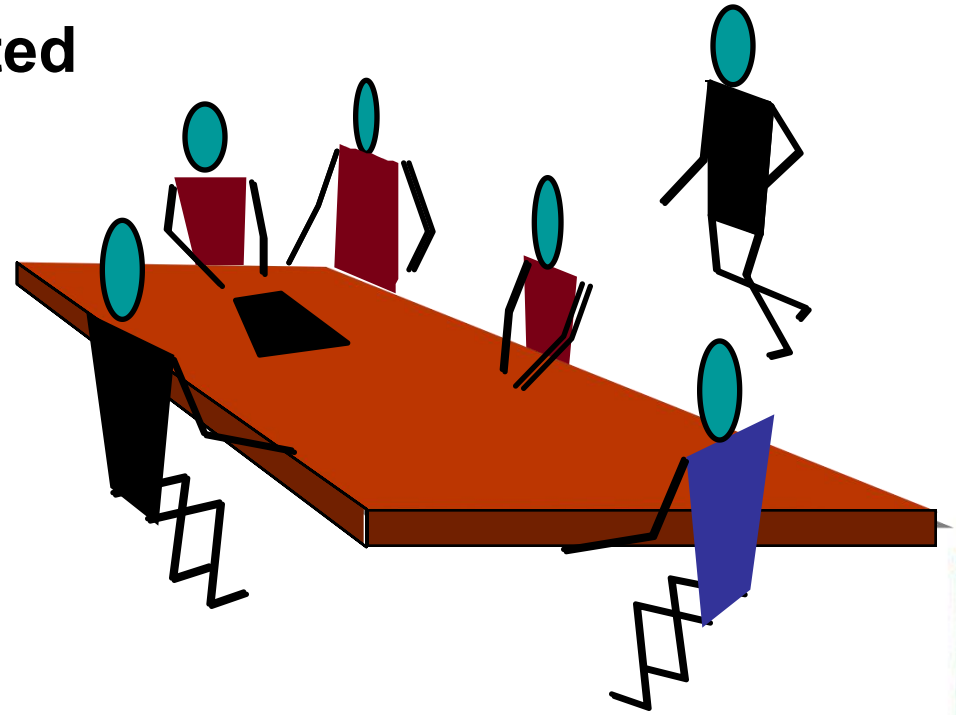
- **Specification**—can be any one (or more) of the following
  - A written **document**
  - A set of **models**
  - A formal mathematical
  - A collection of user scenarios (use-cases)
  - A **prototype**
- **Validation**—a review mechanism that looks for
  - errors in content or interpretation
  - 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 set of context-free 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

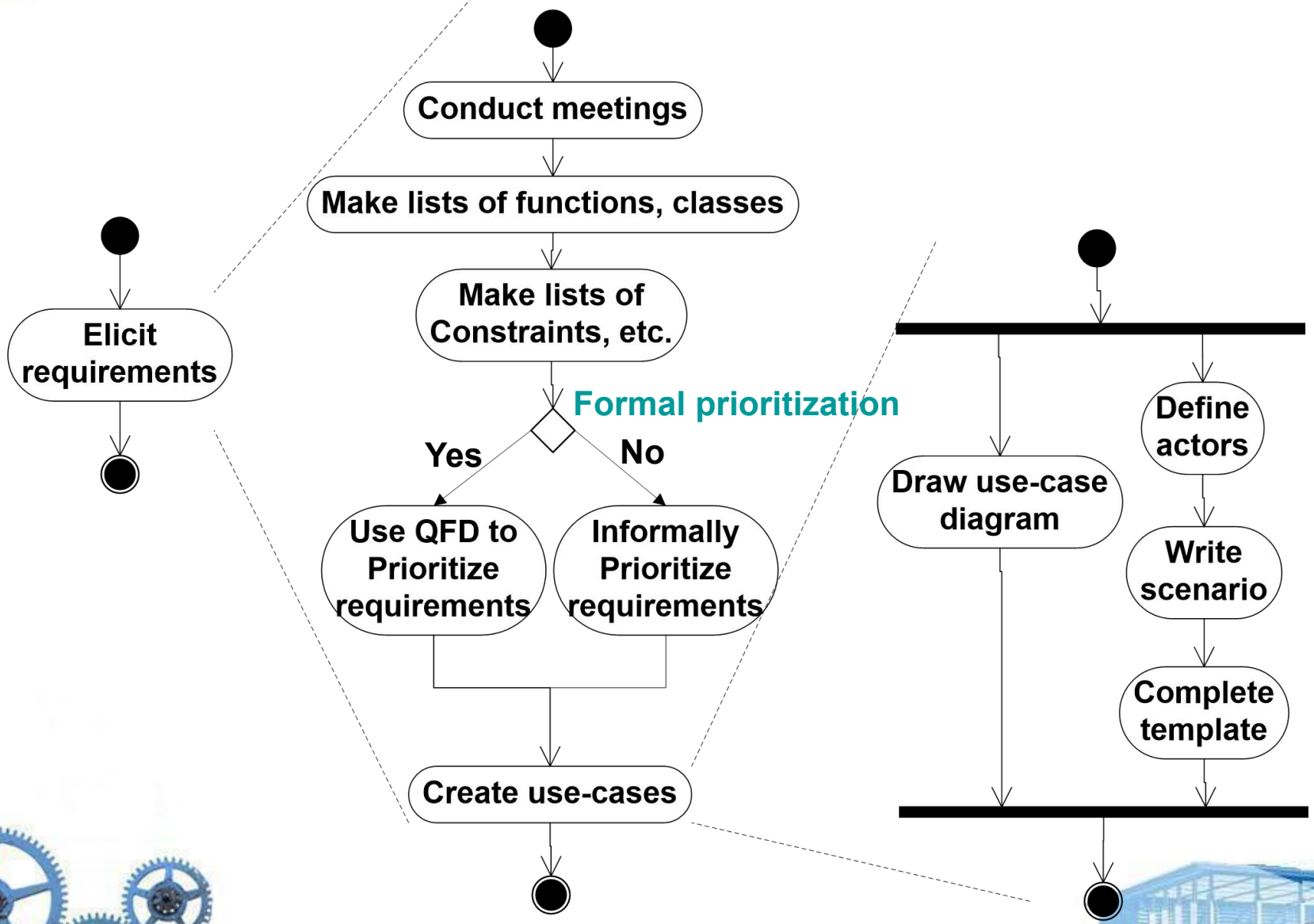


Eliciting Requirements (cont.)

- 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 hardest single part of building a software system is deciding what to build
  - Problem of scope
  - Problem of understanding
  - Problem of volatility

  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
- It identifies **three** types of requirements
  - Normal requirements
  - Expected requirements
  - Exciting requirements

Non-Functional Requirements

- **Non-Functional Requirment (NFR) – quality attribute, performance attribute, security attribute, or general system constraint. A two phase process is used to determine which NFR's are compatible:**
  - **The first phase is to create a matrix using each NFR as a column heading and the system SE guidelines a row labels**
  - **The second phase is for the team to prioritize each NFR using a set of decision rules to decide which to implement by classifying each NFR and guideline pair as complementary, overlapping, conflicting, or independent**

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?

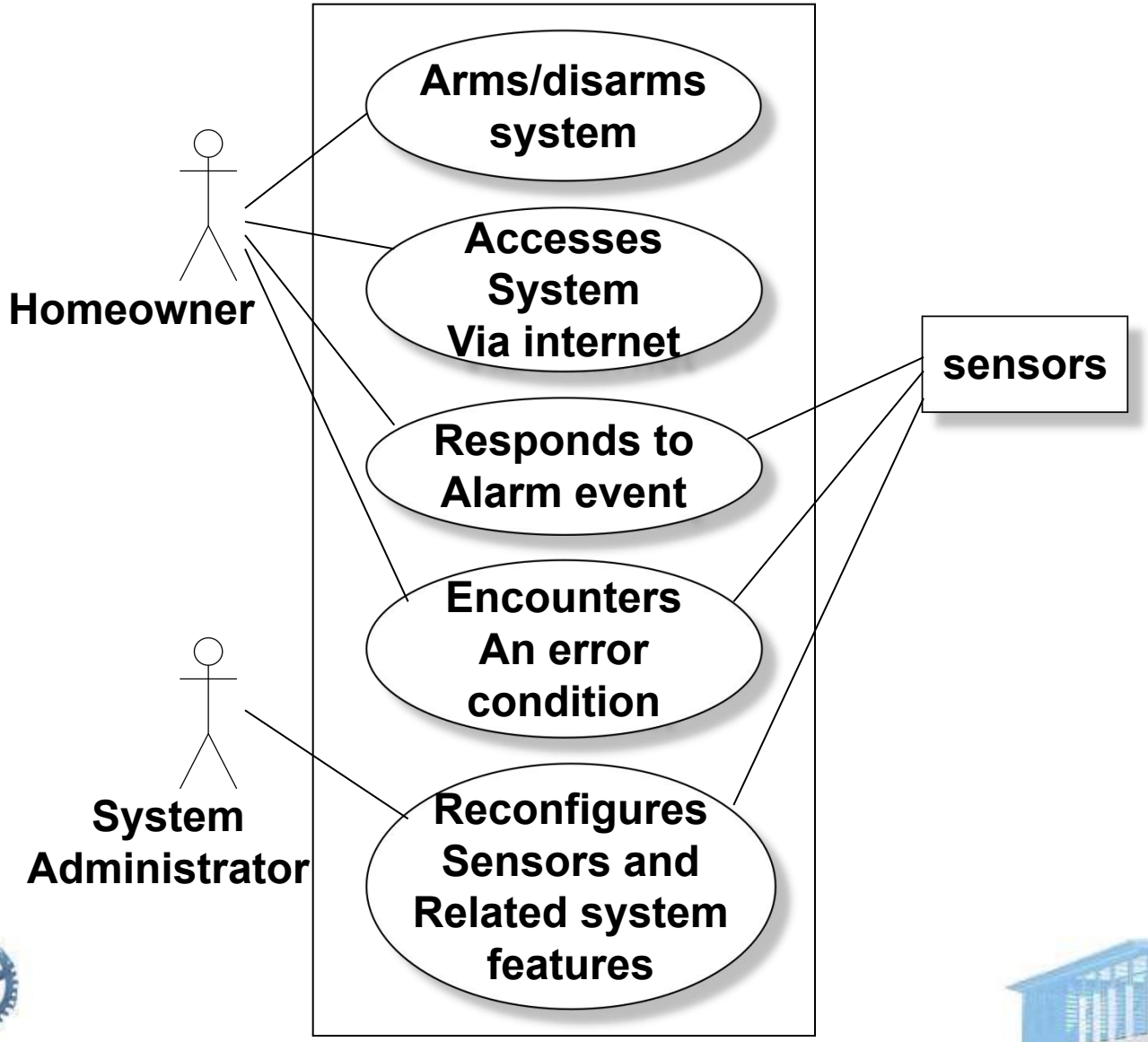
Use-Cases

- Each scenario answers the following questions (cont.)
  - 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?

A Example -- SafeHome

Our research indicates that the market for home security systems is growing at a rate of 40% per year. We would like to enter this market by building a microprocessor-based home security system that would protect against and/or recognize a variety of undesirable situations such as illegal entry, fire, flooding, and others. The product will use appropriate sensors to detect each situation, can be programmed by the homeowner, and will automatically telephone a monitoring agency when a situation is detected.

Use-Case Diagram

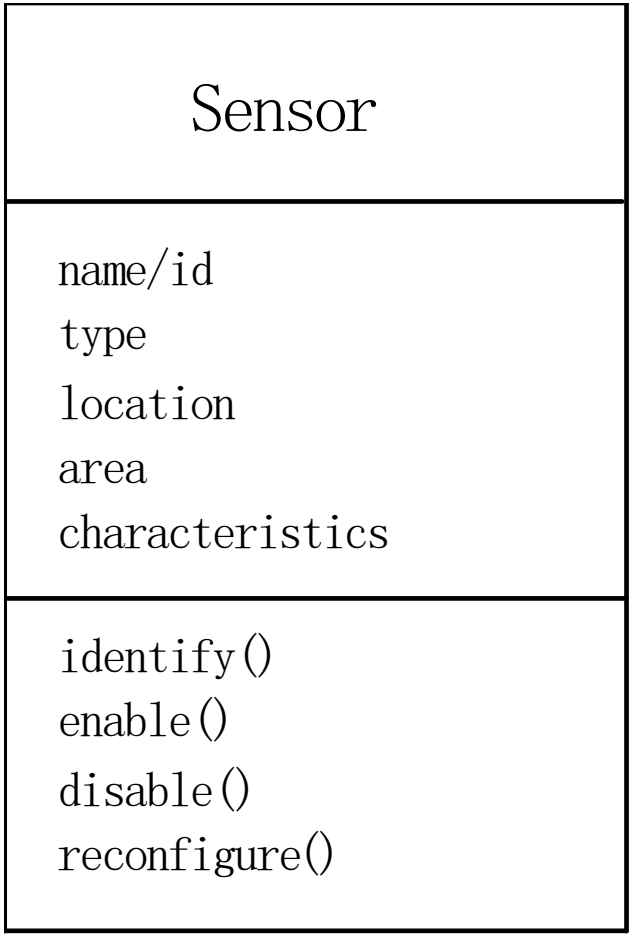


Building the Analysis Model

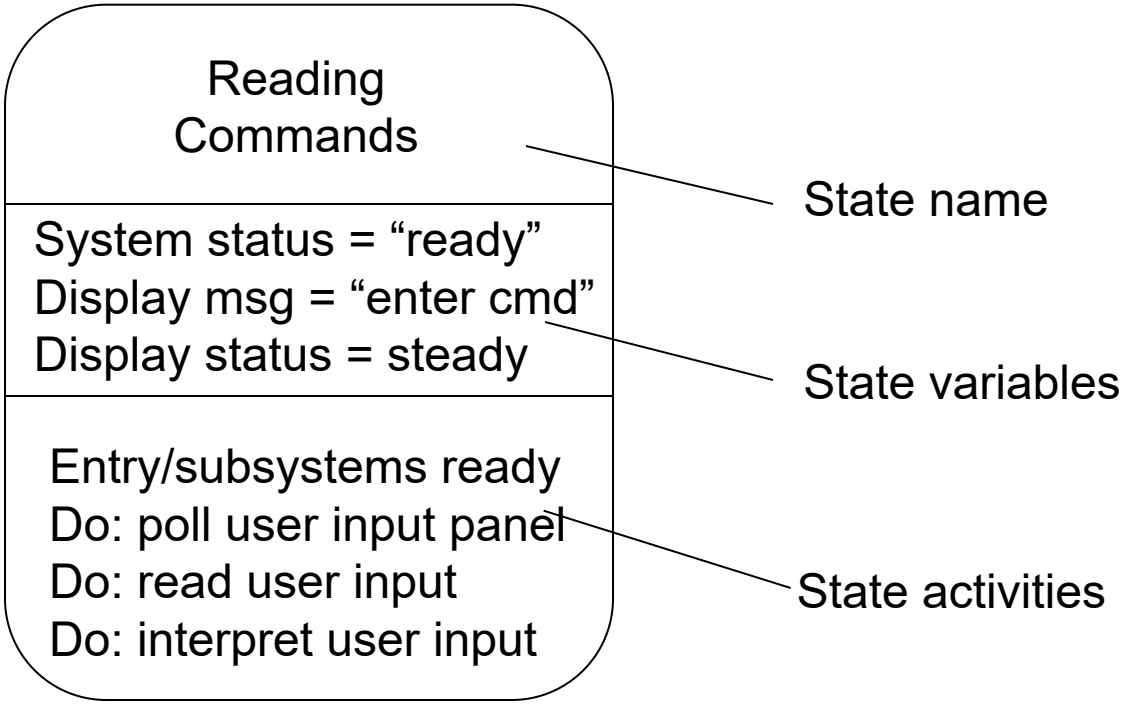
- Elements of the analysis model
  - Scenario-based elements
    - Use-case and user-case diagram
    - Sequence of activities within certain context
  - Class-based elements
    - Class diagram
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram

Class Diagram

From the *SafeHome* system …



State Diagram



State Diagram

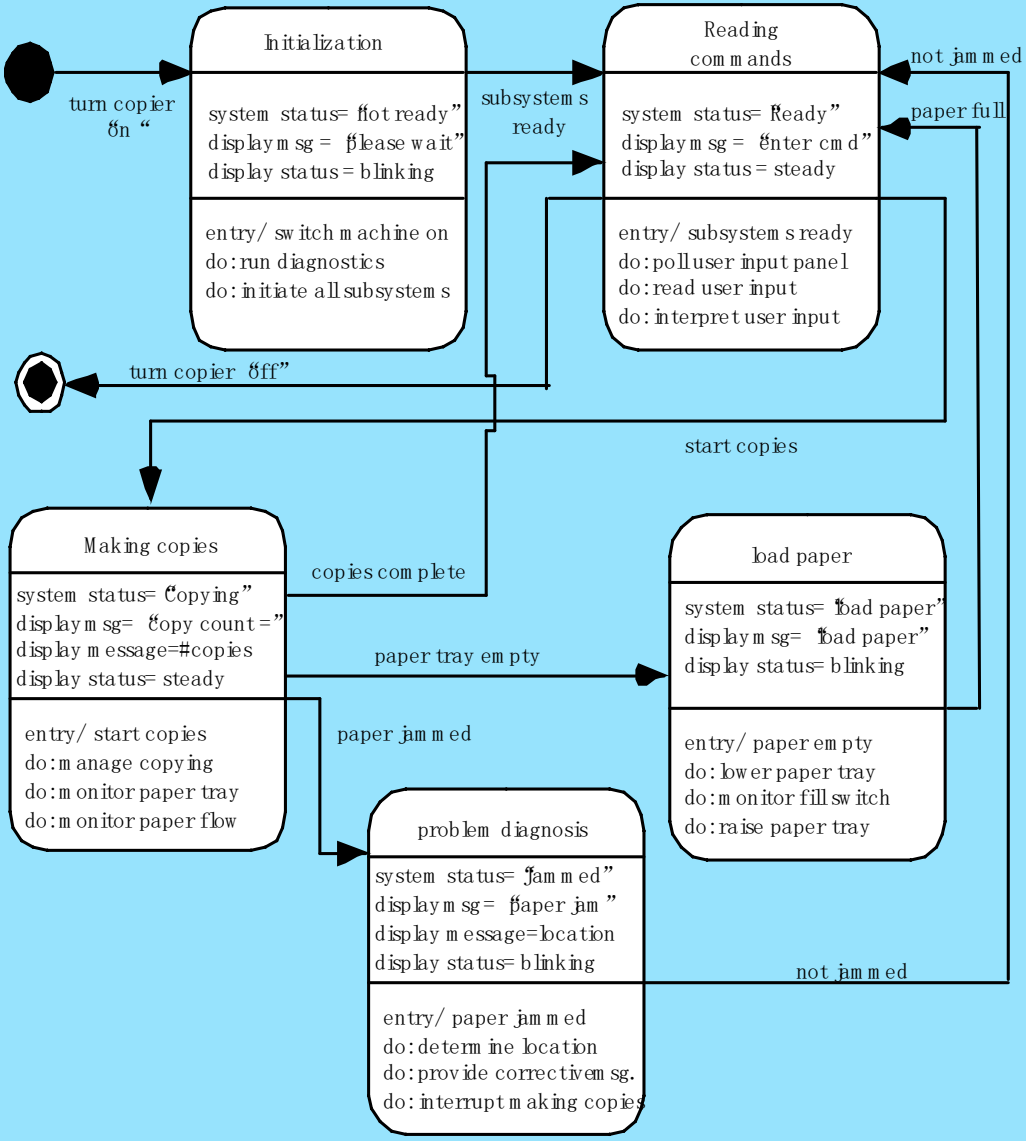


Figure 7.6 Preliminary UML state diagram for a photocopier

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.

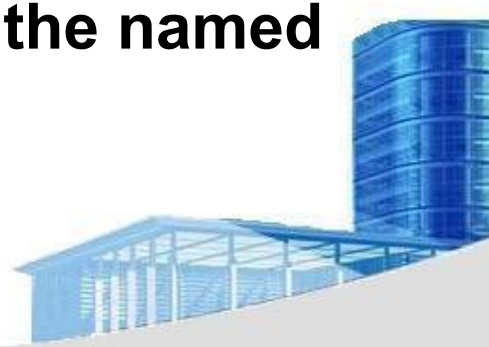
Analysis Patterns (cont.)

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: One 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 pe… negotiation
- **Determine each c…**
  - Win conditio…s are not alway…
- **Negotiate**
  - Work toward a set of requirements that lead to "win-win"

If different customers/users cannot agree on requirements, the risk of failure is very high.

Validating Requirements

- 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?

Validating Requirements (cont.)

- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
- Do any requirements conflict with other requirements?
- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?

Validating Requirements (cont.)

- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Has the requirements model been "partitioned" in a way that exposes progressively more detailed information about the system.
- Have requirements patterns been used to simplify the requirements model. Have all patterns been properly validated? Are all patterns consistent with customer requirements?

Requirements Monitoring

Especially needes in incremental development

- Distributed debugging – uncovers errors and determines their cause
- Run-time verification – determines whether software matches its specification
- Run-time validation – assesses whether evolving software meets user goals
- Business activity monitoring – evaluates whether a system satisfies business goals
- Evolution and codesign – provides information to stakeholders as the system evolves