Software Engineering

CS305, Autumn 2020

Week 4

Class Progress…

- Last week:
  - Requirements Engineering Detailed Steps
    - Elicit, Analyze, Specify, Validate, Manage change
  - Requirements modeling
    - Goal-oriented, text-based methods, graphical based methods
  - Object Oriented Analysis and Design – overview
    - Object Modeling Technique
    - Unified Modeling Language (UML) and structural diagrams

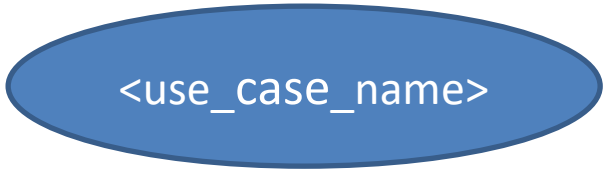
Class Progress…

- This class: UML behavioral diagrams
  - Describe behavior or dynamic aspect of the system
  - E.g.

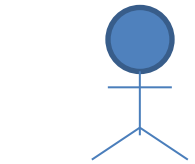
    Use Case diagram / user stories / scenarios

Use Case Diagrams

- Describes outside view of the system
  - Interaction of *outside entities* (Actors) with the system
  - System actions that result in observable actions of value to the actors
- Notation (important ones):



Use case



Actor

Connector between actor and use case (indicates “is the actor of”)

Actor

- Entity: human or device that interacts with the system
- Plays some role
  - Can play more than one role
    - E.g. customer of a bank can also be an employee of the bank (customer and employee are actors)
  - More than one entity can play the same role
    - E.g. an employee and a regular customer can both play the role of a *customer*
  - Can appear in more than one use case

Running Example

1. Registrar sets up the curriculum for a semester using a scheduling algorithm
2. One course may have multiple course offerings (think: sections)
3. Each course offering has a number, location, and time
4. Students register for courses using a registration form
5. Students may add/drop courses for a certain period after registration
6. Professors use the system to receive their course attendance sheets / course rosters
7. Users of the system are assigned passwords to validate at logon

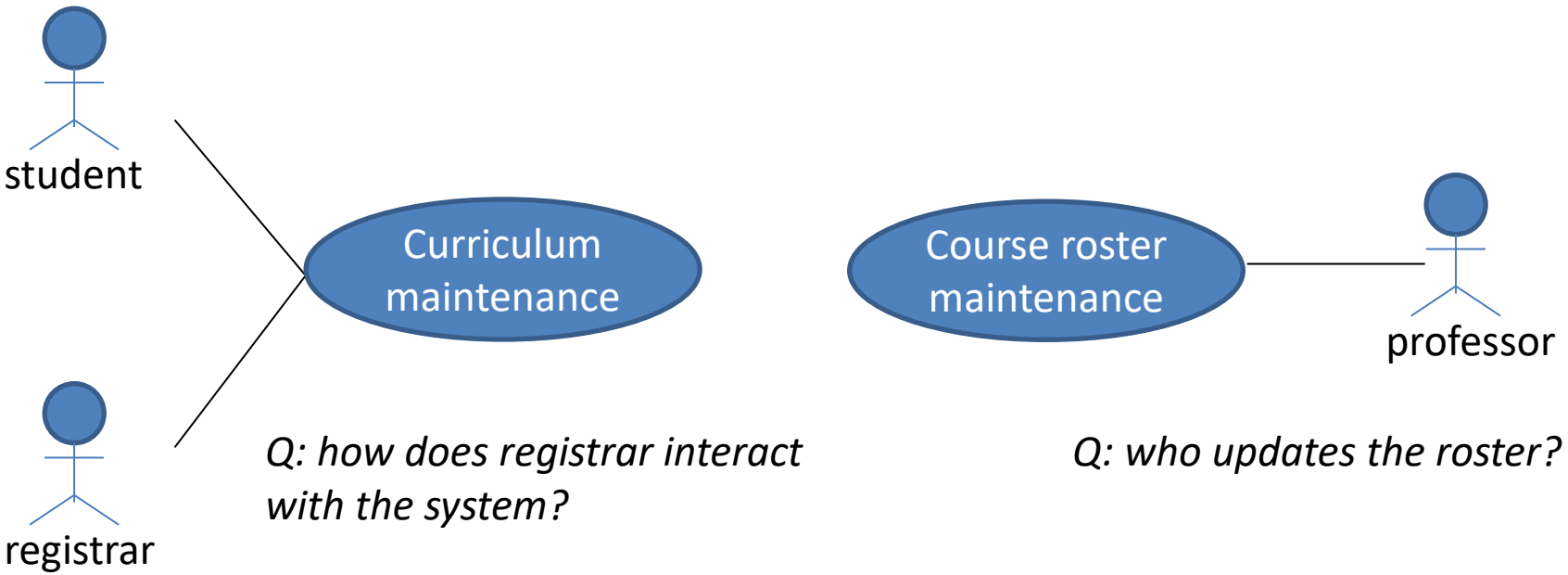
Exercise: Identify Actors

1. Registrar
2. curriculum
3. Semester
4. Scheduling algorithm
5. Course
6. Course offerings
7. Students
8. Registration form
9. Professors
10. Passwords

Exercise: Identify Actors

1. Registrar
2. Curriculum
3. Semester
4. Scheduling algorithm
5. Course
6. Course offerings
7. Students
8. Registration form
9. Professors
10. Passwords

Example Use Case Diagrams

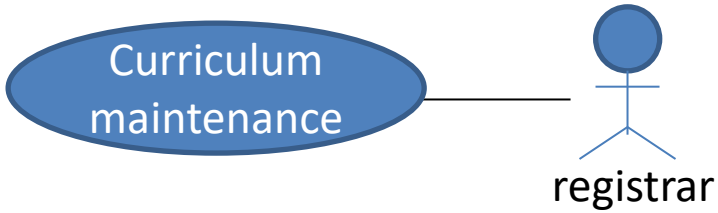


Q: how to document the interactions?

Documenting use case - guidelines

- Describe flow of events either *formally or informally*
  - How the use case starts and ends
  - Normal flow of events
  - Alternative flow of events
  - Exceptional flow of events
- Formal way
  - Use case diagrams, pseudocodes
- Informal way
  - Textual description

Documenting use case - example



- Registrar provides a password to log in to the system
- If the password is valid, the system asks to specify a semester
- Registrar enters the desired semester, and the system prompts the registrar to select an activity: ADD / DELETE / REVIEW / QUIT
- When selected ADD / DELETE, the system allows registrar to add / delete a course
  - When done, the system runs the scheduling algorithm
- When selected REVIEW, the system displays the curriculum for that semester
- When selected QUIT, the system logs out the registrar

Use cases - role

- Why important?
  - More effective requirements elicitation
  - Starting point for analyzing architecture (next topic)
  - Identify priority of users (e.g. Registrar. *If the registrar cannot perform his assigned role? How can a student use the system?*)
    - Help in better planning
  - Help in writing test cases even before the system is defined / coded

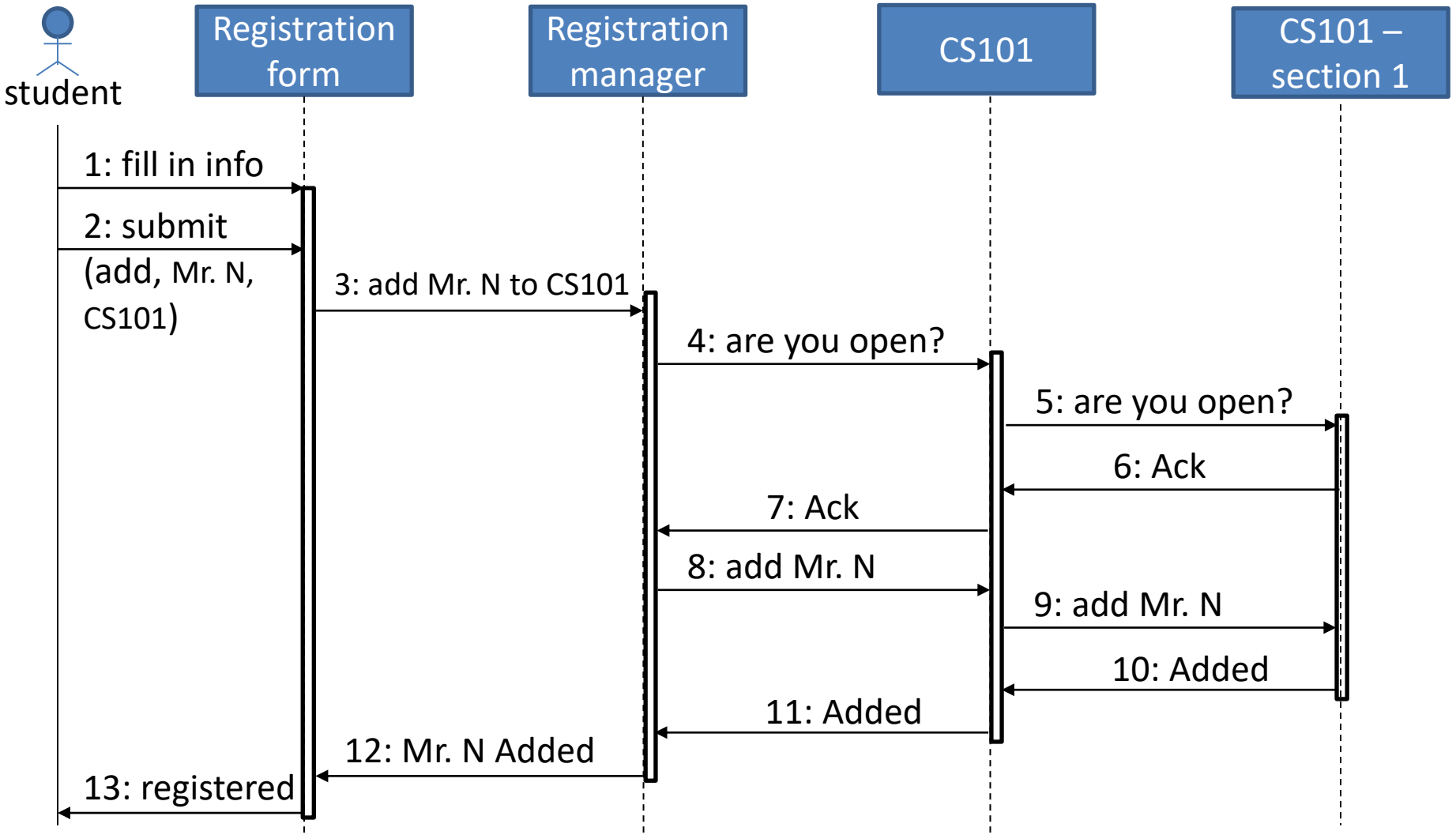
Use case diagram creation - guidelines

- Choose a name that conveys purpose
- Put a single scenario into a use case
- Define the flow of events clearly - helps understand how system works
- Omit irrelevant details
- Extract common flow of events among multiple user interactions to create new use cases i.e. refine e.g. Registrar, student, professors all log in to the system before performing their roles.

Sequence Diagrams

- Interaction diagram that describes how objects / components communicate and the ordered sequence of messages that are exchanged
- Can be used as a formal way to document a use case

Sequence Diagrams - Example



Sequence diagram creation - guidelines

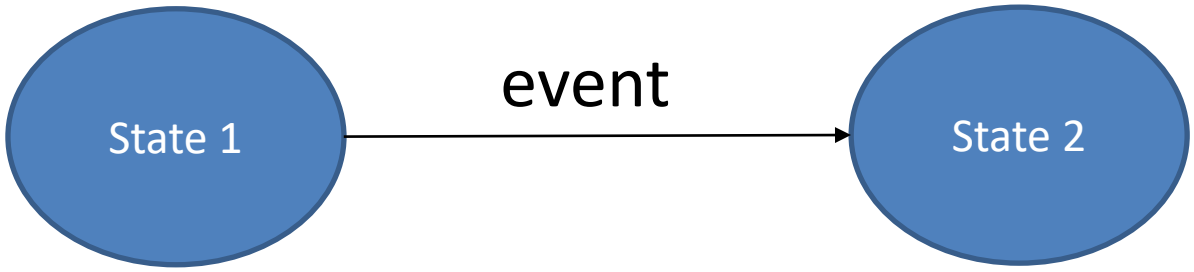
- Draw objects that participate in the interaction at the top along X-axis
  - Place objects that *initiate the interaction* towards the left
- Add object lifelines – lines that show the existence of an object over a period
  - Add dashed lines for all except the left-most object
- Place messages from top to bottom
  - Annotate messages with numbers for added clarity
- Add focus of control – thin rectangular boxes that indicate the period when the object is in action

State Transition Diagrams

- Shows possible life history of each object / class
- Defined for each relevant object / class
- Shows:
  - States of the class (attributes)
  - Events that cause transition from one state to another
  - Actions that result from state change

State Transition Diagrams - Notation

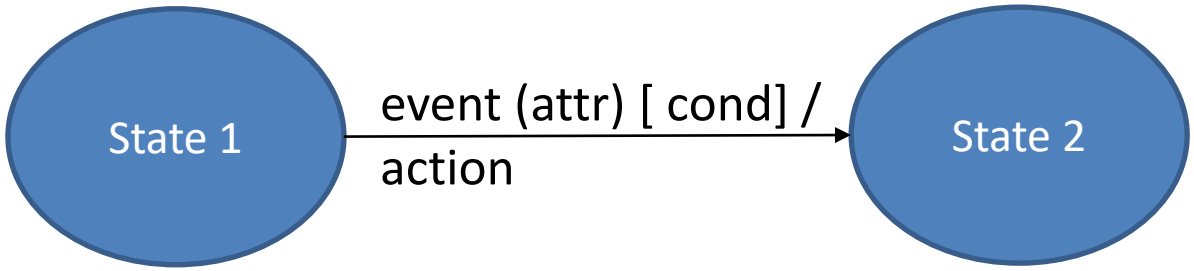
- States are indicated as ovals with names written inside



- Transition is indicated as event that triggers the transition. Indicates passage from one state to another because of some external stimuli (some events may be consumed within the state itself)

State Transition Diagrams - Notation

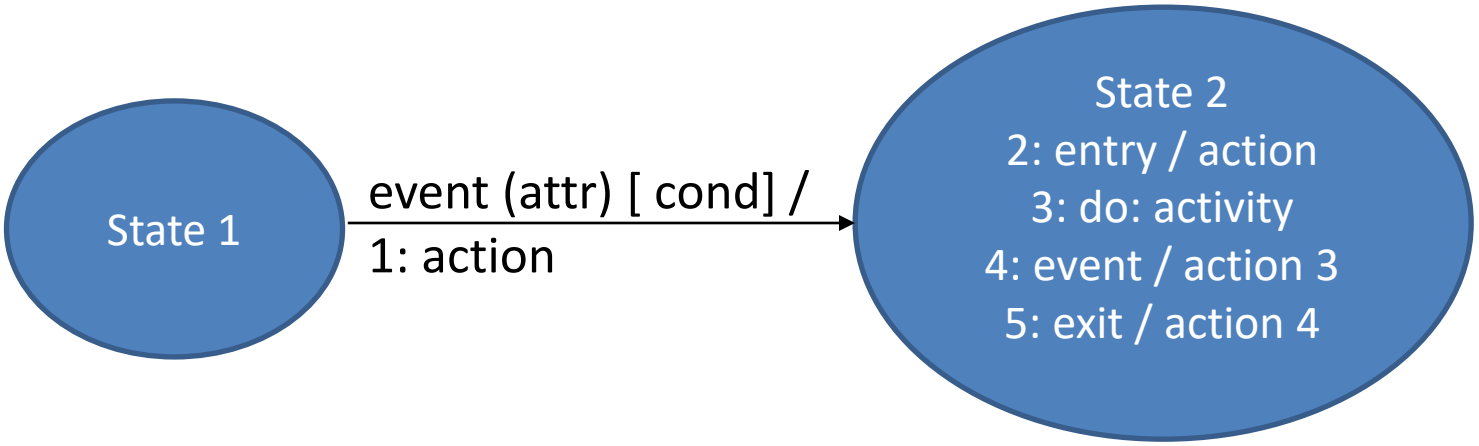
- events might also produce actions



- might also have attributes (analogous to method parameters) and Boolean conditions that indicate that the event is triggered only when the condition is true

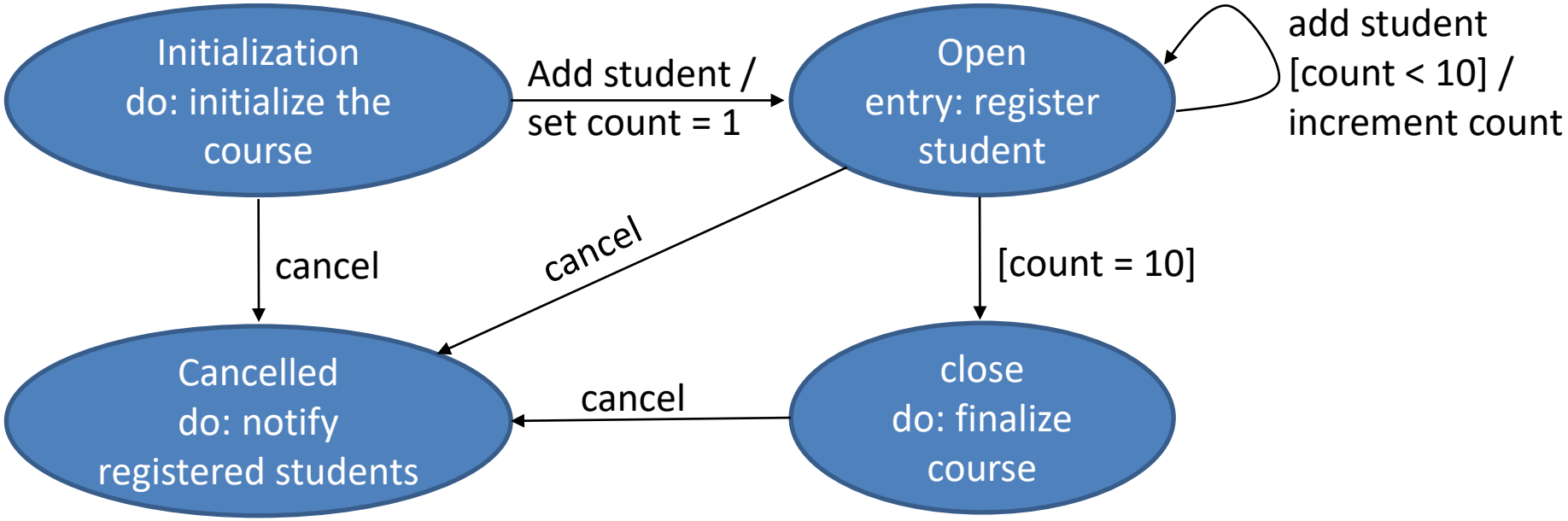
State Transition Diagrams - Notation

- States might also be associated with activities and actions

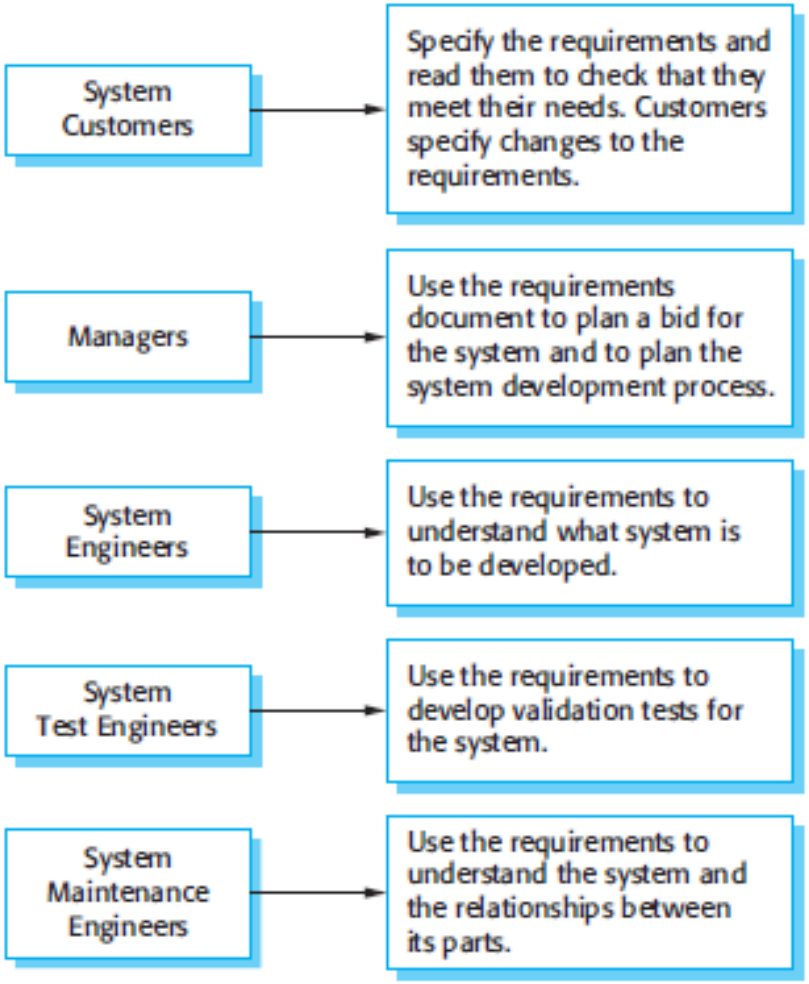


  - activities: operations performed by the object in a given state that take time to complete
  - actions: events that can be triggered upon entry or exit to that state or in response to specific event caused due to an activity performed
  - Numbers indicate the time ordering of actions / activities

State Transition Diagrams - example



Users of a Requirements Document



pic source: Software Engineering, Ian Sommerville

SRS Summary

- Way to communicate requirements to others
- Different projects require different SRSs depending upon the context e.g. small vs. large teams