软件工程概论
Software Engineering

刘伟

liuwei@xidian.edu.cn

88204608

CH4. Requirements

Content

4.1 The Requirements Process

- When a customer requests that we build a new system, the customer has ***some notion***(概念/想法) of what the system will do.
- Often, the new system ***replaces***(替换) an existing system or way of doing things. The new system is an enhancement or extension of a current (manual or automated) system.
- No matter whether its functionality is old or new, each software-based system has ***a purpose***, usually expressed in what the system can do.

4.1 The Requirements Process

- A ***requirement*** is a feature(特性/特征) of the system or a description of something the system is capable of doing in order to fulfill(完成/实现) the system’s purpose.

4.1 The Requirements Process

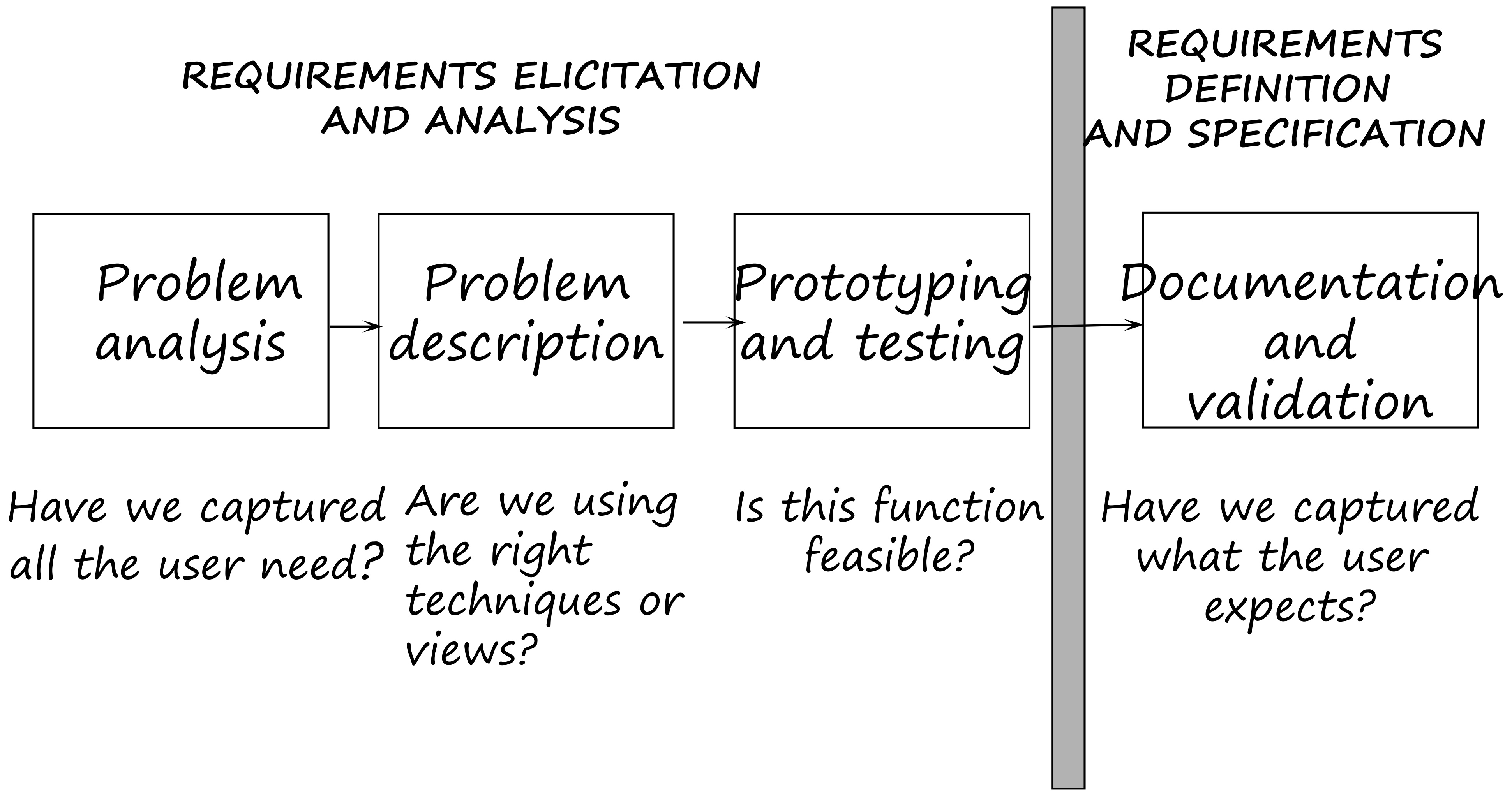
- A ***requirement*** is an expression of desired behavior
- A requirement deals with
  - objects or entities
  - the state they can be in
  - functions that are performed to change states or object characteristics
- Requirements focus on the customer needs, not on the solution or implementation
  - designate ***what*** behavior, without saying ***how*** that behavior will be realized

4.1 The Requirements Process

Why are Requirements important?

- Top factors that caused project to fail
  - Incomplete requirements (13.1%)
  - Lack of user involvement(12.4%)
  - Lack of resources(10.6%)
  - Unrealistic expectations(9.9%)
  - Lack of executive support(9.3%)
  - Changing requirements and specifications(8.7%)
  - Lack of planning(8.1%)
  - System no longer needed(7.5%)
- Some part of the requirements process is involved in almost all of these causes
- Requirements error can be expensive if not detected early

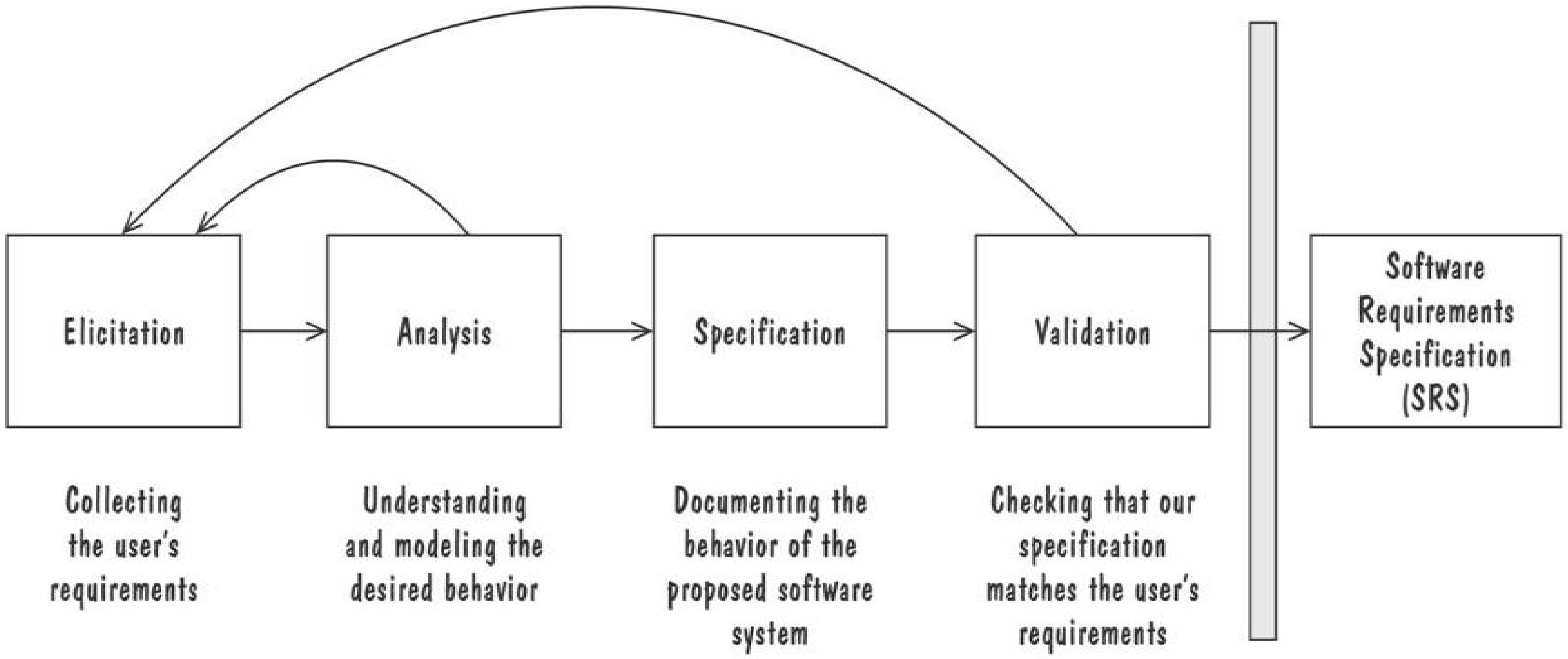
4.1 The Requirements Process
Process for determining Requirements


REQUIREMENTS ELICITATION AND ANALYSIS
REQUIREMENTS DEFINITION AND SPECIFICATION
Problem analysis
Problem description
Prototyping and testing
Documentation and validation
Have we captured all the user need?
Are we using the right techniques or views?
Is this function feasible?
Have we captured what the user expects?

The process of determining requirements

4.1 The Requirements Process
Process for capturing Requirements

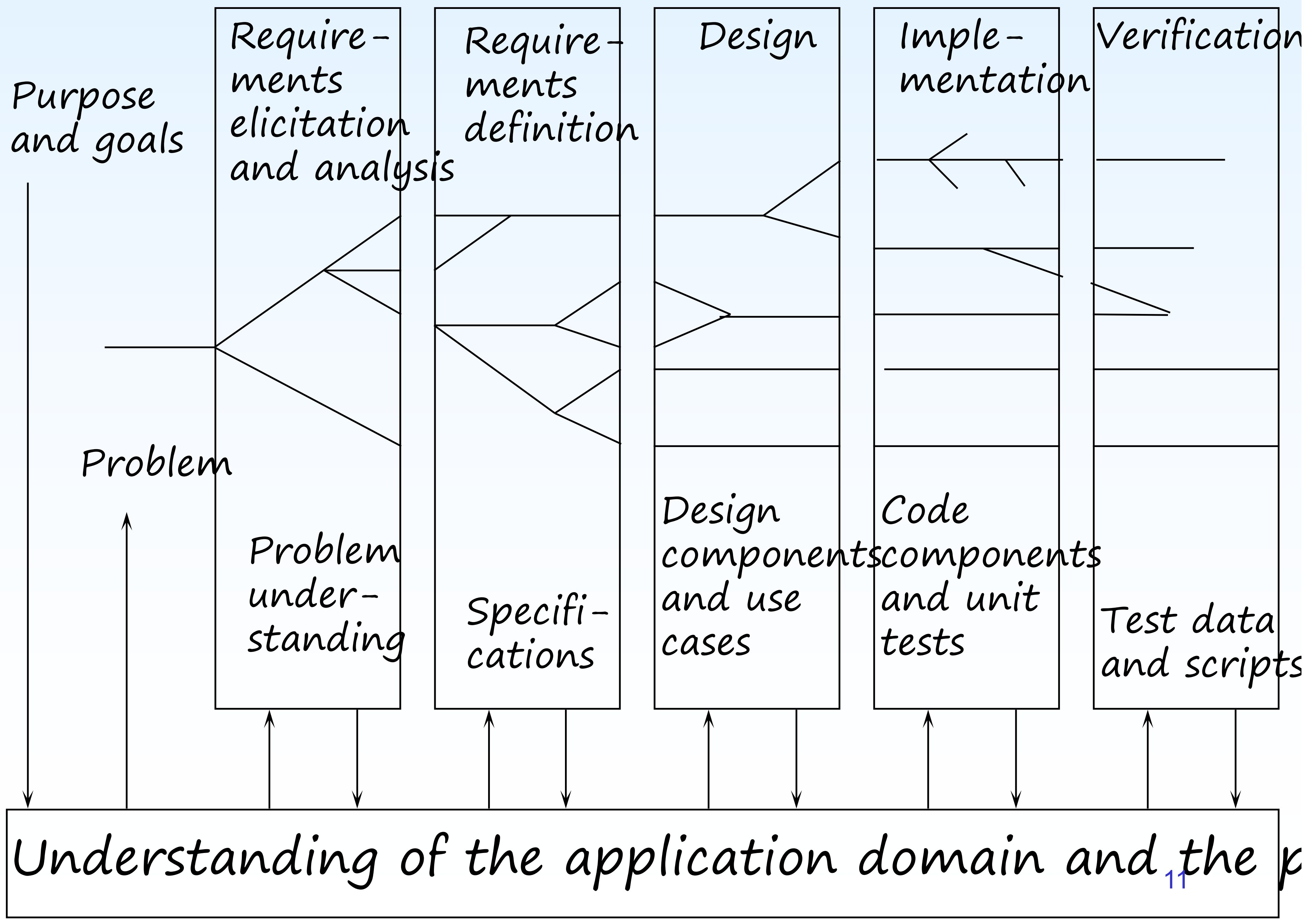
- Performed by the req. analyst or system analyst
- The final outcome is a Software Requirements Specification (SRS) document


Elicitation
Analysis
Specification
Validation
Software Requirements Specification (SRS)
Collecting the user's requirements
Understanding and modeling the desired behavior
Documenting the behavior of the proposed software system
Checking that our specification matches the user's requirements

4.1 The Requirements Process

Two Kinds of Requirements Documents

- Requirements definition: complete listing of what the customer expects the system to do.
- Requirements specification (规格说明书): restates the definition in technical terms so that the designer can start on the design.
- Configuration management: supports direct correspondence between the two documents.


Purpose and goals
Problem
Require-ments elicitation and analysis
Problem under-standing
Require-ments definition
Specifi-cations
Design
Design components and use cases
Imple-mentation
Code components and unit tests
Verification
Test data and scripts
Understanding of the application domain and the p

4.1 The Requirements Process

Configuration management

- Set of procedures that track
  - requirements that define what the system should do
  - design modules that are generated from requirements
  - program code that implements the design
  - tests that verify the functionality of the system
  - documents that describe the system

4.2 Requirements Elicitation

- Customers do not always undertand what their needs and problems are
- It is important to discuss the requirements with everyone who has a stake in the system
- Come up with agreement on what the requirements are
  - If we can not agree on what the requirements are, then the project is doomed to fail

4.2 Requirements Elicitation

- Stakeholder
  - **Customers**: buy the software after it is developed
  - **Users**: use the system
  - **Domain experts**: familiar with the problem that the software must automate
  - **Market Researchers**: conduct surveys to determine future trends and potential customers
  - **Lawyers or auditors**: familiar with government, safety, or legal requirements
  - **Software engineers** or other technology experts

4.2 Requirements Elicitation

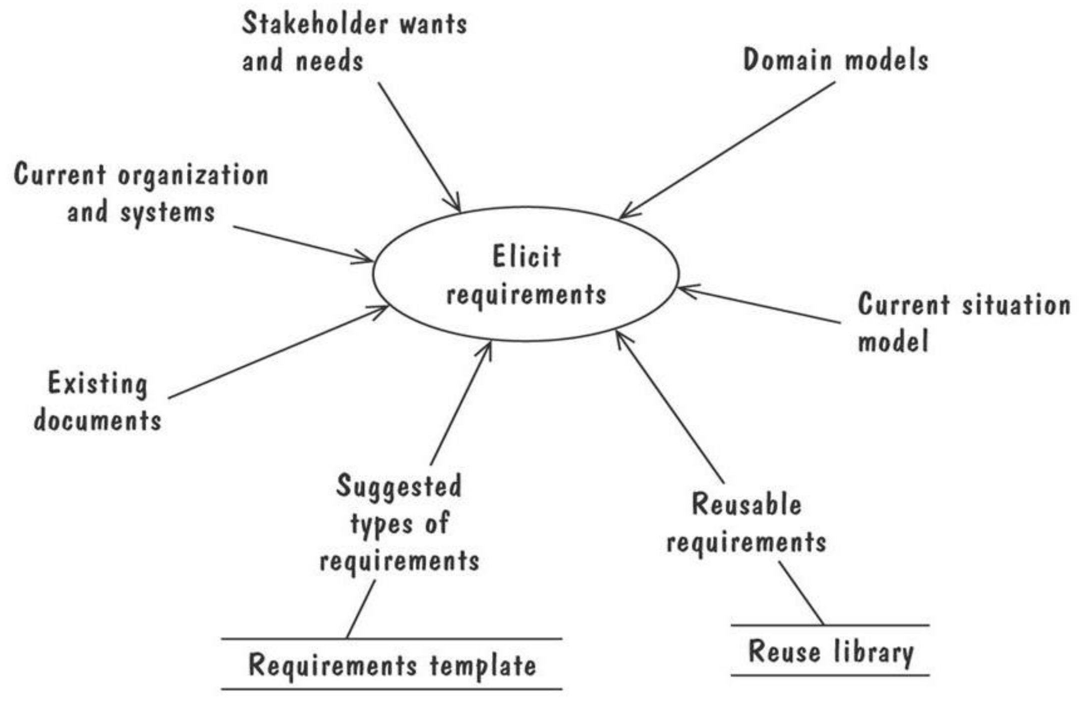
- Helpful to all parties in understanding what is really needed.
- It is also useful when a software development project is constrained by time or resources.
- A requirements addresses the purpose of the system without regard for how the system is to be implemented.

4.2 Requirements Elicitation

- Means of Eliciting Requirements
  - Interviewing stake holders
  - Reviewing available documentations
  - Observing the current system (if one exists)
  - Apprenticing with users to learn about user's task in more details
  - Interviewing users or stakeholders in groups
  - Using domain specific strategies, such as Joint Application Design, or PIECES
  - Brainstorming with current and potential users

4.2 Requirements Elicitation

- Means of Eliciting Requirements
  - The Volere requirements process model suggests some additional sources for requirements



4.3 Types of Requirements

- **Functional requirement**: describes required behavior in terms of required activities
- **Quality requirement** or **nonfunctional requirement**: describes some quality characteristic that the software must possess
- **Design constraint**: a design decision such as choice of platform or interface components
- **Process constraint**: a restriction on the techniques or resources that can be used to build the system

P150

4.3 Types of Requirements

- Functional Requirements examples:
  - System shall communicate with external system X.
  - What conditions must be met for a message to be sent.
- Nonfunctional Requirements examples:
  - Paychecks distributed no more than 4 hours after initial date are read.
  - System limits access to senior managers.

4.3 Types of Requirements

- Fit criteria form objective standards for judging whether a proposed solution satisfies the requirements（**P104**）
  - It is easy to set fit criteria for quantifiable requirements
  - It is hard for subjective quality requirements
- Three ways to help make requirements testable
  - Specify a quantitative description for each adverb and adjective
  - Replace pronouns with specific names of entities
  - Make sure that every noun is defined in exactly one place in the requirements documents

4.3 Types of Requirements – Resolving Conflicts

- Different stakeholder has different set of requirements
  - potential conflicting ideas
- Need to prioritize requirements
- Prioritization might separate requirements into three categories
  - *essential*: absolutely must be met
  - *desirable*: highly desirable but not necessary
  - *optional*: possible but could be eliminated

4.3 Types of Requirements

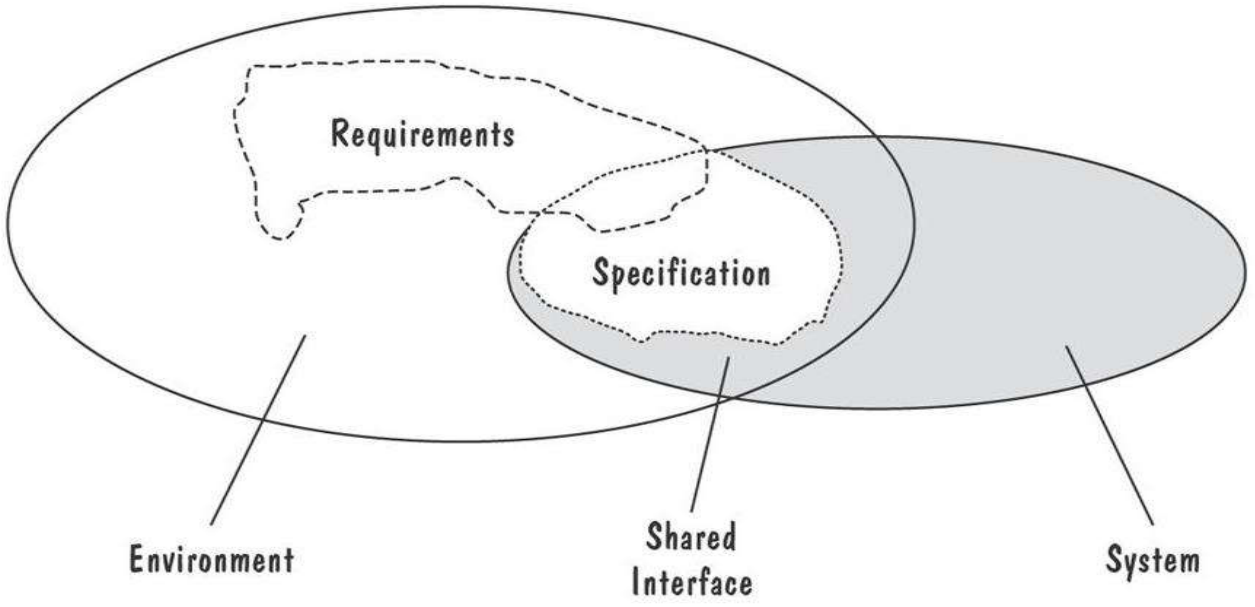
- The requirements definition and specification documents describe everything about how the system is to interact with its environment. Included are the following kinds of items.
  - Physical environment
  - Interfaces
  - Users and human factors
  - Functionality
  - Documentation
  - Data
  - Resources
  - Security
  - Quality assurance

4.3 Types of Requirements

- Two Kinds of Requirements Documents
  - **Requirements definition**: a complete listing of everything the customer wants to achieve
  - **Requirements specification**: restates the requirements as a specification of how the proposed system shall behave
  - Example: Turnstile(十字转门).

4.3 Types of Requirements

- Two Kinds of Requirements Documents
  - Requirements defined anywhere within the environment's domain, including the system's interface
  - Specification restricted only to the intersection between environment and system domain



4.4 Characteristics of Requirements

- The high quality requirements have some characteristics.
  - Correct
  - Consistent
  - Unambigious
  - Complete
  - Feasible
  - Relevant
  - Testable
  - Traceable

4.5 Modeling Notations

- As with many activities in computer science, requirements definition is often performed best by ***working down from the top.***
- Notice that natural language may not be the precise and unambiguous medium needed for expressing the system's functionality and the relationship of its relevant parts.

4.5 Modeling Notations

- It is important to have standard notations for modeling, documenting, and communicating decisions
- Modeling helps us to understand requirements thoroughly
  – Holes in the models reveal unknown or ambiguous behavior
  – Multiple, conflicting outputs to the same input reveal inconsistencies in the requirements

4.5 Modeling Notations

- The methods of modeling notations
  - Entity-Relationship Diagrams
  - Event Traces
  - State Machines
  - Data-Flow Diagrams
  - Functions and Relations
  - Logic
  - Algebraic Specifications

4.5 Modeling Notations – ER Diagrams

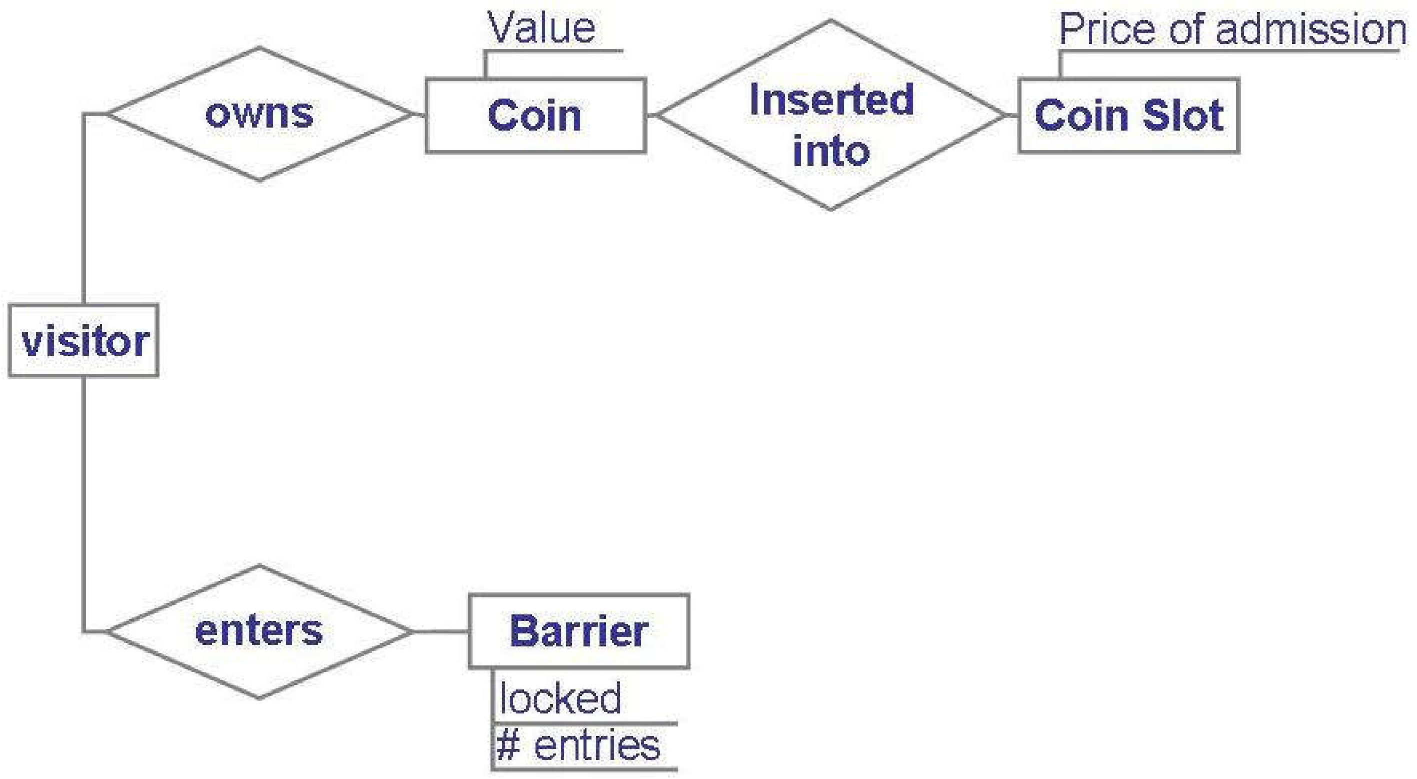
- ER diagrams is a popular graphical notational paradigm for representing conceptual models
- A conceptual models identify what objects or entities are involved in the problem, what they look like (by define their attributes), and how they relate to one other.
- We use ER diagrams to
  - Model the relationships among objects in a problem
  - Model the structure of a software application
  - Describe database schema

4.5 Modeling Notations – ER Diagrams

- Has three core constructs
  - An *entity*: depicted as a rectangle, represents a collection of real-world objects that have common properties and behaviors
  - A *relationship*: depicted as an edge between two entities, with diamond in the middle of the edge specifying the type of relationship
  - An *attribute*: an annotation on an entity that describes data or properties associated with the entity

4.5 Modeling Notations – ER Diagrams

- Entity diagram of turnstile problem



4.5 Modeling Notations – ER Diagrams

- ER diagrams are popular because
  - they provide an overview of the problem to be addressed
  - the view is relatively stable when changes are made to the problem's requirements
- The simplicity of ER notations is deceptive; in fact, it is quiet difficult to use ER modeling notations well in the practice.
- The primary criteria:
  - Whether a choice results in a clearer description
  - Whether a choice unnecessarily constrains design decisions

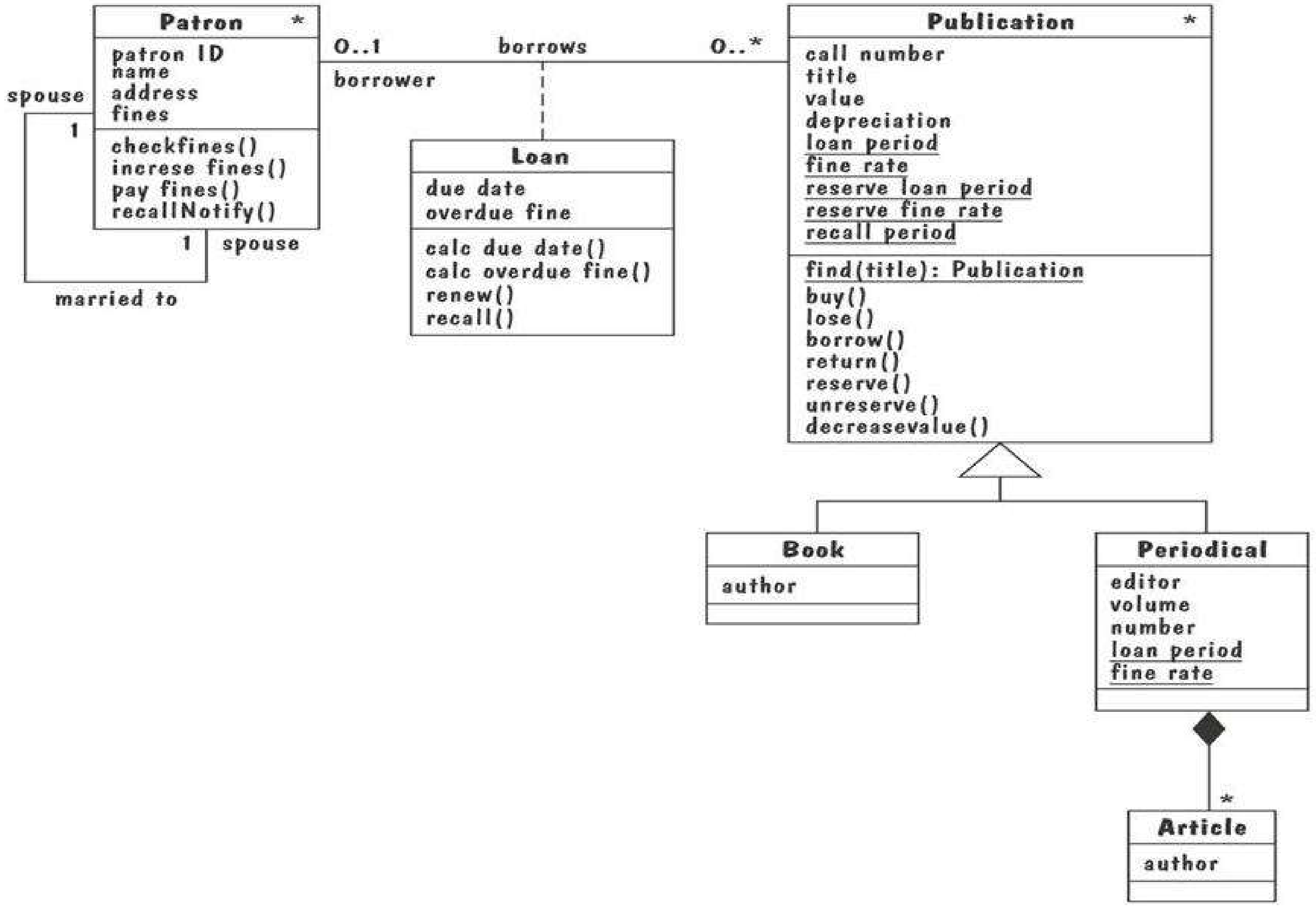
4.5 Modeling Notations – ER Diagrams

Example: UML Class Diagram

- **UML (Unified Modeling Language)** is a collection of notations used to document software specifications and designs
- It represents a system in terms of
  - *objects*: akin to entities, organized in classes that have an inheritance hierarchy
  - *methods*: actions on the object's variables
- The **class diagram** is the flagship model in any UML specification
  - A sophisticated ER diagram relating the classes (entities) in the specification

4.5 Modeling Notations – ER Diagrams

Example: UML Class Diagram



4.5 Modeling Notations – ER Diagrams

Example: UML Class Diagram

- Attributes and operations are associated with the class rather than instances of the class
- A **class-scope attribute** represented as an underlined attribute, is a data value that is shared by all instances of the class
- A **class-scope operation** written as underlined operation, is an operation performed by the abstract class rather than by class instances
- An **association**, marked as a line between two classes, indicates a relationship between classes' entities

4.5 Modeling Notations – ER Diagrams

Example: UML Class Diagram

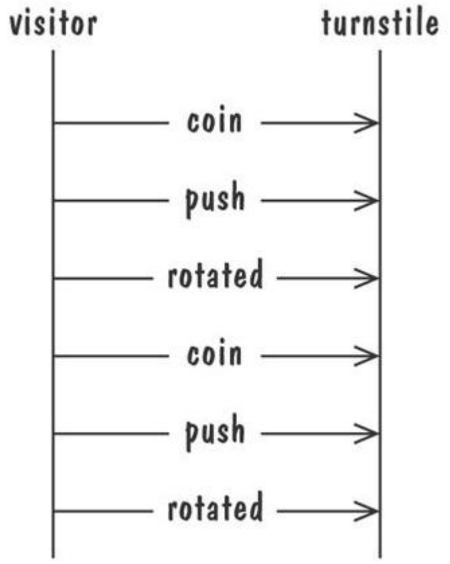
- **Aggregate association** is an association that represents interaction, or events that involve objects in the associated (marked with white diamond)
  - “*has-a*” relationship
- **Composition association** is a special type of aggregation, in which instances of the compound class are physically constructed from instances of component classes (marked with black diamond)

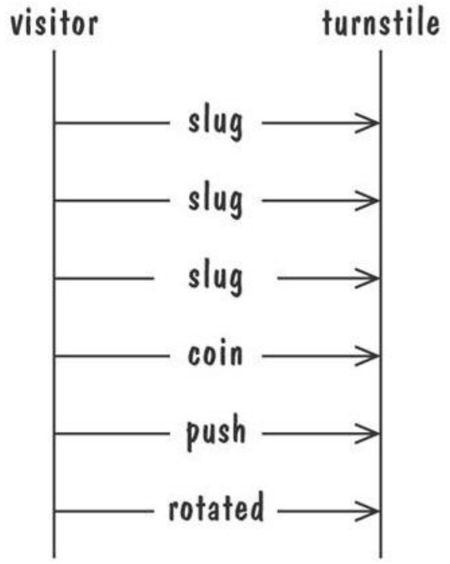
4.5 Modeling Notations – Event Traces

- A graphical description of a sequence of events that are exchanged between real-world entities
  - *Vertical line*: the timeline of distinct entity, whose name appears at the top of the line
  - *Horizontal line*: an event or interaction between the two entities bounding the line
  - Time progresses from top to bottom
- Each graph depicts a single trace, representing one of several possible behaviors
- Traces have a semantic that is relatively precise, simple and easy to understand

4.5 Modeling Notations – Event Traces

- Graphical representation of two traces for the turnstile problem
  - trace on the left represents typical behavior
  - trace on the right shows exceptional behavior





4.5 Modeling Notations – Event Traces

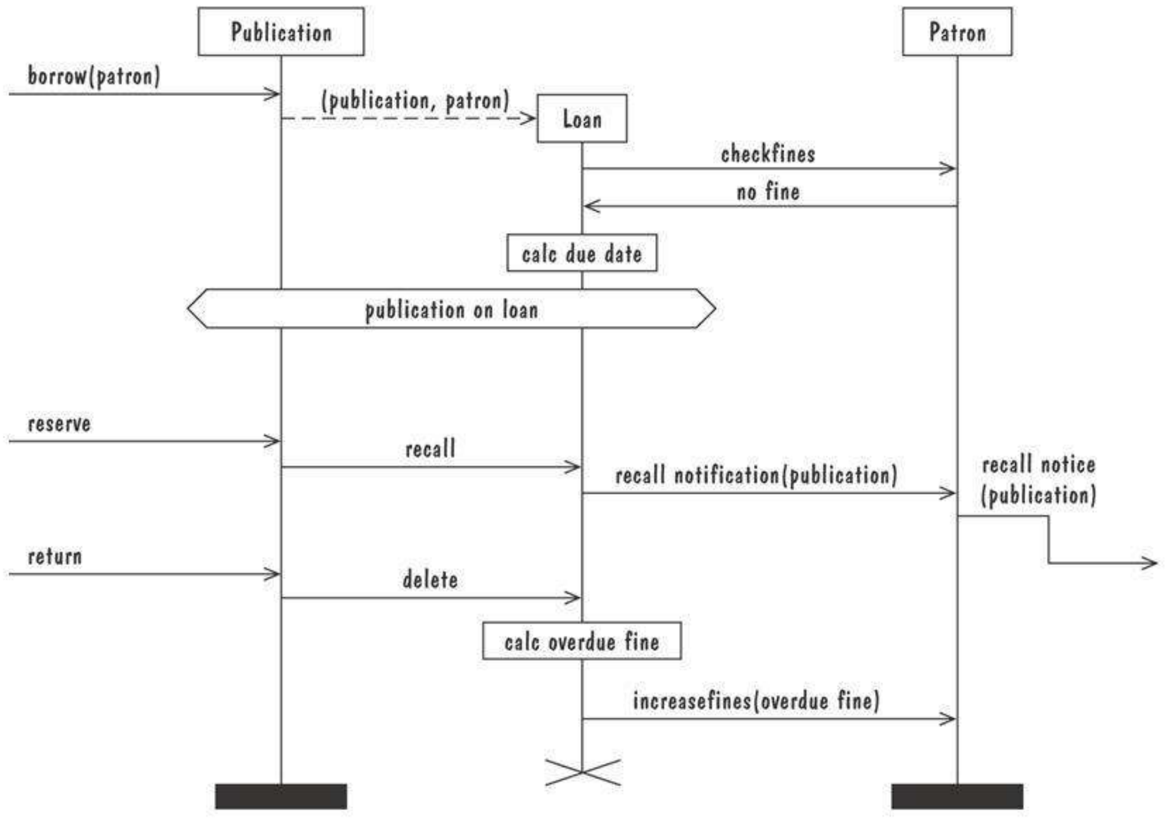
Example: Message Sequence Chart

- An enhanced event-trace notation, with facilities for creating and destroying entities, specifiying actions and timers, and composing traces
  - *Vertical line* represents a participating entity
  - *A message* is depicted as an arrow from the sending entity to the receiving entity
  - *Actions* are specified as labeled rectangles positioned on an entity's execution line
  - *Conditions* are important states in an entity's evolution, represented as labeled hexagon

4.5 Modeling Notations – Event Traces

Example: Message Sequence Chart

- Message sequence chart for library loan transaction

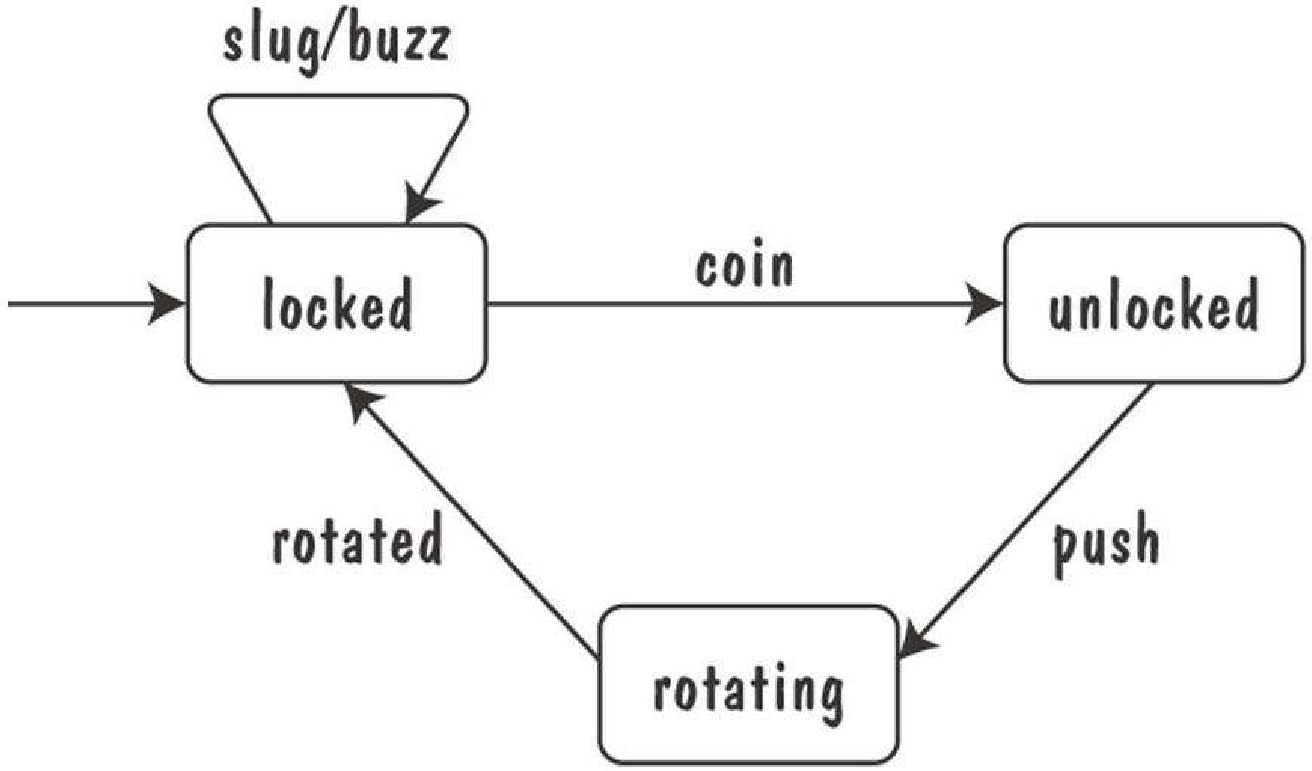


4.5 Modeling Notations – State Machines

- A graphical description of all dialog between the system and its environment
  - Node (*state*) represents a stable set of conditions that exists between event occurences
  - Edge (*transition*) represents a change in behavior or condition due to the occurrence of an event
- Useful both for specifying dynamic behavior and for describing how behavior should change in response to the history of events that have already occurred.

4.5 Modeling Notations – State Machines

- Finite state machine model of the turnstile problem



4.5 Modeling Notations – State Machines

- **A path**: starting from the machine's initial state and following transitions from state to state
  - A trace of observable events in the environment
- **Deterministic state machine:** for every state and event there is a unique response

4.5 Modeling Notations – State Machines

Example: UML Statechart Diagrams

- A UML statechart diagram depicts the dynamic behavior of the objects in a UML class
  - UML class diagram has no information about how the entities behave, how the behaviors change
- A UML model is a collection of concurrently executing statecharts
- UML statechart diagram has a rich syntax, including state hierarchy, concurrency, and intermachine communication

4.5 Modeling Notations – State Machines

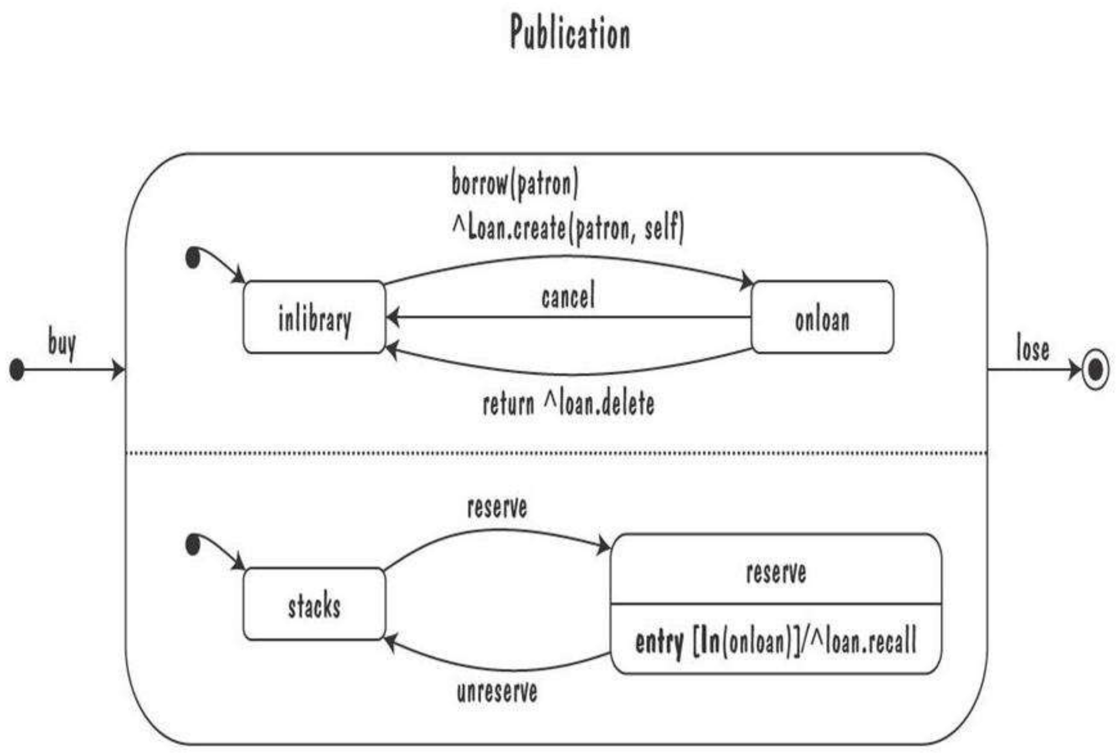
Example: UML Statechart Diagrams

- **State hierarchy** is used to unclutter diagrams by collecting into superstate those states with common transitions
- A **superstate** can actually comprise multiple concurrent submachines, separated by dashed lines
  - The submachines are said to operate *concurrently*

4.5 Modeling Notations – State Machines

Example: UML Statechart Diagrams

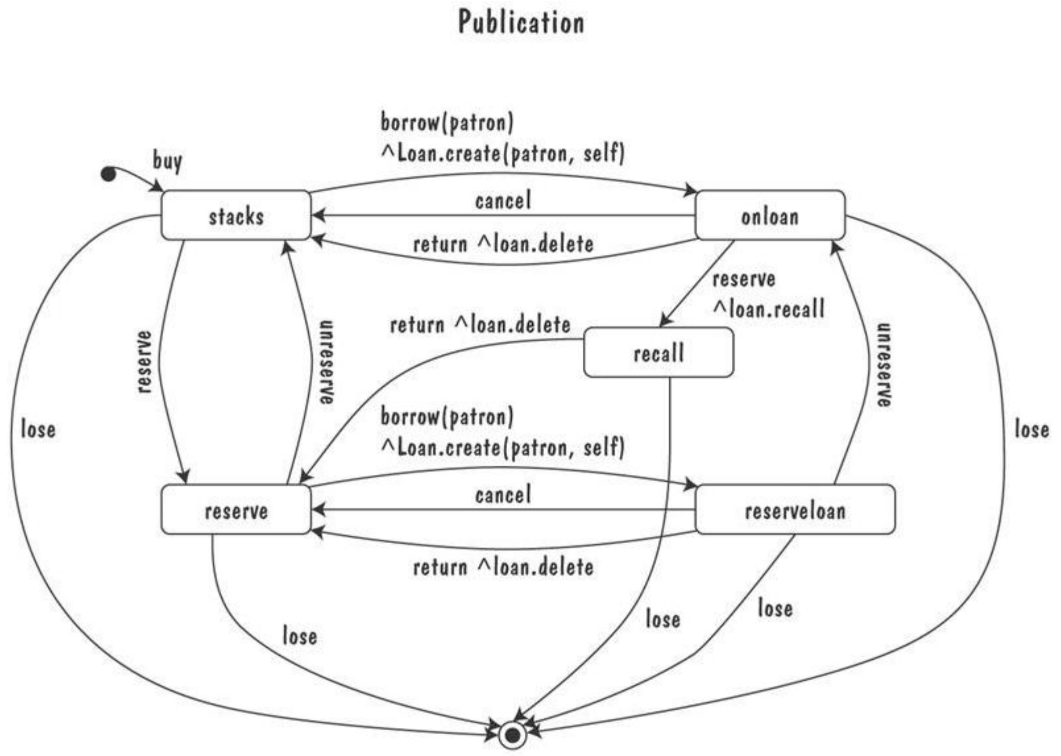
- The UML statechart diagram for the Publication class from the Library class model.



4.5 Modeling Notations – State Machines

Example: UML Statechart Diagrams

- An equivalent statechart for Publication class that does not make use of state hierarchy or concurrency
  - comparatively messy and and repetitive



4.5 Modeling Notations – State Machines

Ways of Thinking about State

- Equivalence classes of possible future behavior
- Periods of time between consecutive event
- Named control points in an object's evolution
- Partition of an object's behavior

4.5 Modeling Notations – State Machines

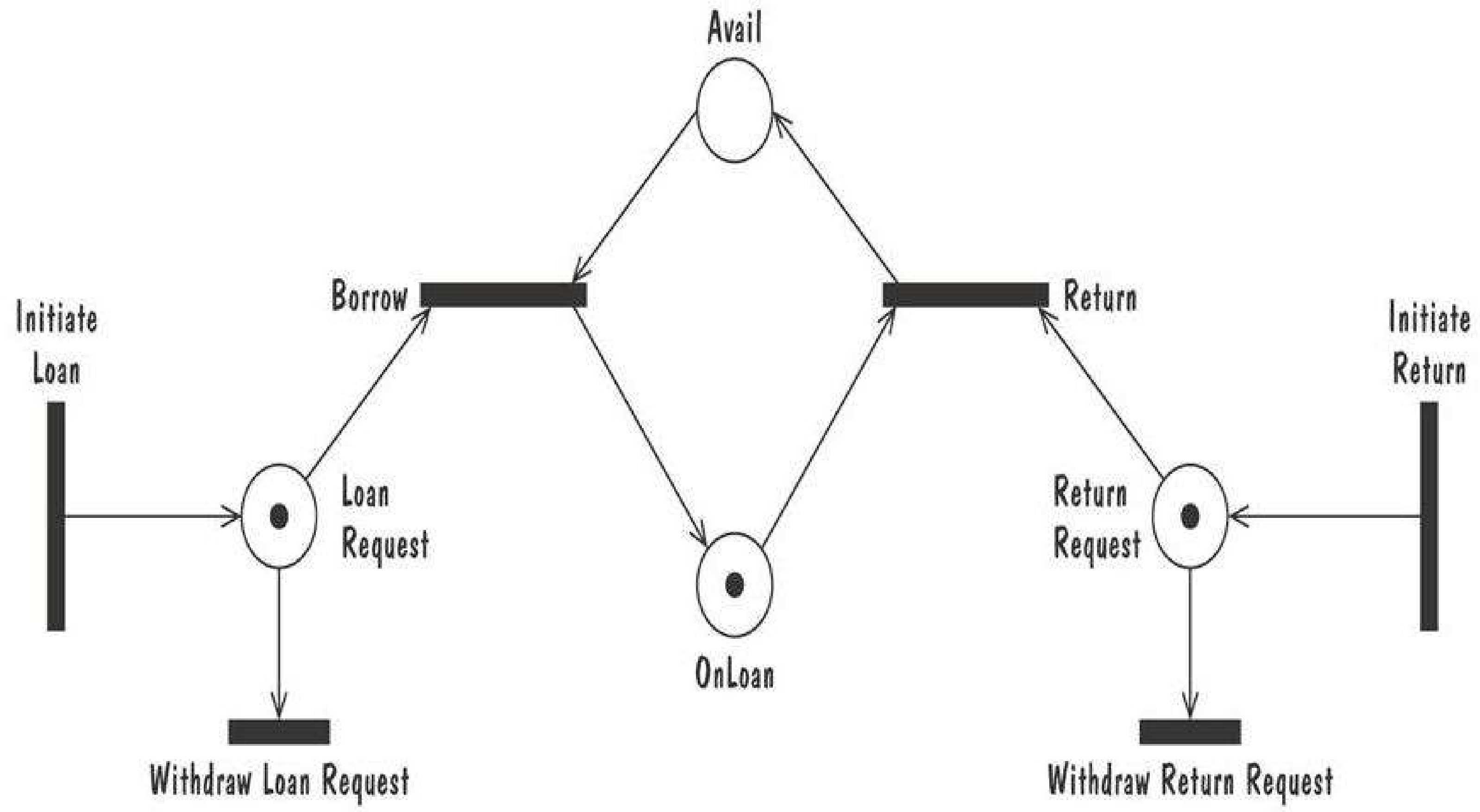
Example: Petri Nets

- A form or state-transition notation that is used to model concurrent activities and their interaction
  - Circles (*places*) represent activities or conditions
  - Bars represents *transitions*
  - *Arcs* connect a transition with its input places and its output places
  - The places are populated with *tokens,* which act as enabling conditions for the transitions
  - Each arc can be assigned *a weight* that specifies how many tokens are removed from arc's input place, when the transition fires

4.5 Modeling Notations – State Machines

Example: Petri Nets

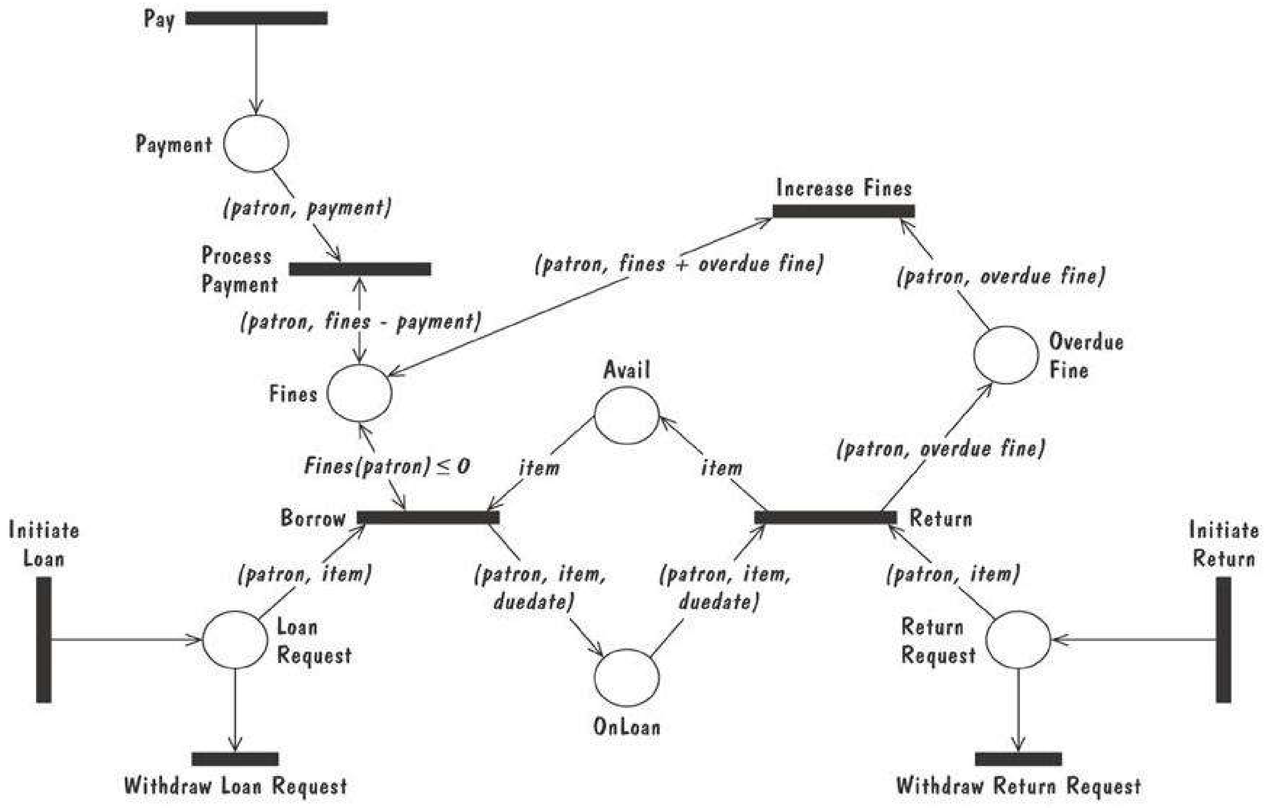
- Petri net of book loan



4.5 Modeling Notations – State Machines

Example: Petri Nets

- A high level Petri net specification for the library problem

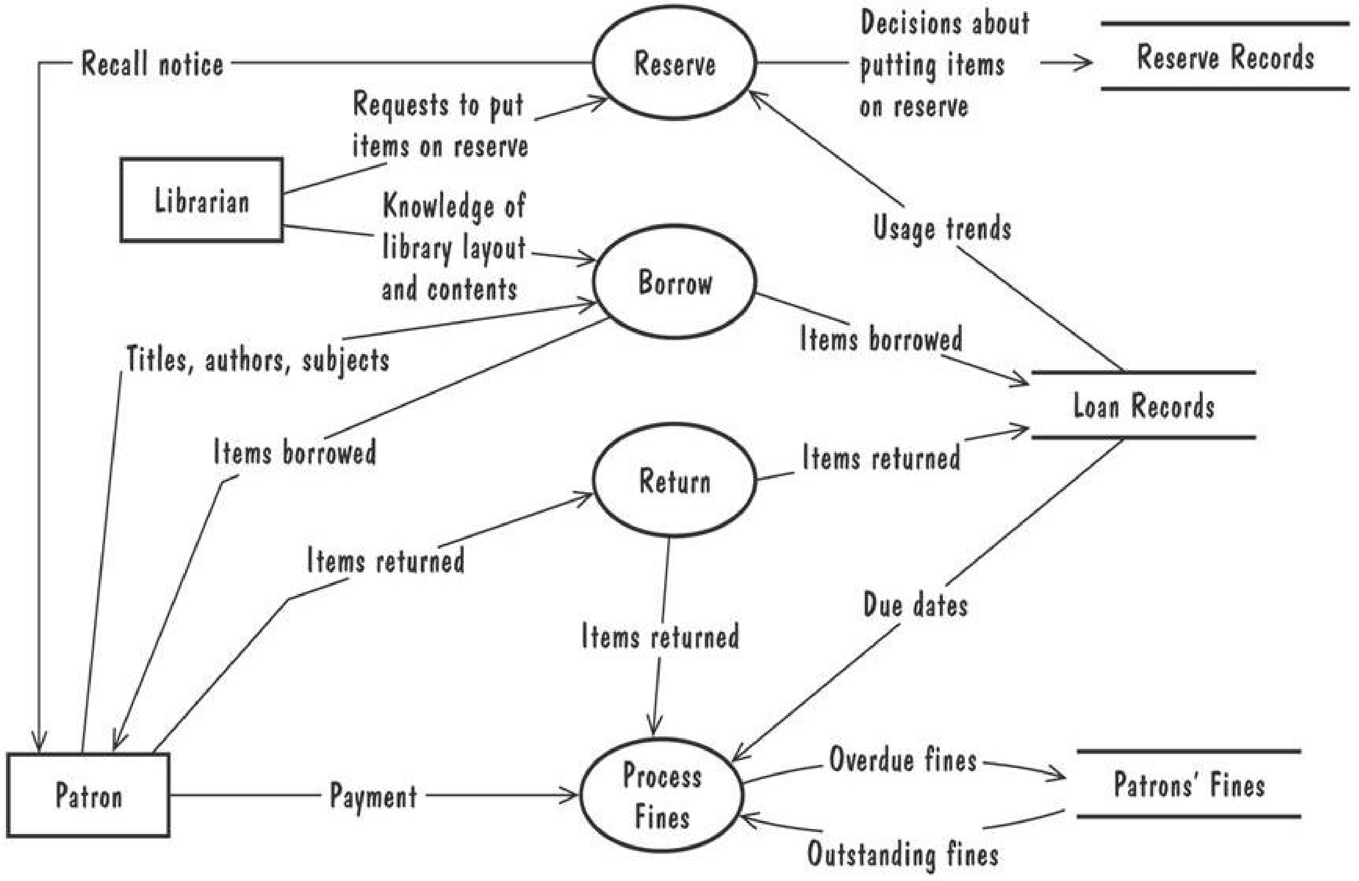


4.5 Modeling Notations – Data-Flow Diagrams

- ER diagram, event trace, state machines depict only lower-level behaviors
- A data-flow diagram (DFD) models functionality and the flow of data from one function to another
  - A buble represents a *process*
  - An arrow represents *data flow*
  - A *data store*: a formal repository or database of information
  - Rectangles represent *actors*: entities that provide input data or receive the output result

4.5 Modeling Notations – Data-Flow Diagrams

- A high-level data-flow diagram for the library problem



4.5 Modeling Notations – Data-Flow Diagrams

- Advantage:
  - Provides an intuitive model of a proposed system's high-level functionality and of the data dependencies among various processes
- Disadvantage:
  - Can be aggravatingly ambiguous to a software developer who is less familiar with the problem being modeled

4.5 Modeling Notations – Data-Flow Diagrams

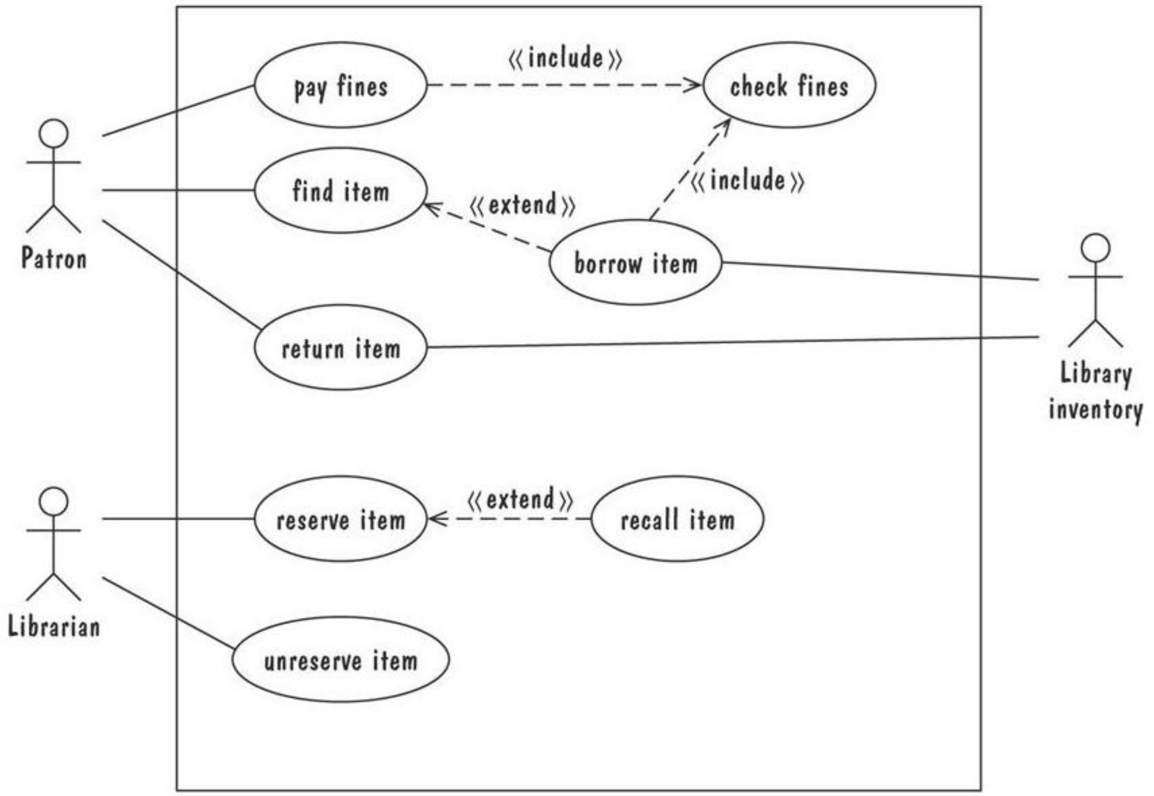
Example: Use Cases

- Components
  - A large box: *system boundary*
  - Stick figures outside the box: *actors*, both human and systems
  - Each oval inside the box: a use case that represents some major required functionality and its variant
  - A line between an actor and use case: the actor participates in the use case
- Use cases do not model all the tasks, instead they are used to specify user views of essential system behavior

4.5 Modeling Notations – Data-Flow Diagrams

Example: Use Cases

- Library use cases including borrowing a book, returning a borrowed book, and paying a library fine.

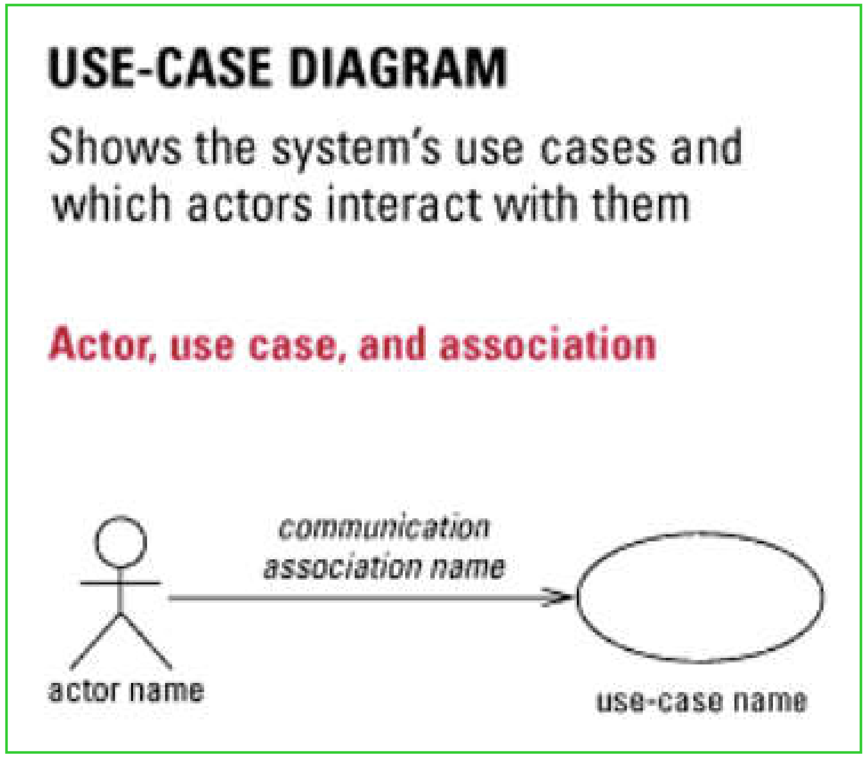


4.5 Modeling Notations – Data-Flow Diagrams

用例图（Use-Case Diagram）(描述需求的UML图)

Actor表示要与本系统发生交互的一个角色单元（人或其他系统）。

use-case表示由本系统提供的一个功能单元。


USE-CASE DIAGRAM

Shows the system's use cases and which actors interact with them

Actor, use case, and association

4.5 Modeling Notations – Data-Flow Diagrams

- Example: ATM withdrawal – scenario
  - 1. enter ATM card
  - 2. enter PIN number
  - 3. system verifies that PIN is correct
  - 4. system asks “show balance” or “withdrawal”
  - 5. customers selects “withdrawal”
  - 6. system asks amount
  - 7. customers enters amount
  - 8. system verifies amount <= available balance
  - 9. system dispenses money
  - 10. system asks if receipt is required
  - 11. customers requests receipt
  - 12. system prints receipt
  - 13. systems returns ATM card

4.5 Modeling Notations – Data-Flow Diagrams

- Variant: PIN incorrect
  - At step 2, the system determines the PIN is incorrect
  - 2b. Ask user to re-enter PIN
  - 3b. Return to primary scenario at step 3

- Variant: user requests account balance
  - At step 5, user selects “show balance”
  - 5c. system displays account balance
  - 6c. system asks if other services are required
  - 7c. customers confirms
  - 8c. Return to primary scenario at step 4

4.5 Modeling Notations – Data-Flow Diagrams

- Actors
  - the customer withdrawing money
  - the database system that contains account information
- Use cases
  - Enter PIN
  - Verify PIN
  - Select Service (withdrawal or balance)
  - Enter amount
  - Check account balance
  - Print receipt
  - Dispense money

4.5 Modeling Notations – Data-Flow Diagrams

- Example: ATM withdrawal – scenario
  - 1. enter ATM card
  - 2. enter PIN number
  - 3. system verifies that PIN is correct
  - 4. system asks "show balance" or "withdrawal"
  - 5. customers selects "withdrawal"
  - 6. system asks amount
  - 7. customers enters amount
  - 8. system verifies amount <= available balance
  - 9. system dispenses money
  - 10. system asks if receipt is required
  - 11. customers requests receipt
  - 12. system prints receipt
  - 13. systems returns ATM card

— Enter PIN

— Verify PIN

} Select Service

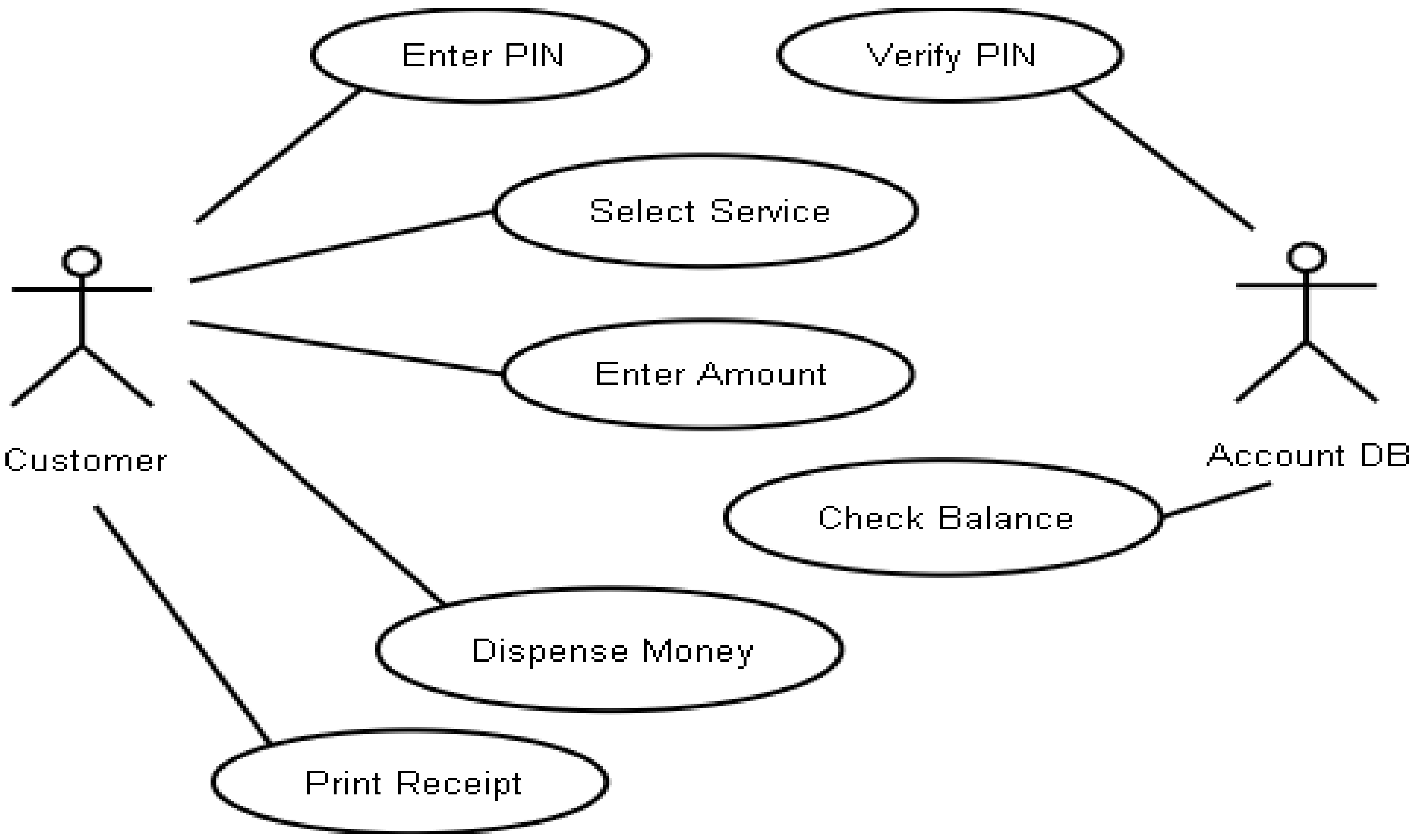
} Enter amount

— Check account balance

— Dispense money

} Print receipt

4.5 Modeling Notations – Data-Flow Diagrams



4.5 Modeling Notations – Functions and Relations

- **Formal methods** or **approach:** mathematically based specification and design techniques
- Formal methods model requirements or software behavior as a collection of mathematical **functions** or **relations**
  - Functions specify the state of the system's execution, and output
  - A relation is used whenever an input value maps more than one ouput value
- Functional method is consistent and complete

4.5 Modeling Notations – Functions and Relations

- Example: representing turnstile problem using two functions
  - One function to keep track of the state
  - One function to specify the turnstile output

$$\mathit{NetState}(s,e) = \begin{cases} \mathit{unlocked} & s=\mathit{locked} \text{ AND } e=\mathit{coin} \\ \mathit{rotating} & s=\mathit{unlocked} \text{ AND } e=\mathit{push} \\ \mathit{locked} & (s=\mathit{rotating} \text{ AND } e=\mathit{rotated}) \\ & \text{OR } (s=\mathit{locked} \text{ AND } e=\mathit{slug}) \end{cases}$$

$$\mathit{Output}(s,e) = \begin{cases} \mathit{buzz} & s=\mathit{locked} \text{ AND } e=\mathit{slug} \\ \langle \mathit{none} \rangle & \mathit{Otherwise} \end{cases}$$

4.5 Modeling Notations – Functions and Relations

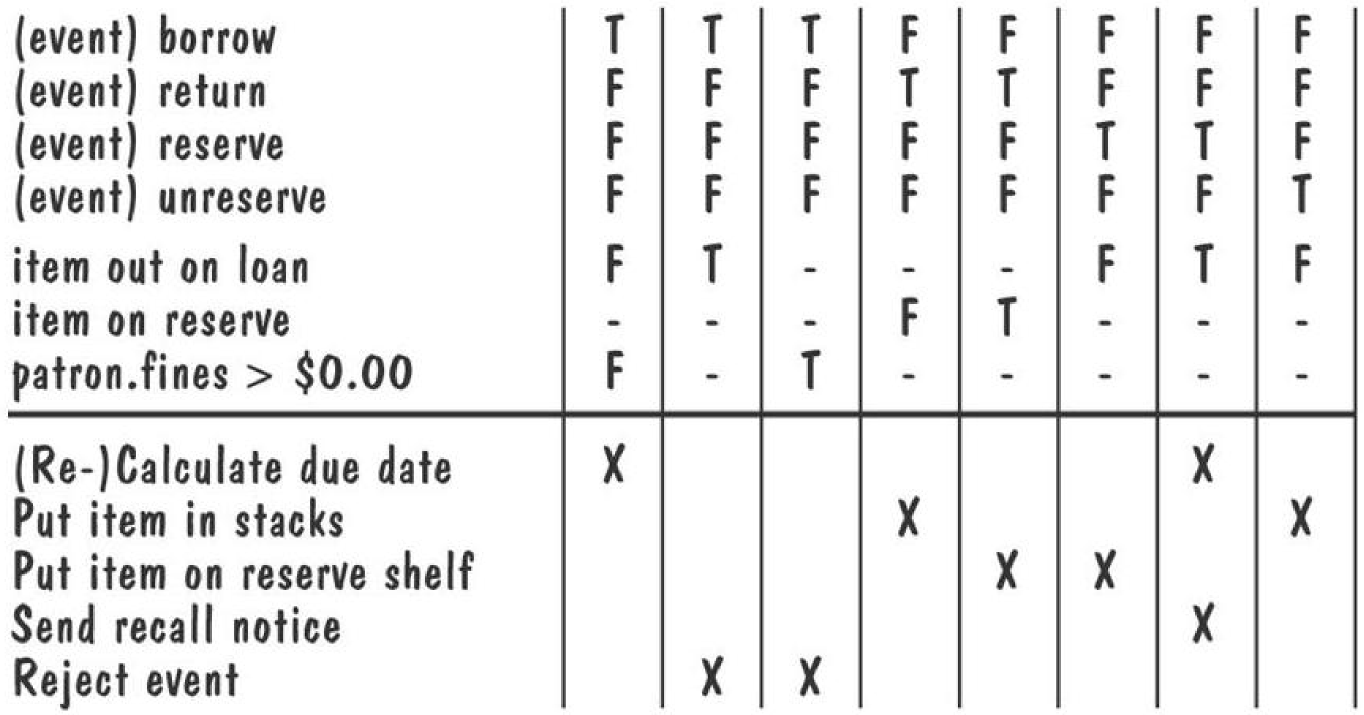
Example: Decision Tables

- It is a tabular representation of a functional specification that maps events and conditions to appropriate responses or action
- The specification is formal because the inputs (events and conditions) and outputs (actions) may be expressed in natural language
- If there is *n* input conditions, there are 2^n possible combinations of input conditions
- Combinations map to the same set of results and can be combined into a single column

4.5 Modeling Notations – Functions and Relations

Example: Decision Tables

- Decision table for library functions `borrow`, `return`, `reserve`, and `unreserve`.



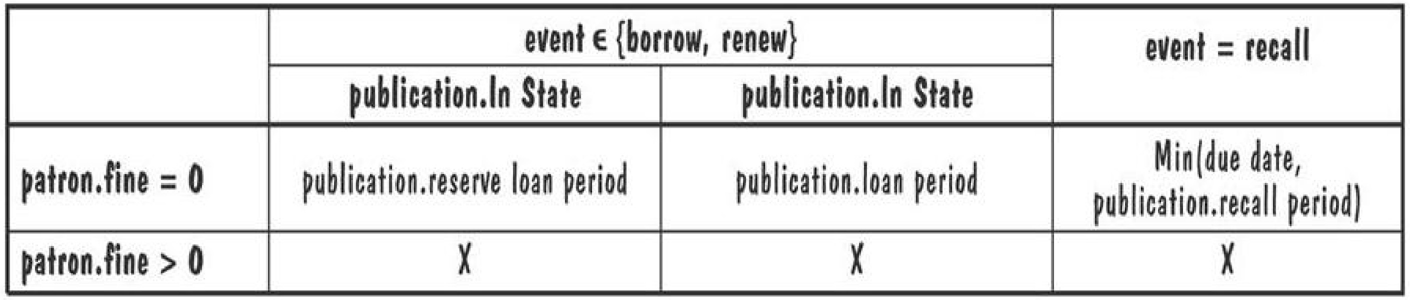
(event) borrow	T	T	T	F	F	F	F	F
(event) return	F	F	F	T	T	F	F	F
(event) reserve	F	F	F	F	F	T	T	F
(event) unreserve	F	F	F	F	F	F	F	T
item out on loan	F	T	-	-	-	F	T	F
item on reserve	-	-	-	F	T	-	-	-
patron.fines > $0.00	F	-	T	-	-	-	-	-
(Re-)Calculate due date	X						X	
Put item in stacks				X				X
Put item on reserve shelf					X	X		
Send recall notice							X	
Reject event		X	X					

4.5 Modeling Notations – Functions and Relations

Example: Parnas Tables

- Tabular representations of mathematical functions or relations
  - The column and row headers are predicates used to specify cases
  - The internal table entries store the possible function results
  - An entry “X” either could be invalid under the specified conditions or the combindation of conditions is infeasible

Calc due date(patron, publication, event, Today) =



	event ∈ {borrow, renew}		event = recall
	publication.In State	publication.In State	
patron.fine = 0	publication.reserve loan period	publication.loan period	Min(due date, publication.recall period)
patron.fine > 0	X	X	X

4.5 Modeling Notations – Logic

- An **operational notation** is a notation used to describe a problem or a proposed software solution in terms of situational behavior
  - Model of case-based behavior
  - Examples: state machine, event traces, data-flow diagram, functional method
- A **descriptive notation** is a notation that describes a problem or a proposed solution in terms of its properties or its variant
  - Example: logic

4.5 Modeling Notations – Logic

- A logic consists of a language for expressing properties and a set of inference rules for deriving new, consequent properties from the stated properties
- In logic, a property specification represents only those values of the property's variables for which the property's expression evaluates to true
- It is first-order logic, comprising typed variables, constants, functions, and predicates

4.5 Modeling Notations – Logic

- Consider the following variables of the turnstile problem, with their initial value

```
num_coins : integer := 0;                /* number of coins inserted */
num_entries : integer := 0;              /* number of half-rotations of
                                         turnstile */
barrier :{locked, unlocked}:= locked;/* whether barrier is locked         */
may_enter : boolean   := false;          /* event of coin being inserted    */
push  : boolean    := false;             /* turnstile is pushed sufficiently
                                       hard to rotate it one-half rotation*/
```

- The first-order logic expressions

$num\_coins \geq num\_entries$

$(num\_coins \geq num\_entries \Leftrightarrow (barrier = unlocked)$

$(barrier = locked) \Leftrightarrow \neg may\_enter$

4.5 Modeling Notations – Logic

- Temporal logic introduces additional logical connectives for constraining how variables can change value over time
- The following connectives constrain future variable values, over a single execution
  - □f Ξ f is *true* now and throughout the rest of execution
  - □ f Ξ f is *true* now or at some future point in the execution
  - ○f Ξ f is *true* in the next point in the execution
  - f W g = f is *true* until a point where g is true, but g may never be true
- Turnstile properties expressed in temporal logic

  □(insert_coin => ○ (may_enter W push))

  □(□ n(insert_coin ∧ num_coins=n) => ○(num_coins=n+1))

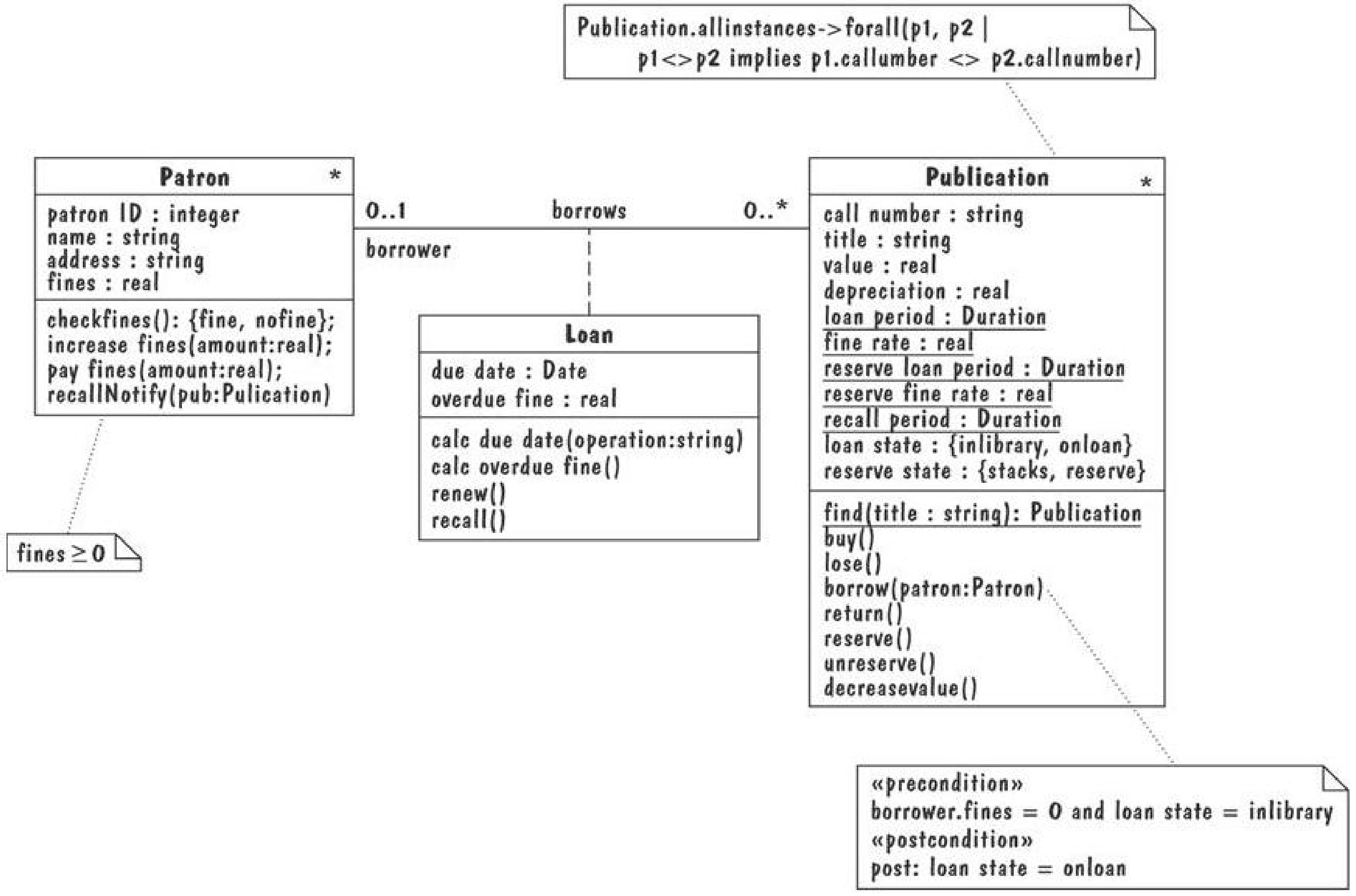
4.5 Modeling Notations – Logic

Example: Object Constrain Language (OCL)

- A constrain language that is both mathematically precise and easy for non-mathematicians to read, write, and understand
- Designed for expressing constrains on object models, and expressing queries on object type

4.5 Modeling Notations – Logic

Example: Object Constrain Language (OCL)

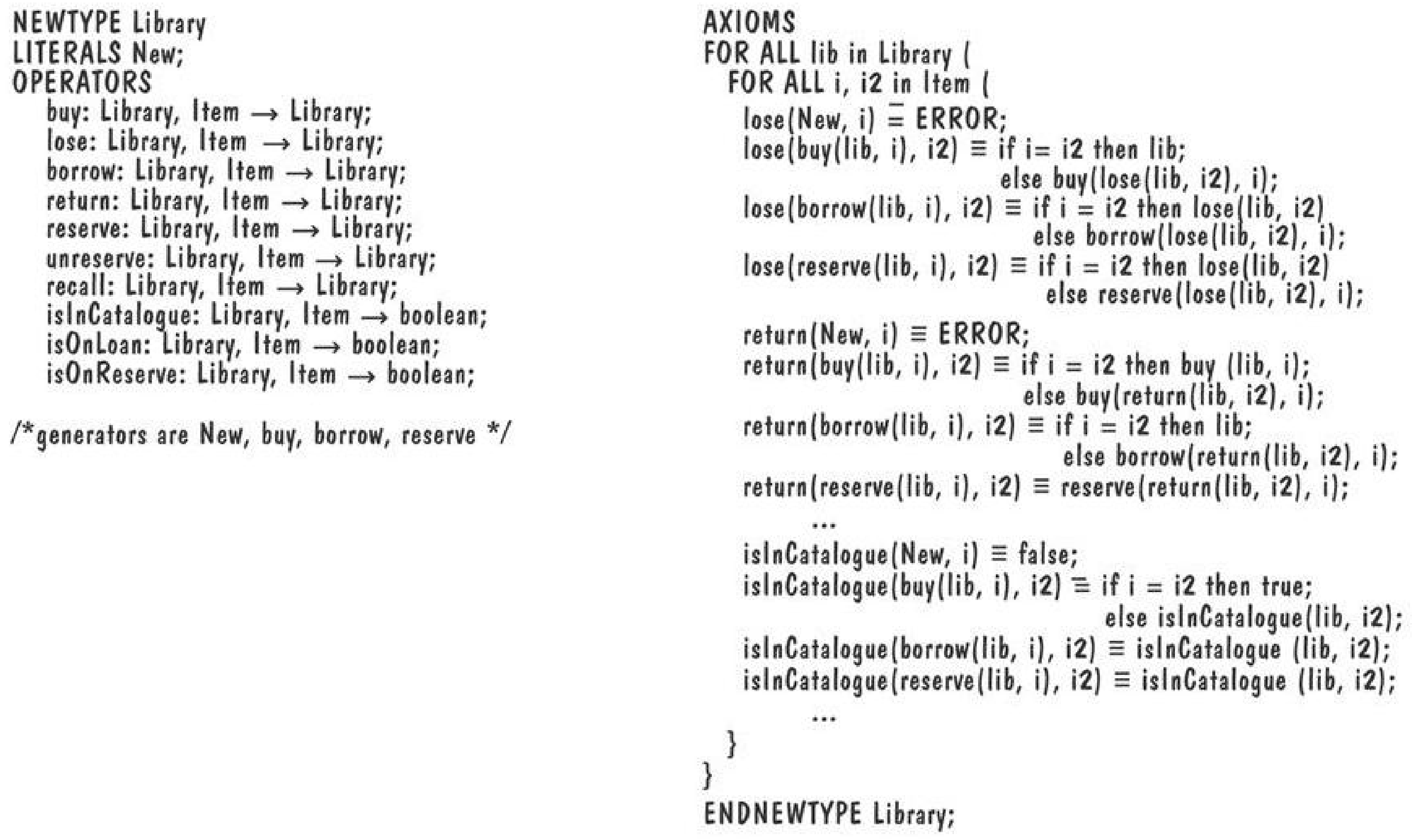


4.5 Modeling Notations – Algebraic Specifications

- To specify the behavior of operations by specifying interactions between pairs of operations rather than modeling individual operations
- It is hard to define a set of axioms that is complete and consistent and that reflects the desired behavior

4.5 Modeling Notations – Algebraic Specifications

- Partial SDL data specification for the library problem



```
NEWTYPE Library
LITERALS New;
OPERATORS
    buy: Library, Item → Library;
    lose: Library, Item → Library;
    borrow: Library, Item → Library;
    return: Library, Item → Library;
    reserve: Library, Item → Library;
    unreserve: Library, Item → Library;
    recall: Library, Item → Library;
    isInCatalogue: Library, Item → boolean;
    isOnLoan: Library, Item → boolean;
    isOnReserve: Library, Item → boolean;

/*generators are New, buy, borrow, reserve */
```

```
AXIOMS
FOR ALL lib in Library (
  FOR ALL i, i2 in Item (
    lose(New, i) ≡ ERROR;
    lose(buy(lib, i), i2) ≡ if i= i2 then lib;
                              else buy(lose(lib, i2), i);
    lose(borrow(lib, i), i2) ≡ if i = i2 then lose(lib, i2)
                                 else borrow(lose(lib, i2), i);
    lose(reserve(lib, i), i2) ≡ if i = i2 then lose(lib, i2)
                                  else reserve(lose(lib, i2), i);
    return(New, i) ≡ ERROR;
    return(buy(lib, i), i2) ≡ if i = i2 then buy (lib, i);
                                else buy(return(lib, i2), i);
    return(borrow(lib, i), i2) ≡ if i = i2 then lib;
                                   else borrow(return(lib, i2), i);
    return(reserve(lib, i), i2) ≡ reserve(return(lib, i2), i);
          ...
    isInCatalogue(New, i) ≡ false;
    isInCatalogue(buy(lib, i), i2) ≡ if i = i2 then true;
                                       else isInCatalogue(lib, i2);
    isInCatalogue(borrow(lib, i), i2) ≡ isInCatalogue (lib, i2);
    isInCatalogue(reserve(lib, i), i2) ≡ isInCatalogue (lib, i2);
          ...
  }
}
ENDNEWTYPE Library;
```

4.6 Requirements and Specification Languages

Unified Modeling Language (UML)

- Combines multiple notation paradigms
- Eight graphical modeling notations, and the OCL constrain language, including
  - Use-case diagram (a high-level DFD)
  - Class diagram (an ER diagram)
  - Sequence diagram (an event trace)
  - Collaboration diagram (an event trace)
  - Statechart diagram (a state-machine model)
  - OCL properties (logic)

4.6 Requirements and Specification Languages

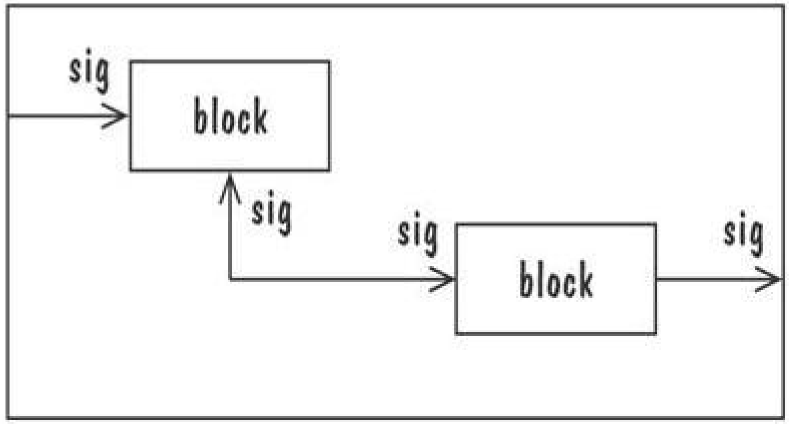
Specification and Description Language (SDL)

- Standardized by the International Telecommunication Union
- Specifies the behavior of real-time, concurrent, distributed processes that communicate with each other via unbounded message queues
- Comprises
  - SDL system diagram (a DFD)
  - SDL block diagram (a DFD)
  - SDL process diagram (a state-machine model)
  - SDL data type (algebraic specification)
- Often accompanied by a set of Message Sequence Chart (MSC)

4.6 Requirements and Specification Languages

SDL System Diagram

- The top-level blocks of the specification and communication channels that connect the blocks
- Channels are directional and are labelled with the type of signals
- Message is asynchronous

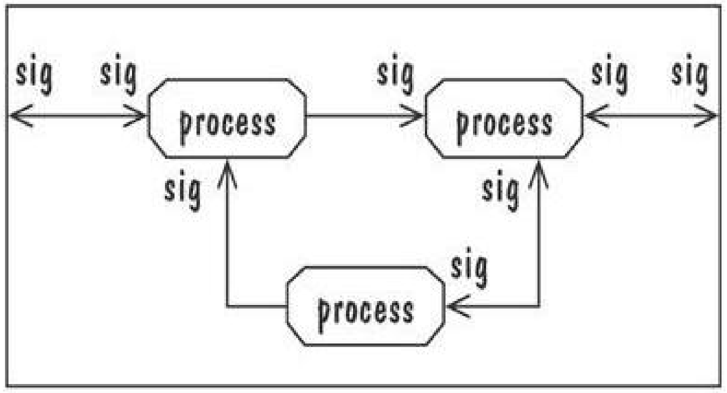


(a) An SDL system or block of blocks

4.6 Requirements and Specification Languages

SDL Block Diagram

- SDL block diagram models a lower-level collection of blocks and the message-delaying channels that interconnect them
- Figure depicts a collection of lowest-level processes that communicate via signal routes
- Signal routes pass messages synchronously

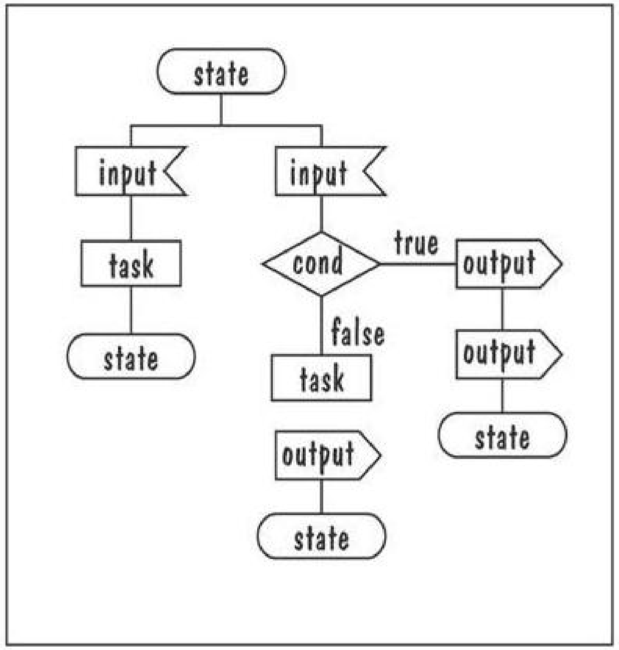


(b) An SDL block of processes

4.6 Requirements and Specification Languages

SDL Process Diagram

- It is a state-machine whose transitions are sequences of language constructs (input, decisions, tasks, outputs) that start and end at state constructs



(c) An SDL process

4.6 Requirements and Specification Languages

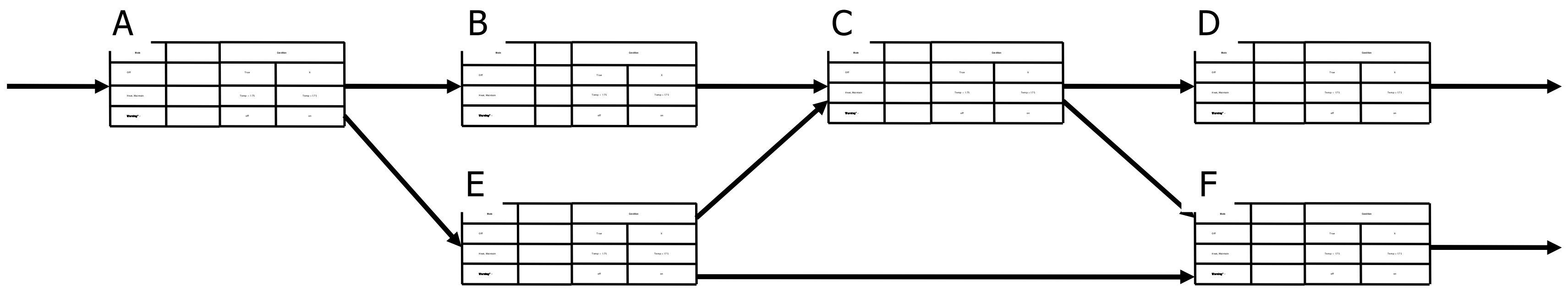
Software Cost Reduction (SCR)

- Collection of techniques that were designed to encourage software developers to employ good software-engineering design principles
- Models software requirements as a mathematical function, REQ, that maps monitored variables to controlled variables
  - monitored variables: environmental variables that are sensed by the system
  - controlled variables: environmental variables that are set by the system
- The function REQ is decomposed into a collection of tabular functions

4.6 Requirements and Specification Languages

Software Cost Reduction (SCR)

- REQ is the result of composing the tabular functions into network (a DFD) as shown in the picture.
- Edges reflect the data dependecies among the functions
- Execution steps start with a change in the value of one monitored variable, then propagate through the network, in a single syncronized step



4.6 Requirements and Specification Languages

Other Features of Requirement Notations

- Some techniques include notations
  - for the degree of uncertainty or risk with each requirement
  - for tracing requirements to other system documents such as design or code, or to other systems, such as when requirements are reused
- Most specification techniques have been automated to some degree

4.7 Prototyping requirements

- Throw-away prototypes
- Evolutionary prototypes
- Rapid prototypes

4.7 Prototyping requirements

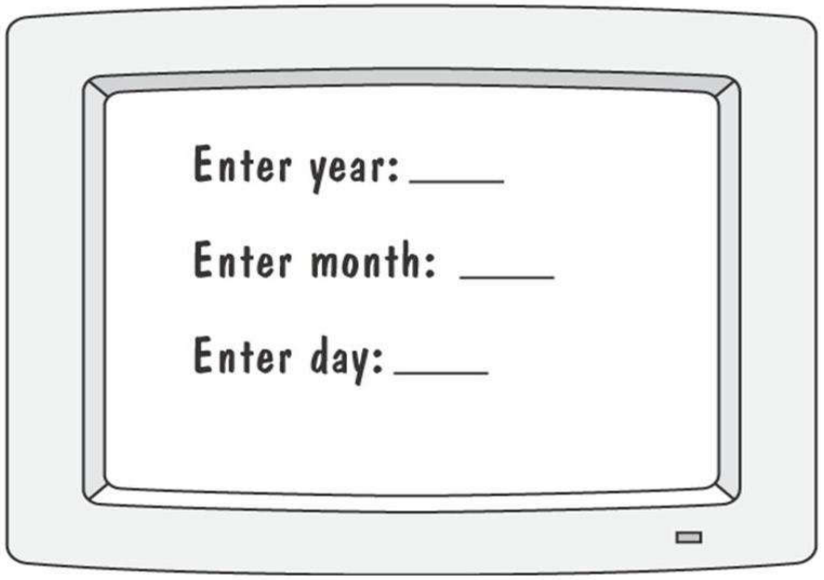
Building a Prototype

- To elicit the details of proposed system
- To solicit feedback from potential users about
  - which aspects they would like to see improve
  - which features are not so useful
  - what functionality is missing
- Determine whether the customer's problem has a feasible solution
- Assist in exploring options for optimizing quality requirements

4.7 Prototyping Requirements

Prototyping Example

- Prototype for building a tool to track how much a user exercises each day
- Graphical respresentation of first prototype, in which the user must type the day, month and year



Enter year: ____

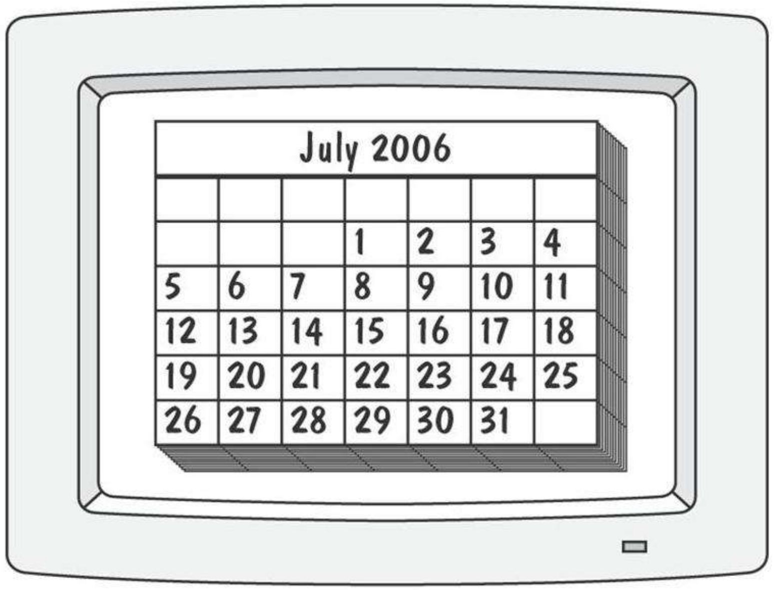
Enter month: ____

Enter day: ____

4.7 Prototyping Requirements

Prototyping Example

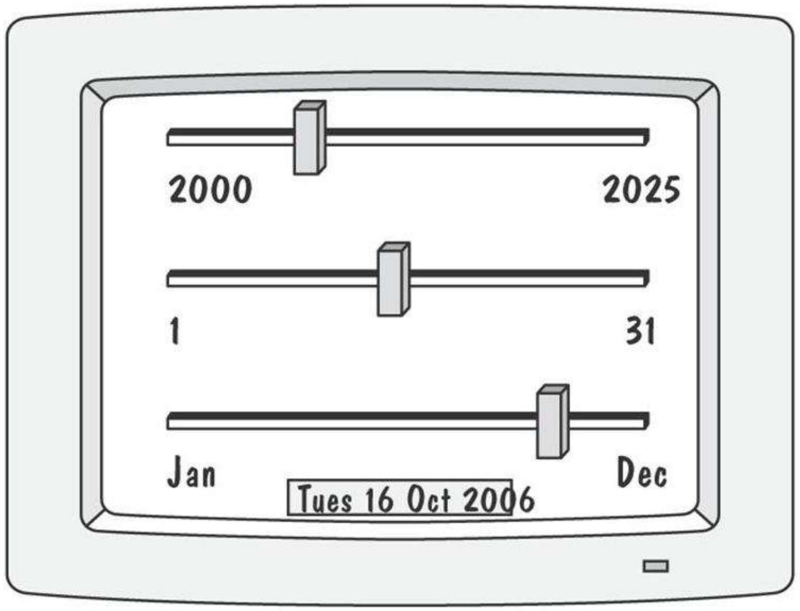
- Second prototype shows a more interesting and sophisticated interface involving a calendar
  - User uses a mouse to select the month and year
  - The system displays the chart for that month, and the user selects the appropriate date in the chart



4.7 Prototyping Requirements

Prototyping Example

- Third prototype shows that instead of a calendar, the user is presented with three slide bars
  - User uses the mouse to slide each bar left or right
  - The box at the bottom of the screen changes to show the selected day, month, and year



4.7 Prototyping Requirements

Prototyping vs. Modeling

- Prototyping
  - Good for answering questions about the user interfaces
- Modeling
  - Quickly answer questions about constraints on the order in which events should occur, or about the synchronization of activities

4.8 Requirements documentation

- Requirements definition document: what the customer wants
  - general purpose
  - background and objectives of system
  - description of customer-suggested approach
  - detailed characteristics
  - operational environment
- Requirements specification document: what the designers need to know

4.8 Requirements documentation

How developers see users	*How users see developers*
Users don’t know what they want.	Developers don’t understand operational needs.
Users can’t articulate what they want.	Developers place too much emphasis on technicalities.
Users have too many needs that are politically motivated.	Developers try to tell us how to do our jobs.
Users want everything right now.	Developers can’t translate clearly-stated needs into a successful system.
Users can’t prioritize needs.	Developers say no all the time.
Users refuse to take responsibility for the system.	Developers are always over budget.
Users are unable to provide a usable statement of needs.	Developers are always late.
Users are not committed to system development projects.	Developers ask users for time and effort, even to the detriment of the users’ important primary duties.
Users are unwilling to compromise.	Developers set unrealistic standards for requirements definition.
Users can’t remain on schedule.	Developers are unable to respond quickly to legitimately changing needs.

4.8 Requirements documentation

Requirements Definition: Steps Documenting Process

- Outline the general purpose and scope of the system, including relevant benefits, objectives, and goals
- Describe the background and the rationale behind proposal for new system
- Describe the essential characteristics of an acceptable solution
- Describe the environment in which the system will operate
- Outline a description of the proposal, if the customer has a proposal for solving the problem
- List any assumptions we make about how the environment behaves

4.8 Requirements documentation

Requirements Definition: Steps Documenting Process

- Describe all inputs and outputs in detail, including
  - the sources of inputs
  - the destinations of outputs,
  - the value ranges
  - data format of inputs and outputs data
  - data protocols
  - window formats and organizations
  - timing constraint
- Restate the required functionality in terms of the interfaces' inputs and outputs
- Devise fit criteria for each of the customer's quality requirements

4.8 Requirements documentation

IEEE Standard for SRS Organized by Objects

1. Introduction to the Document
    - 1.1 Purpose of the Product
    - 1.2 Scope of the Product
    - 1.3 Acronyms, Abbreviations, Definitions
    - 1.4 References
    - 1.5 Outline of the rest of the SRS
2. General Description of Product
    - 2.1 Context of Product
    - 2.2 Product Functions
    - 2.3 User Characteristics
    - 2.4 Constraints
    - 2.5 Assumptions and Dependencies
3. Specific Requirements
    - 3.1 External Interface Requirements
        - 3.1.1 User Interfaces
        - 3.1.2 Hardware Interfaces
        - 3.1.3 Software Interfaces
        - 3.1.4 Communications Interfaces
    - 3.2 Functional Requirements
        - 3.2.1 Class 1
        - 3.2.2 Class 2
        - …
    - 3.3 Performance Requirements
    - 3.4 Design Constraints
    - 3.5 Quality Requirements
    - 3.6 Other Requirements
4. Appendices

4.8 Requirements documentation

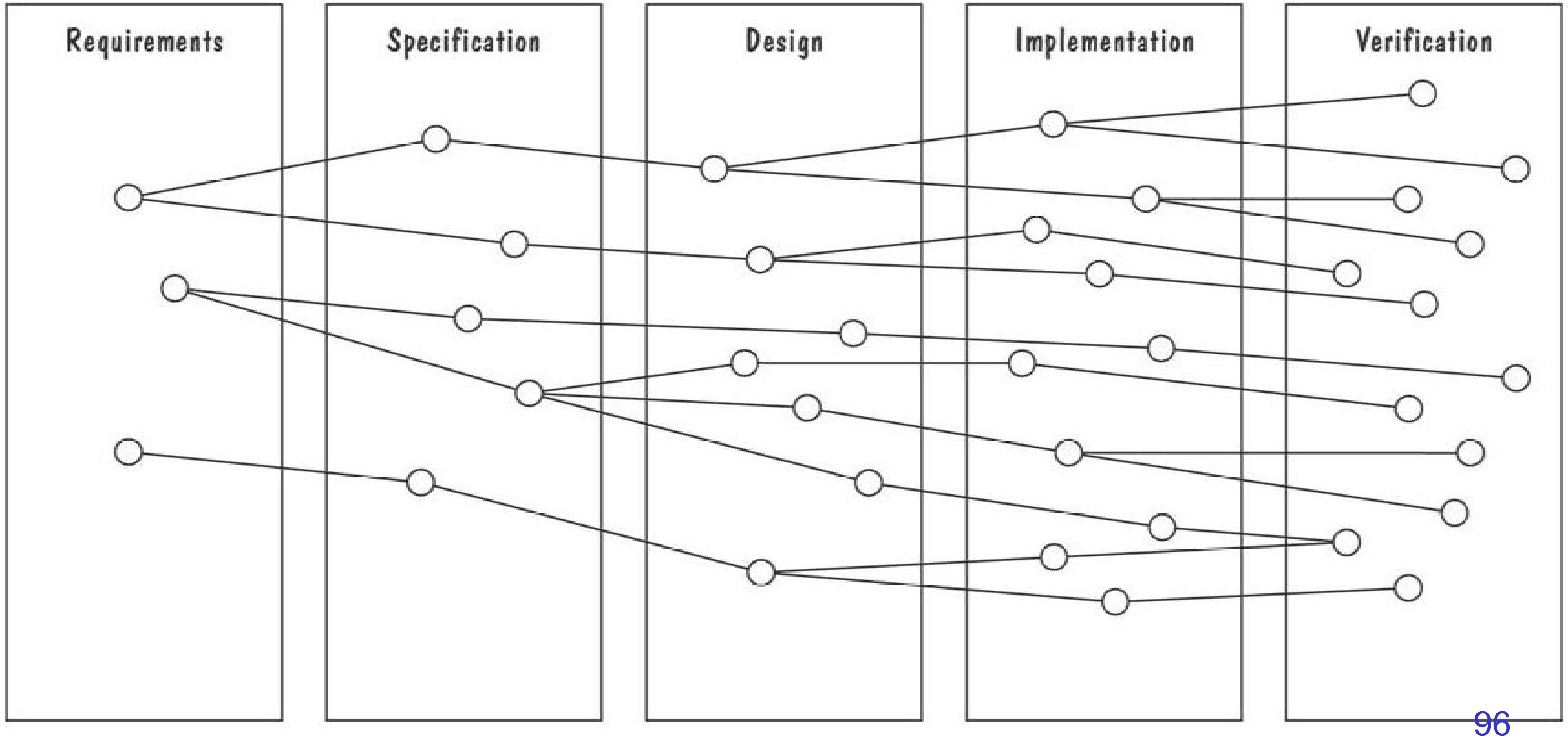
Process Management and Requirements Traceability

- Process management is a set of procedures that track
  - the requirements that define what the system should do
  - the design modules that are generated from the requirement
  - the program code that implements the design
  - the tests that verify the functionality of the system
  - the documents that describe the system
- It provides the threads that tie the system parts together

4.8 Requirements documentation

Development Activities

- Horizontal threads show the coordination between development activities



4.9 Validation and Verification

- In requirements validation, we check that our requirements definition accurately reflects the customer's needs
- In verification, we check that one document or artifact conforms to another
- Verification ensures that we build the system right, whereas validation ensures that we build the right system

4.9 Validation and Verification

Validation	Walkthroughs
	Reading
	Interviews
	Reviews
	Checklists
	Models to check functions and relationships
	Scenarios
	Prototypes
	Simulation
	Formal inspections
Verification	Cross-referencing
	Simulation
	Consistency checks
	Completeness checks
Checking	Check for unreachable states or transitions
	Model checking
	Mathematical proofs

4.9 Validation and Verification

Requirements Review

- Review the stated goals and objectives of the system
- Compare the requirements with the goals and objectives
- Review the environment in which the system is to operate
- Review the information flow and proposed functions
- Assess and document the risk, discuss and compare alternatives
- Testing the system: how the requirements will be revalidated as the requirements grow and change

4.9 Validation and Verification

Requirements Review

- Jone and Thayes's studies show that
  - 35% of the faults to design activities for projects of 30,000-35,000 delivered source instructions
  - 10% of the faults to requirements activities and 55% of the faults to design activities for projects of 40,000-80,000 delivered source instructions
  - 8% to 10% of the faults to requirements activities and 40% to 55% of the faults to design activities for projects of 65,000-85,000 delivered source instructions
- Basili and Perricone report
  - 48% of the faults observed in a medium-scale software project were attributed to “incorrect or misinterpreted functional specification or requirements”
- Beizer attributes 8.12% of the faults in his samples to problems in functional requirements

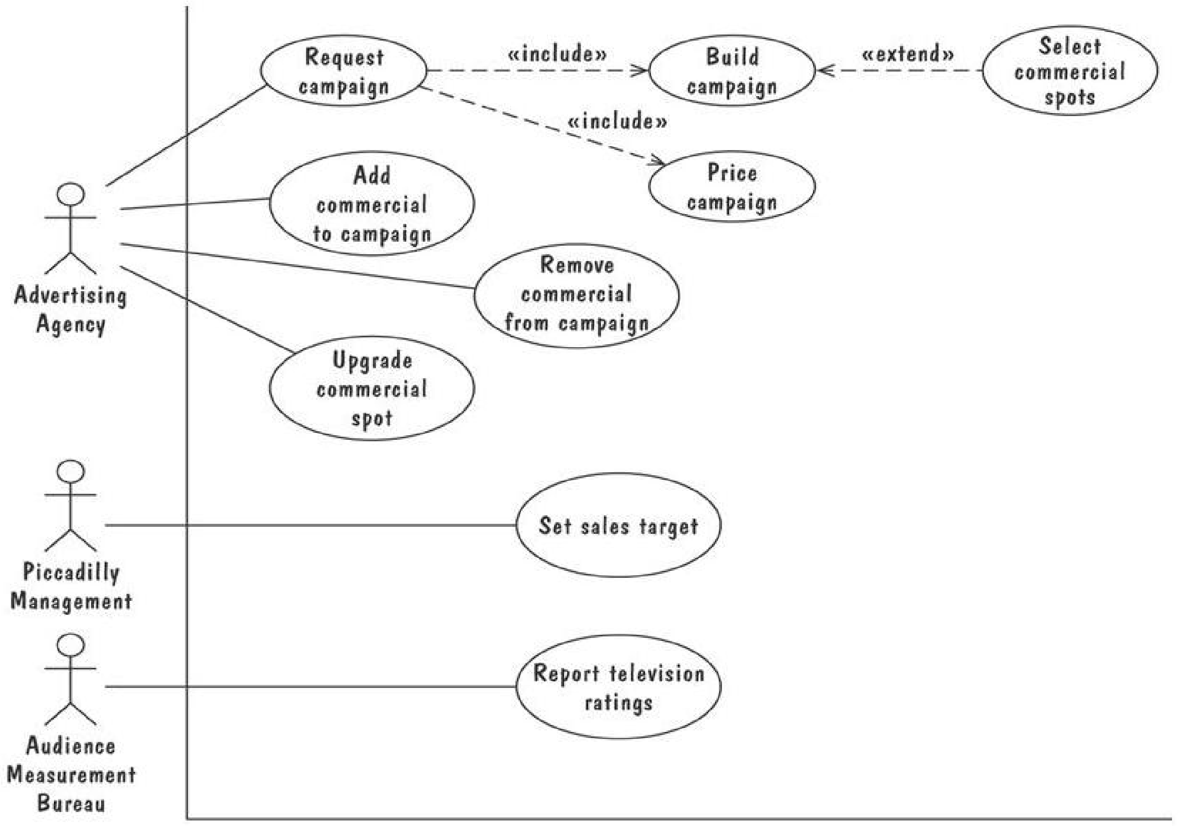
4.9 Validation and Verification

Verification

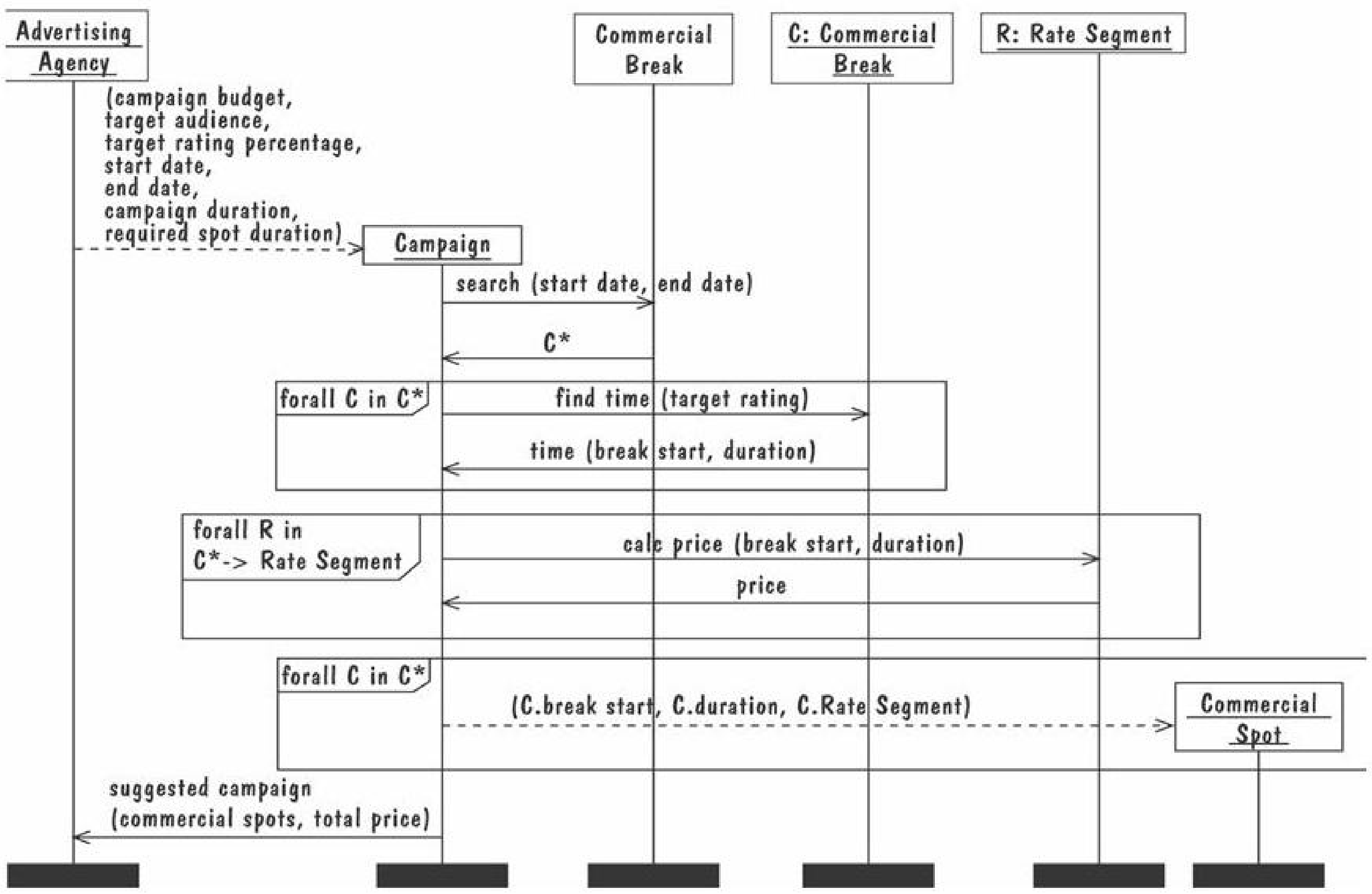
- Check that the requirements-specification document corresponds to the requirements-definition
- Make sure that if we implement a system that meets the specification, then the system will satisfy the customer's requirements
- Ensure that each requirement in the definition document is traceable to the specification

4.10 Information System Example

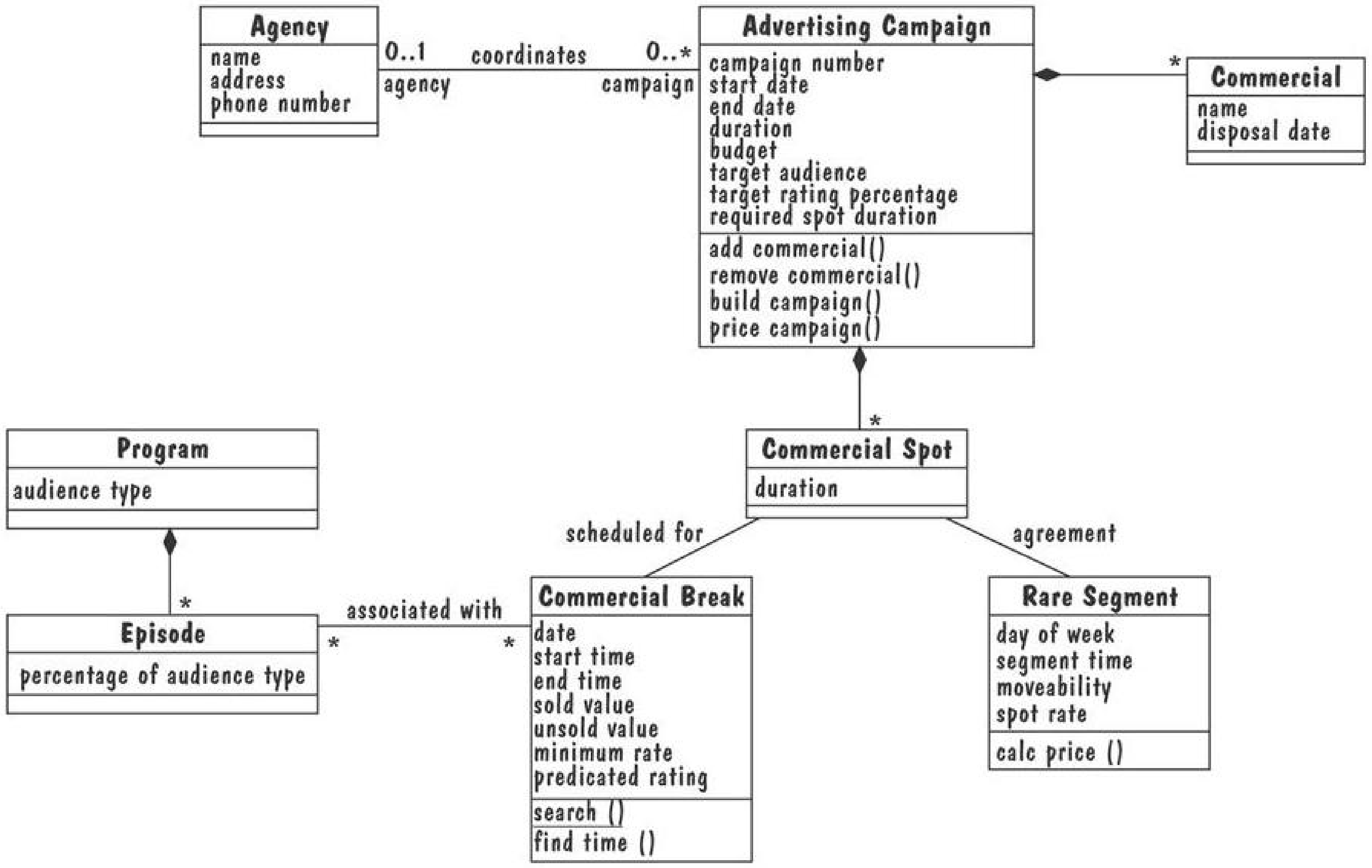
- High-level diagram captures the essential functionality
  - Shows nothing about the ways in which each of these use cases might succeed or fail



4.10 Information System Example


Advertising Agency
Commercial Break
C: Commercial Break
R: Rate Segment
(campaign budget, target audience, target rating percentage, start date, end date, campaign duration, required spot duration)
Campaign
search (start date, end date)
C*
forall C in C*
find time (target rating)
time (break start, duration)
forall R in C*-> Rate Segment
calc price (break start, duration)
price
forall C in C*
(C.break start, C.duration, C.Rate Segment)
Commercial Spot
suggested campaign (commercial spots, total price)

4.10 Information System Example


Agency
name
address
phone number
0..1
agency
coordinates
0..*
campaign
Advertising Campaign
campaign number
start date
end date
duration
budget
target audience
target rating percentage
required spot duration
add commercial()
remove commercial()
build campaign()
price campaign()
*
Commercial
name
disposal date
*
Commercial Spot
duration
scheduled for
agreement
Program
audience type
*
Episode
percentage of audience type
associated with
*
*
Commercial Break
date
start time
end time
sold value
unsold value
minimum rate
predicated rating
search ()
find time ()
Rare Segment
day of week
segment time
moveability
spot rate
calc price ()