中国科学技术大学软件学院
SCHOOL OF SOFTWARE ENGINEERING OF USTC

Capturing the Requirements

中国科学技术大学软件学院
孟宁

Contents

Chapter 4 Objectives

- Eliciting requirements from the customers
- Modeling requirements
- Reviewing requirements to ensure their quality
- Documenting requirements for use by the design and test teams

4.1 The Requirements Process

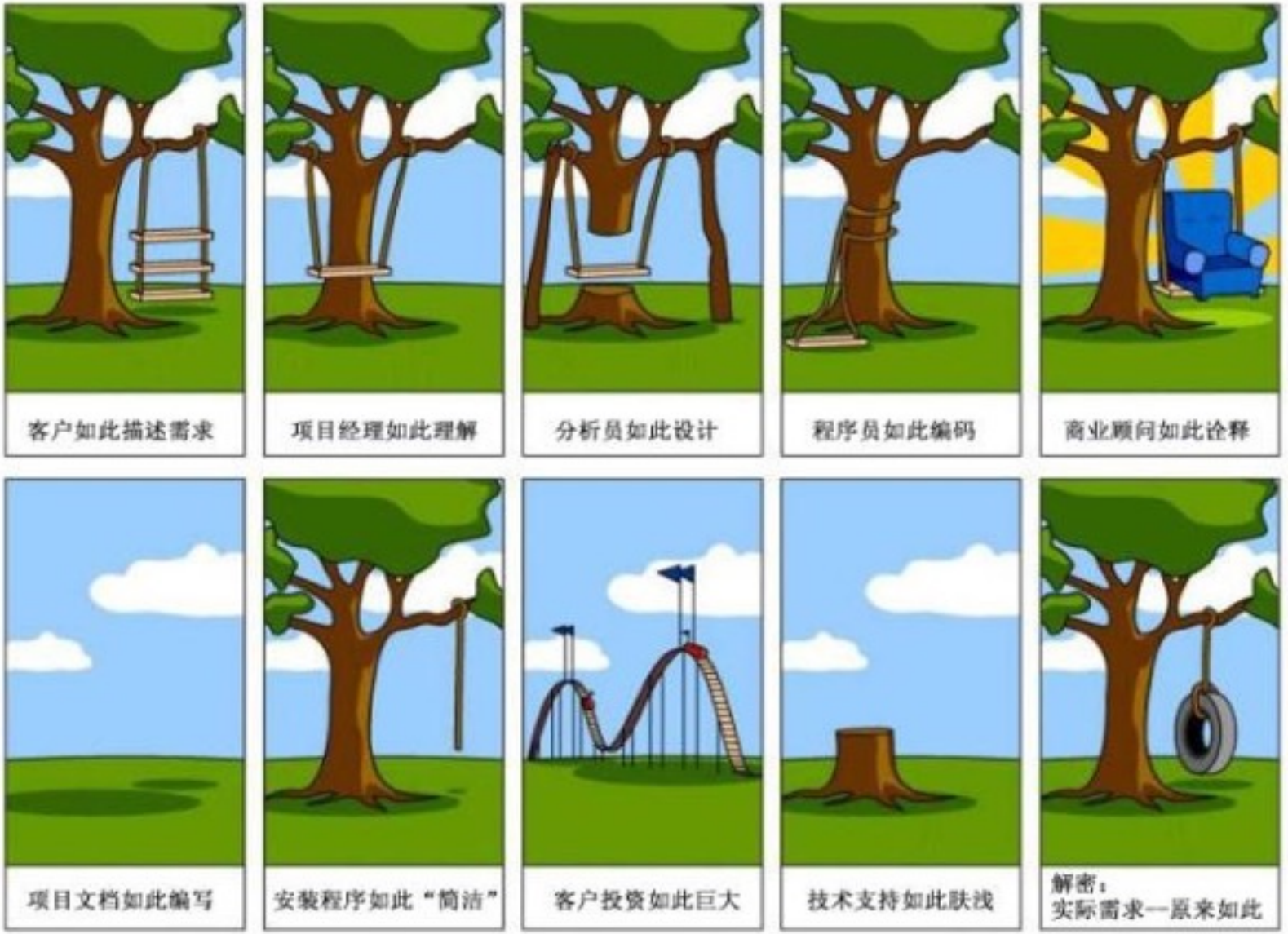
- A ***requirement*** is an expression of desired behavior
- A requirement deals with
  - objects or entities
  - the state they can be in
  - functions that are performed to change states or object characteristics
- Requirements focus on the customer needs, not on the solution or implementation
  -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

4.1 The Requirements Process

Sidebar 4.1 Why Are Requirements Important?

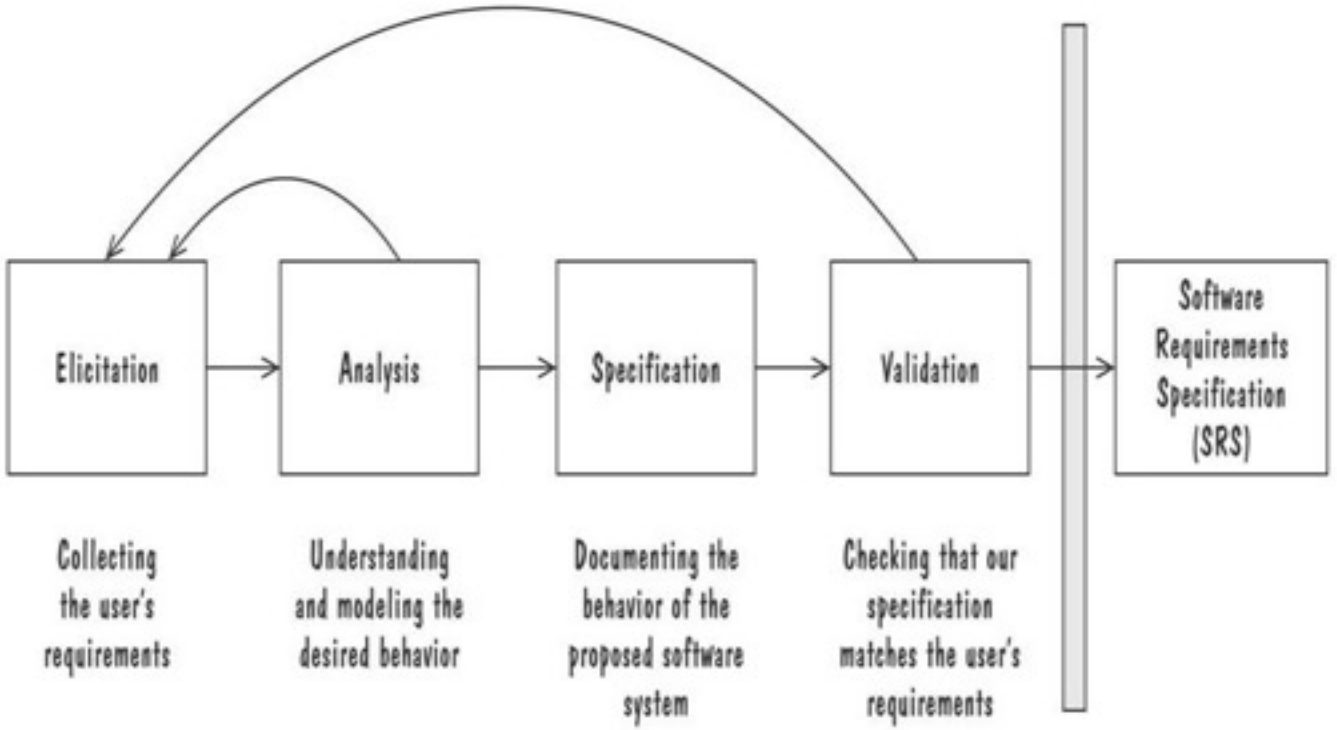
- Top factors that caused project to fail
  - Incomplete requirements
  - Lack of user involvement
  - Unrealistic expectations
  - Lack of executive support
  - Changing requirements and specifications
  - Lack of planning
  - System no longer needed
- Some part of the requirements process is involved in almost all of these causes
- Requirements error can be expensive if not detected early

需求获取的主要困难


客户如此描述需求
项目经理如此理解
分析员如此设计
程序员如此编码
商业顾问如此诠释
项目文档如此编写
安装程序如此“简洁”
客户投资如此巨大
技术支持如此肤浅
解密：
实际需求—原来如此

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



4.1 The Requirements Process

Sidebar 4.2 Agile Requirements Modeling

- If requirements are tightly coupled and complex, we may be better off with a “heavy” process that empasizes up-front modeling
- If the requirements are uncertain, agile methods are an alternative approach
- Agile methods gather and implement the requirements in increments
- Extreme Programming (XP) is an agile process
  - The requirements are defined as we build the system
  - No planning or designing for possible future requirements
  - Encodes the requirements as test cases that eventually implementation must pass

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

Stakeholders

- **Clients**: pay for the software to be developed
- **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

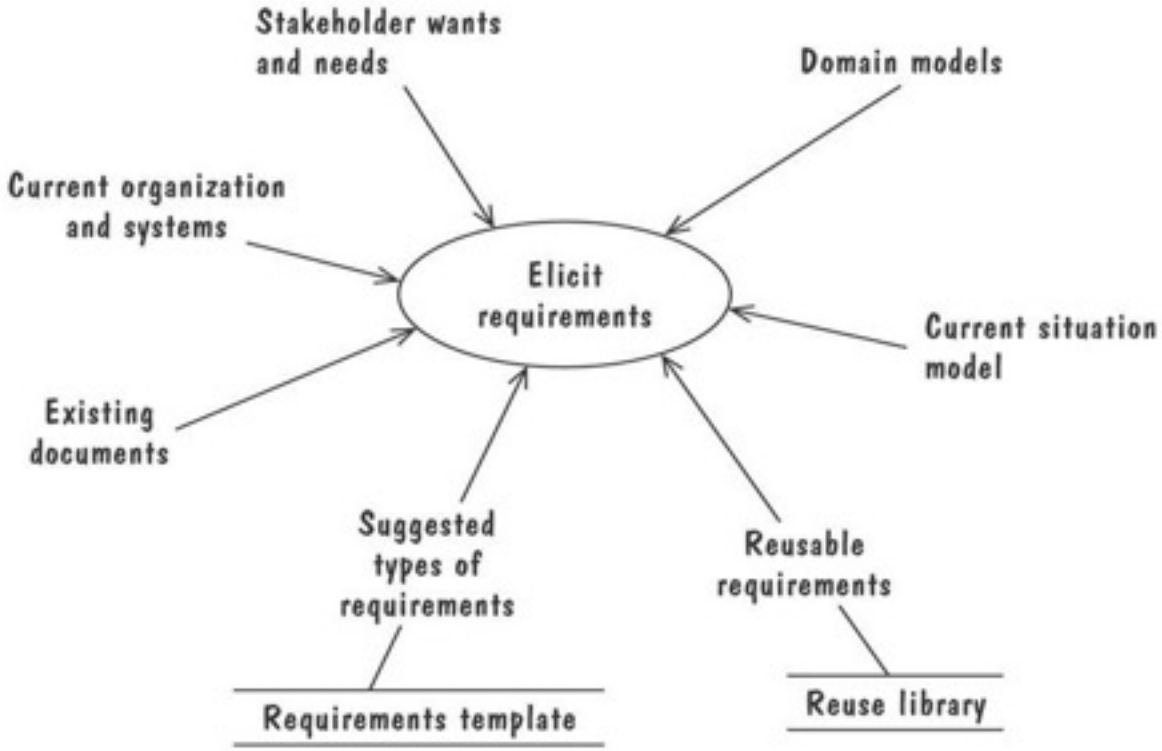
Means of Eliciting Requirements

- 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 user or stakeholders in groups
- Using domain specific strategies, such as Joint Application Design
- Brainstorming with current and potential users

4.2 Requirements Elicitation

Means of Eliciting Requirements (continued)

- 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 posses
- **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

4.3 Types of Requirements

Sidebar 4.4 Making Requirements Testable

- Fit criteria form objective standards for judging whether a proposed solution satisfies the requirements
  - It is easy to set fit criteria for quantifyable requirements
  - It is hard for subjective quality requirements
- Three ways to help make requirements testable
  - Specify a quantitave 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

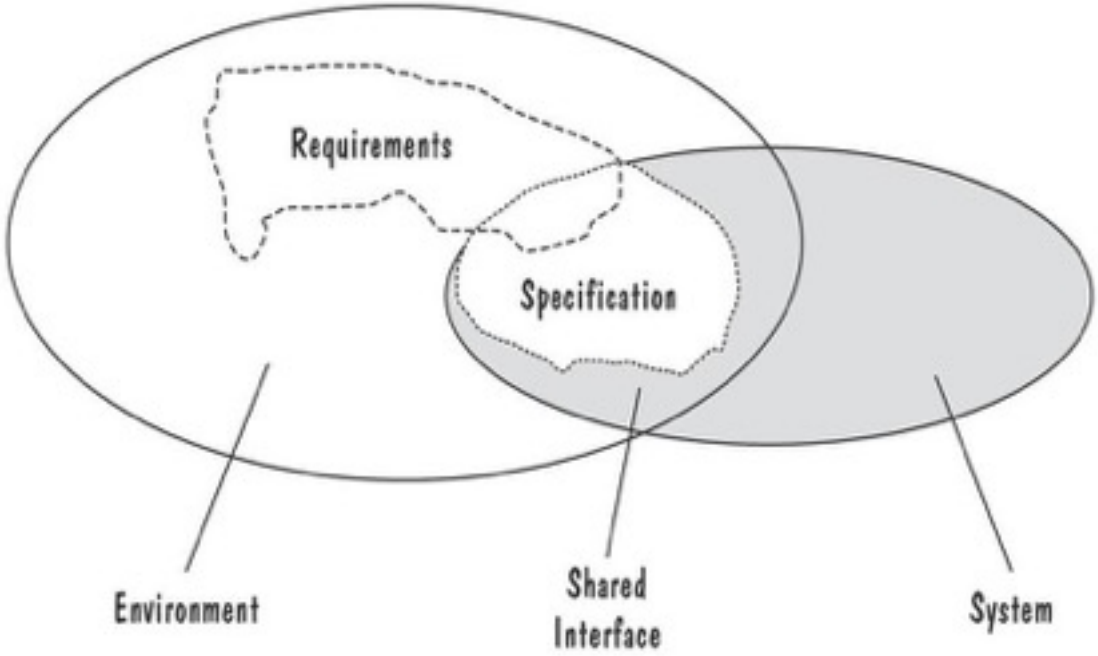
Two Kinds of Requirements Documents

- **Requirements definition**: a complete listing of everything the customer wants to achieve
  - Describing the entities in the environment where the system will be installed
- **Requirements specification**: restates the requirements as a specification of how the proposed system shall behave

4.3 Types of Requirements

Two Kinds of Requirements Documents (continued)

- 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

- Correct
- Consistent
- Unambigious无二义性
- Complete
- Feasible
- Relevant无与主要目标不相关的需求
- Testable
- Traceable

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

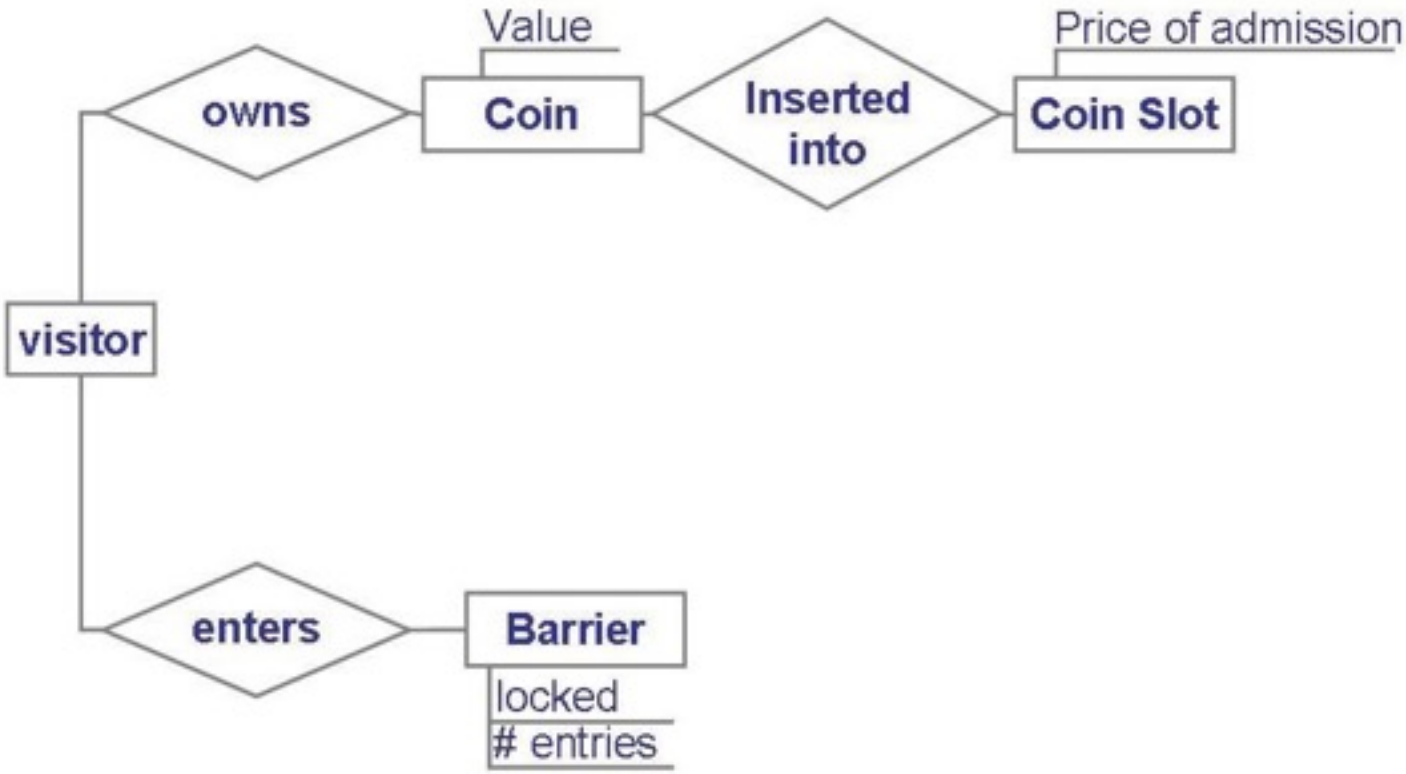
Entity-Relationship Diagrams

- A popular graphical notational paradigm for representing conceptual models
- 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

Entity-Relationship Diagrams (continued)

- Entity diagram of turnstile problem



4.5 Modeling Notations

Entity-Relationship Diagrams (continued)

- 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
- ER diagram is likely to be used to model a problem early in the requirements process

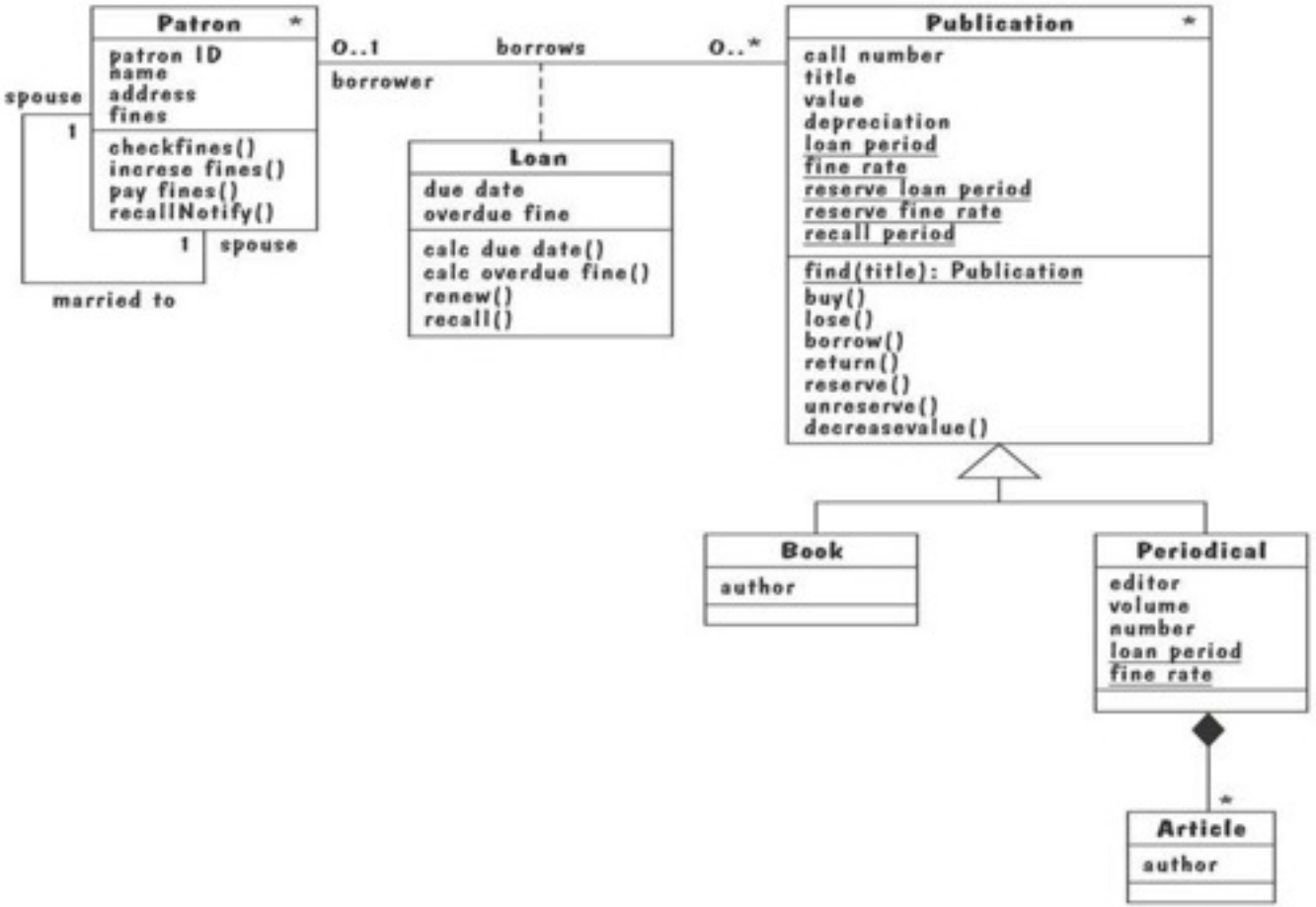
4.5 Modeling Notations

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

UML Class Diagram of Library Problem



4.5 Modeling Notations

UML Class Diagram (continued)

- 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

UML Class Diagram (continued)

- **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

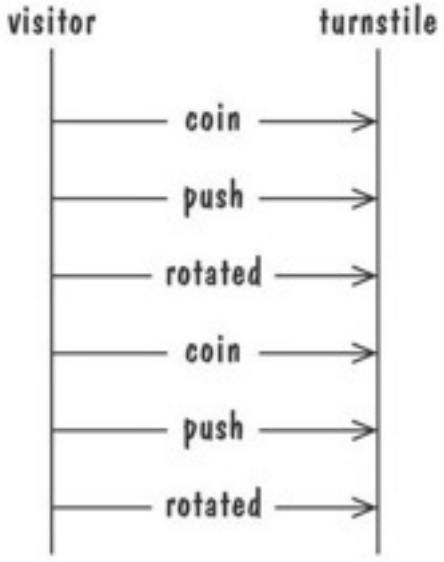
Event Traces

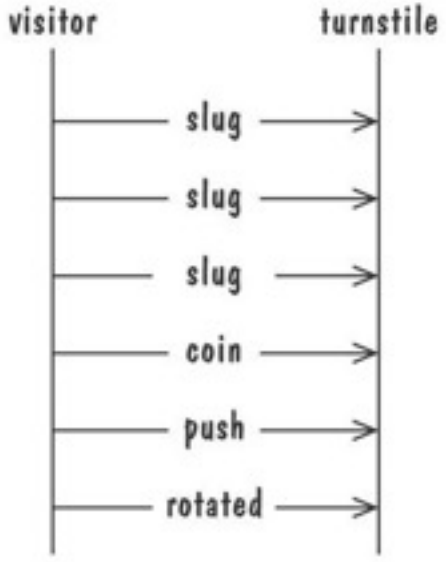
- A graphical description of a sequence of events that are exchanged between real-world entities
  - *Vertical line*: the timeline of distinct entity, whose name appear at the top of the line
  - *Horizontal line*: an event or interaction between the two entities bounding the line
  - Time progresses 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 (continued)

- 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

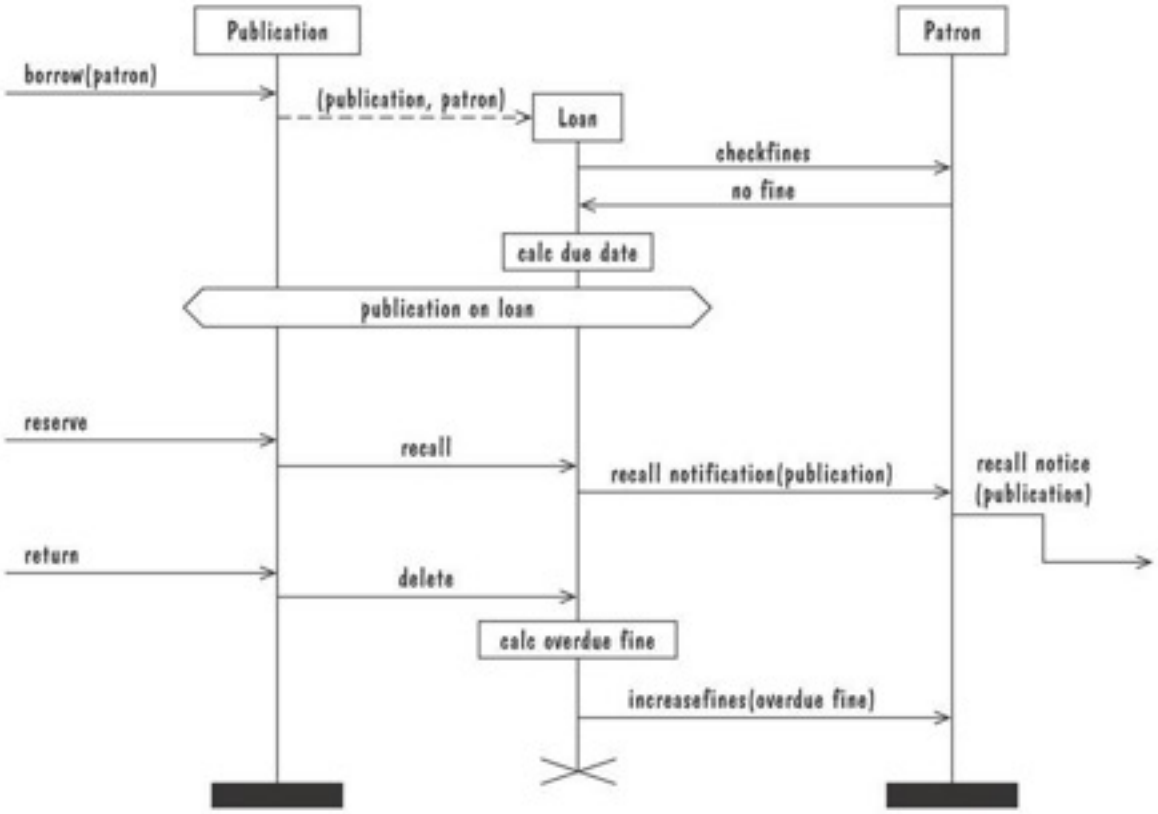
Event Traces Exampe: Message Sequence Chart

- 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

Message Sequence Chart (continued)

- Message sequence chart for library loan transaction



4.5 Modeling Notations

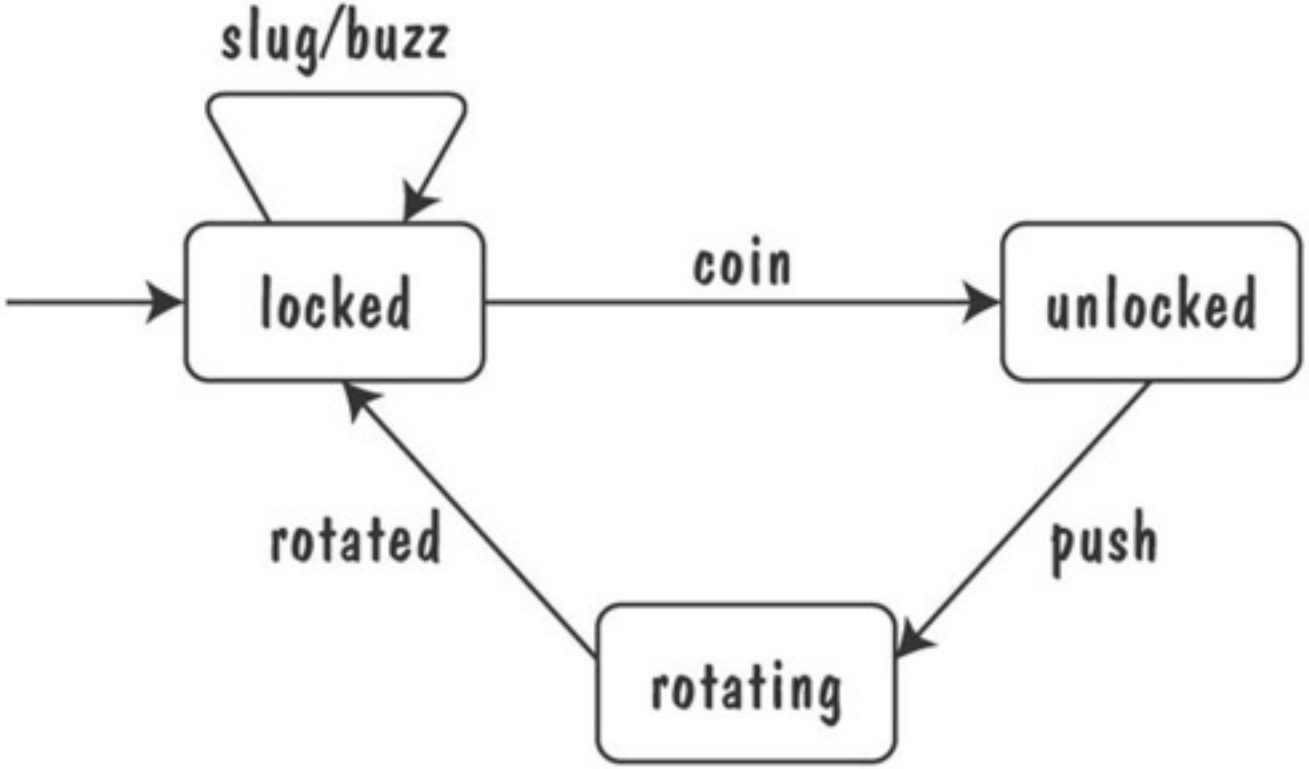
State Machines

- 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 (continued)

- Finite state machine model of the tunstile problem



4.5 Modeling Notations

State Machines (continued)

- **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 have a rich syntax, including state hierarchy, concurrency, and intermachine coomunication

4.5 Modeling Notations

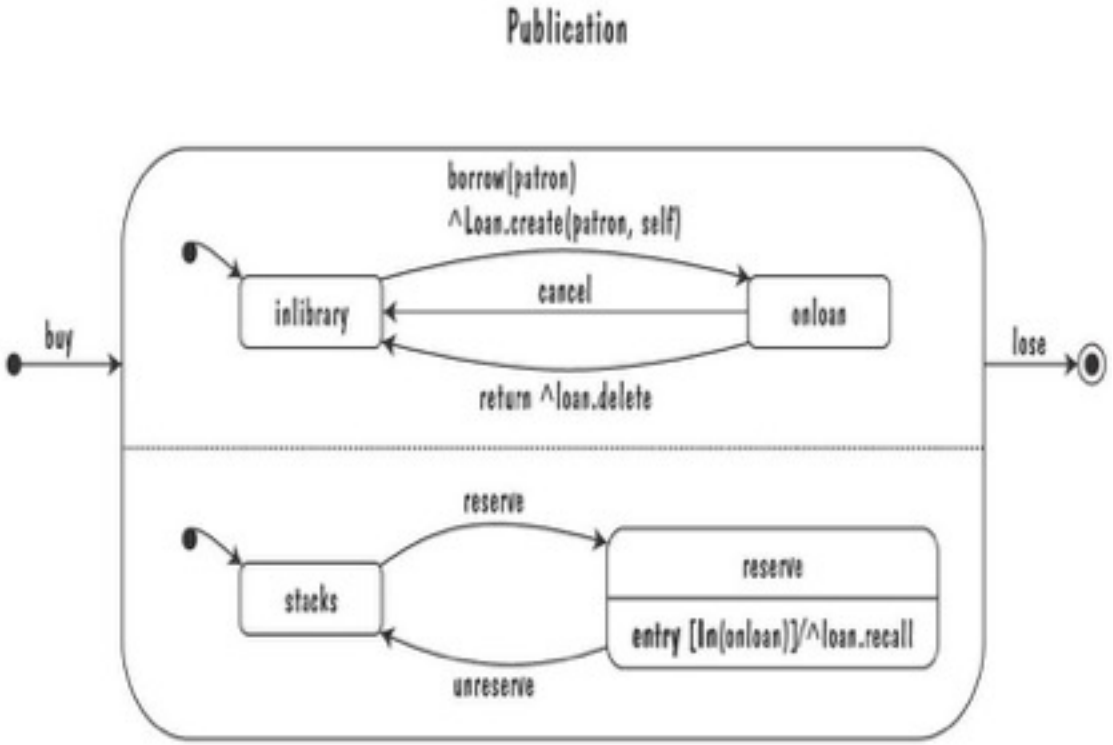
UML Statechart Diagrams (continued)

- **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 line
  - The submachines are said to operate *concurrently*

4.5 Modeling Notations

UML Statechart Diagrams (continued)

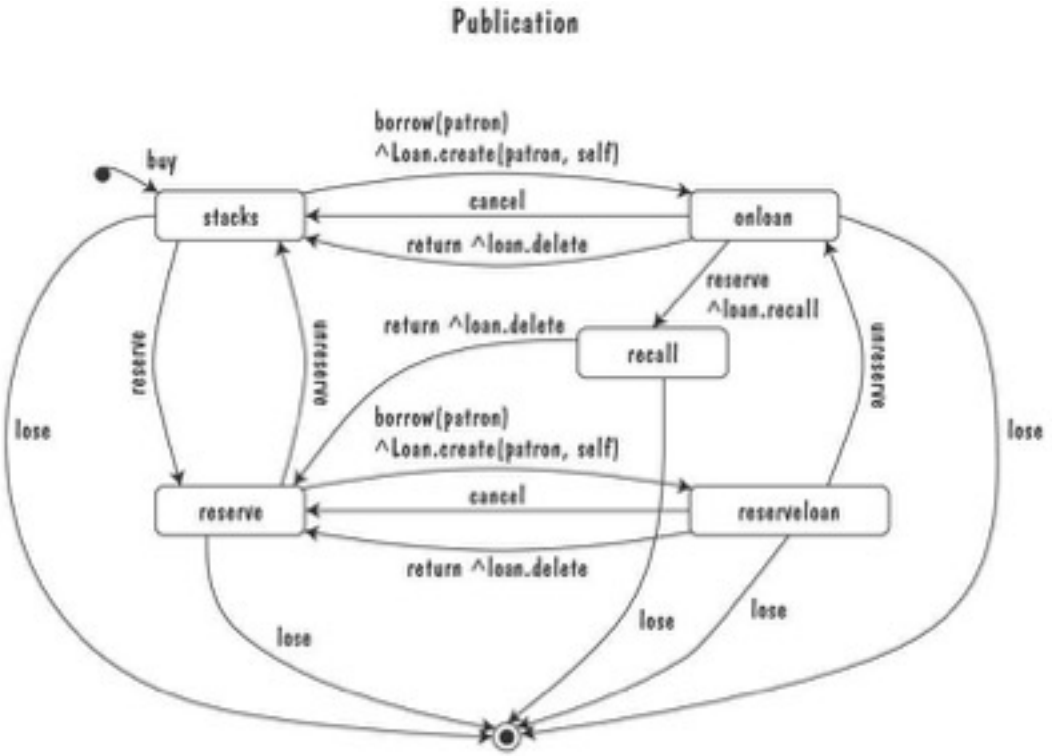
- The UML statechart diagram for the `Publication` class from the Library class model



4.5 Modeling Notations

UML Statechart Diagrams (continued)

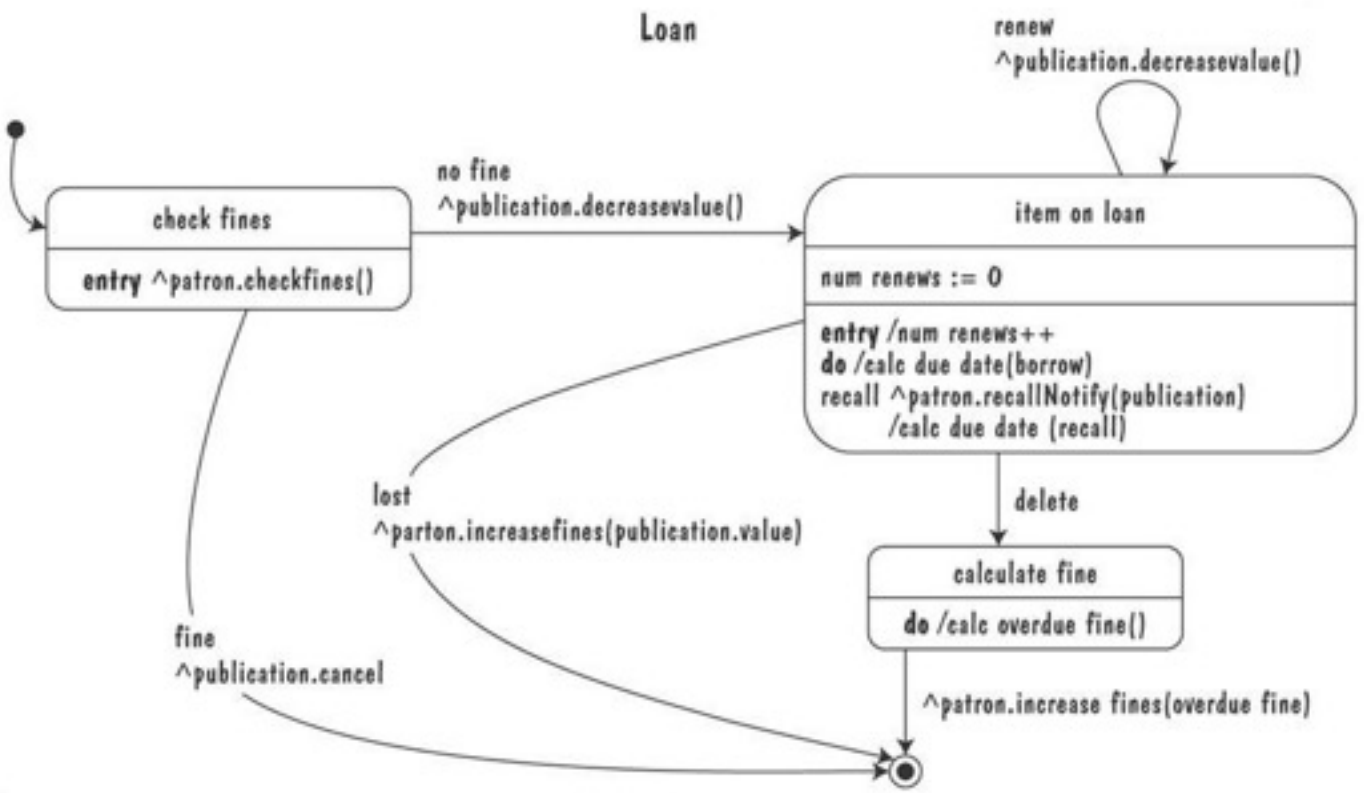
- An equivalent statechart for `Publication` class that does not make use of state hierarchy or concurrency
  - comparatively messy and and repetitive



4.5 Modeling Notations

UML Statechart Diagrams (continued)

- The UML statechart diagram for `Loan` association class illustrates how states can be annotated with local variables, actions and activities



4.5 Modeling Notations

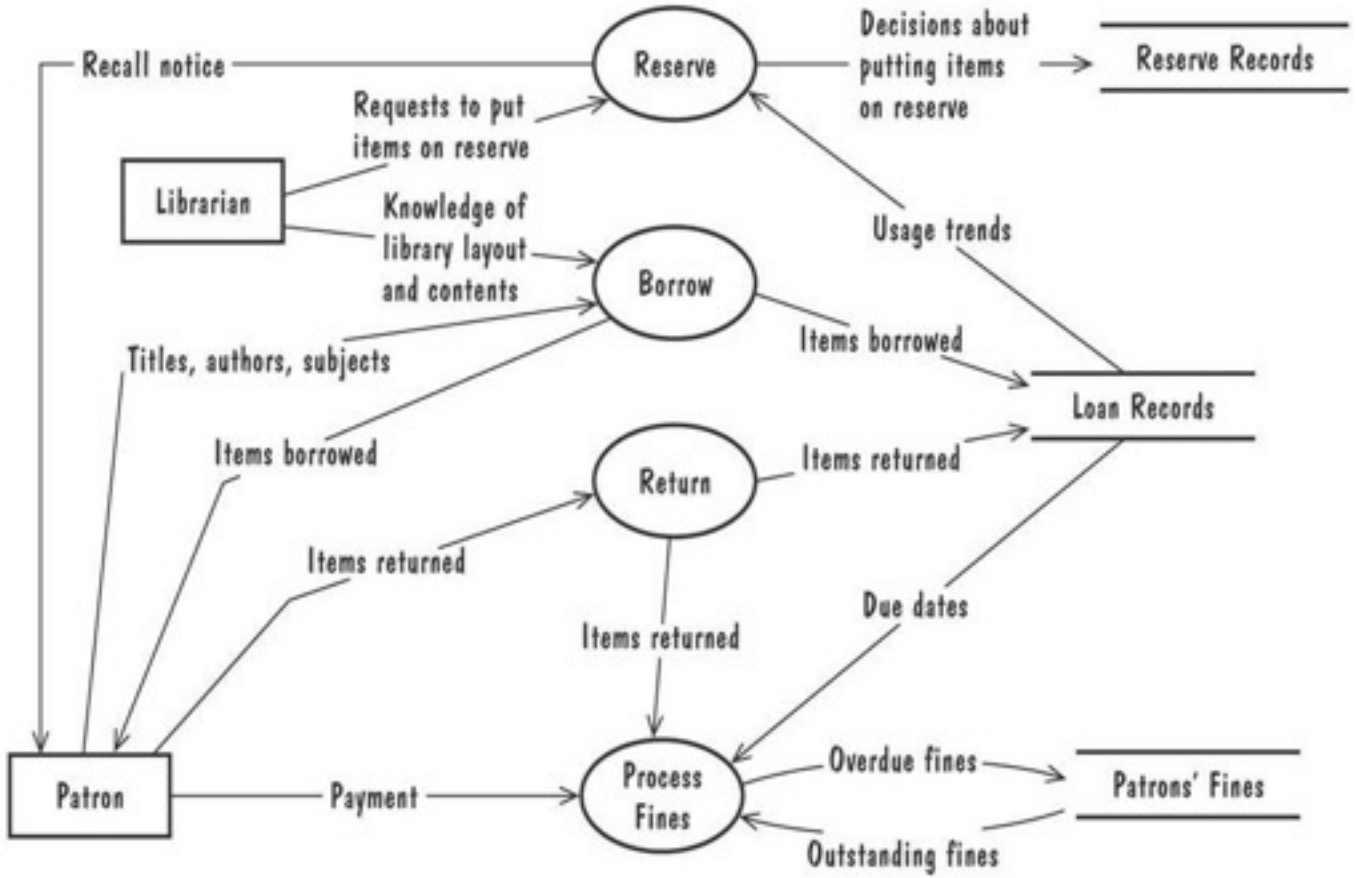
State Machines: Ways of Thinking about State

- 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

Data-Flow Diagrams (continued)

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



4.5 Modeling Notations

Data-Flow Diagrams (continued)

- 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

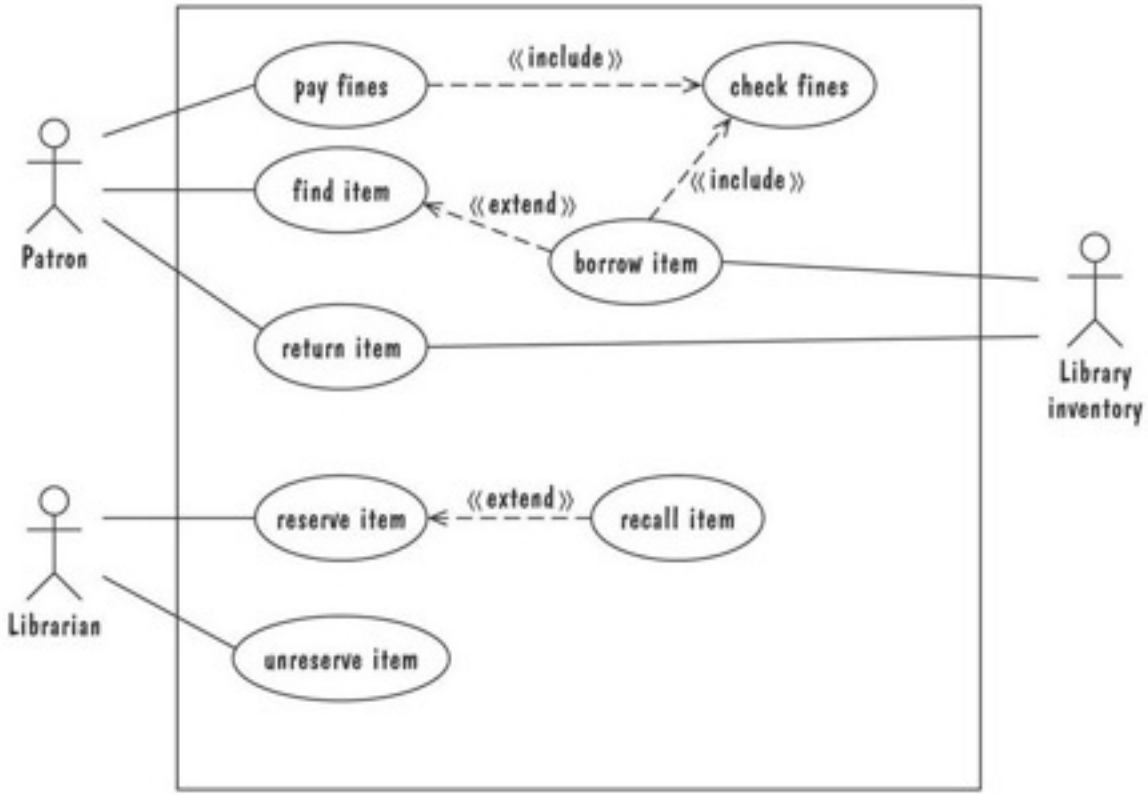
Data-Flow Diagrams Example: Use Cases

- 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

Use Cases (continued)

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



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 (continued)

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

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

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

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 reponses 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 combination of input conditions
- Combinations map to the same set of result can be combined into a single column

4.5 Modeling Notations

Decision Tables (continued)

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

(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					

4.6 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)

4.7 Prototyping Requirements

Building a Prototype

- To elicit the details of proposed system
- To solicit feedback from potential users about
  - what 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 otimizing quality requirements

4.7 Prototyping Requirements

Approaches to Prototyping

- Throwaway approach
  - Developed to learn more about a problem or a proposed solution, and that is never intended to be part of the delivered software
  - Allow us to write “quick-and-dirty”
- Evolutionary approach
  - Developed not only to help us answer questions but also to be incorporated into the final product
  - Prototype has to eventually exhibit the quality requirements of the final product, and these qualities cannot be retrofitted
- Both techniques are sometimes called rapid prototyping

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

Requirement 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 accepatable 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 Specification: Steps Documenting Process

- Describe all inputs and outputs in detail, including
  - the sources of inputs
  - the destinations of ouputs,
  - the value ranges
  - data format of inputs and output 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

Sidebar 4.6 Level of Specification

- Survey shows that one of the problems with requirement specifications was the uneven level of specification
  - Different writing sytles
  - Difference in experience
  - Different formats
  - Overspecifying requirements
  - Underspecifying requirements
- Recommendations to reduce unevenness
  - Write each clause so that it contains only one requirement
  - Avoid having one requirement refer to another requirement
  - Collect similar requirements together

4.8 Requirements Documentation
IEEE Standard for SRS Organized by Objects

1. Intodruction 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
   1. 2.2 Product Functions
   - 2.3 User Characteristics
   - 2.4 Constraints
   - 2.5 Assumptions and Dependencies
- Specific Requirements
   - 3.1 External Interface Requirements
     1. 3.1.1 User Interfaces
     - 3.1.2 Hardware Interfaces
     1. 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

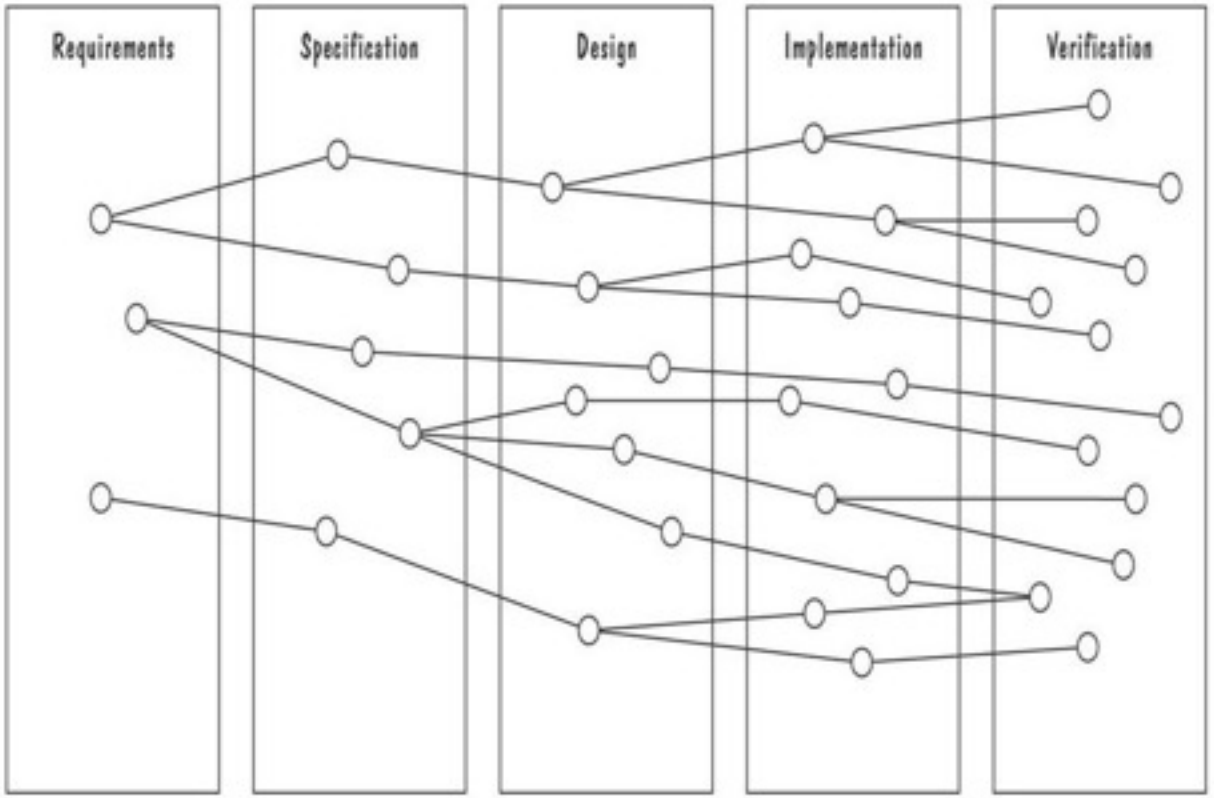
Process Management and Requirements Traceability

- Process managemet 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

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

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 Measuring Requirements

- Measurements focus on three areas
  - product
  - process
  - resources
- Number of requirements can give us a sense of the size of the developed system
- Number of changes to requirements
  - Many changes indicate some instability or uncertainty in our understanding of the system
- Requirement-size and change measurements should be recorded by requirements type

4.10 Measuring Requirements

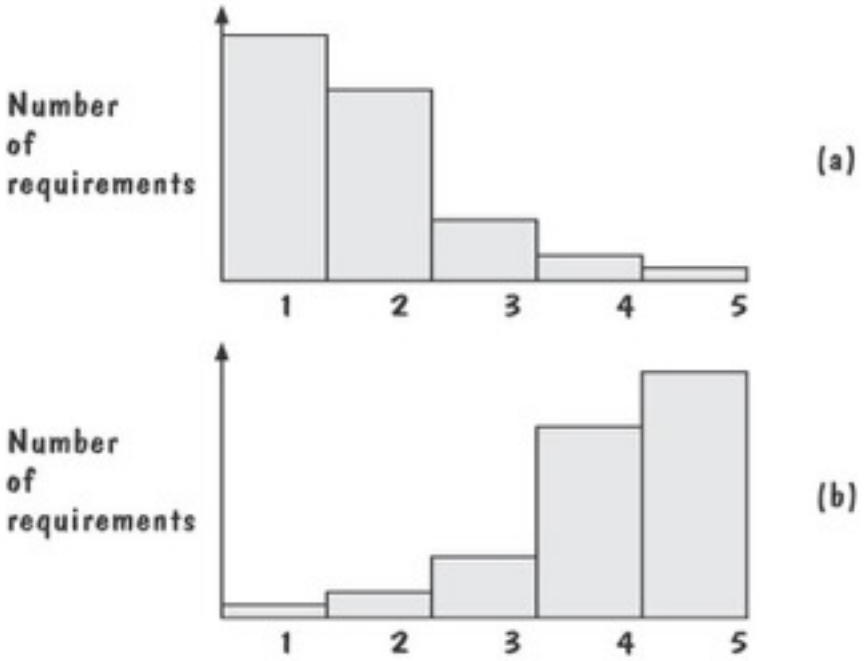
Rating Scheme on Scale from 1 to 5

1. You understand this requirement completely, have designed systems from similar requirements, and have no trouble developing a design from this requirement
2. Some elements of this requirement are new, but they are not radically different from requirements that have been successfully designed in the past
3. Some elements of this requirement are very different from requirements in the past, but you understand the requirement and can develop a good design from it
4. You cannot understand some parts of this requirement, and are not sure that you can develop a good design
5. You do not understand this requirement at all, and can not develop a design

4.10 Measuring Requirements

Testers/Designers Profiles

- Figure (a) shows profiles with mostly 1s and 2s
  - The requirements are in good shape
- Figure (b) shows profiles with mostly 4s and 5s
  - The requirements should be revised



What This Chapter Means for You

- It is essential that the requirements definition and specification documents describe the problem, leaving solution selection to designer
- There are variety of sources and means for eliciting requirements
- There are many different types of definition and specification techniques
- The specification techniques also differ in terms of their tool support, maturity, understandability, ease of use, and mathematical formality
- Requirements questions can be answered using models or prototypes
- Requirements must be validated to ensure that they accurately reflect the customer's expectations

中国科学技术大学软件学院
SCHOOL OF SOFTWARE ENGINEERING OF USTC

谢谢大家！

References

软件工程 - 理论与实践（第四版 影印版） **Software Engineering: Theory and Practice (Fourth Edition),Shari Lawrence Pfleeger,Joanne M. Atlee ,**高等教育出版社

软件工程 - 理论与实践（第四版） **Software Engineering: Theory and Practice (Fourth Edition),Shari Lawrence Pfleeger,Joanne M. Atlee,**杨卫东译, 人民邮电出版社

软件工程—实践者的研究方法（**Software Engineering-A Practitioner's Approach**）; (美) **Roger S. Pressman** 著； 机械工业出版社**ISBN**：**7-111-07282-0**

http://code.google.com/p/advancedsoftwareengineering/