Requirements

Abram Hindle
hindle1@ualberta.ca

Henry Tang
hktang@ualberta.ca

Department of Computing Science
University of Alberta

CMPUT 301 – Introduction to Software Engineering
Slides adapted from Dr. Hazel Campbell, Dr. Ken Wong

Importance of Requirements

- https://www.projectsmart.co.uk/it-project-management/the-curious-case-of-the-chaos-report-2009.php

Reason for Project Failure	% of Responses
Incomplete requirements	13.1
Lack of user involvement	12.4
Lack of resources	10.6
Unrealistic expectations	9.9
Lack of management support	9.3
Changing requirements	8.7
Lack of planning	8.1
System no longer needed	7.5

Requirements

- Types:
  - User requirements
    - What tasks the user can do with the system
  - Functional requirements (features)
    - What behaviors the system does or supports
  - Non-functional requirements (qualities)
    - How well the system should do what it does
    - E.g., response time, resource usage, availability

Requirements

- Types:
  - External interfaces
    - E.g., interfaces to other hardware and software, data sources and sinks, formats, protocols
  - Physical setting
    - E.g., location, workspace, lighting, noise, temperature
  - Developer constraint
    - E.g., implementation technology, documentation

Requirements

- Types:
  - Business requirements
    - Why the system is needed
  - Business constraint
    - What the system or process must comply with
    - E.g., corporate policy, industry standard, government regulation

Requirements

- Requirements should be:
  - Correct
    - Requirements properly represent user needs
  - Complete
    - All possible scenarios are described
  - Consistent
    - Requirements do not contradict each other
  - Clear
    - No ambiguities
  - Realistic
    - Can be achieved by “mere mortals”

Requirements

- Also desired:
  - Traceable
    - Can trace functionality and tests to the requirement being satisfied
  - Verifiable
    - Repeatable test(s) can be designed to show that the system fulfills the requirement

Verifiable Requirements

- Verifiable?
  - “The system shall have a good user interface.”

Verifiable Requirements

- Verifiable?
  - “The system shall respond to the user in under one second for most tasks.”

Verifiable Requirements

- Verifiable?
  - “When the output state changes, it is logged in the event log.”

Verifiable Requirements

- Verifiable?
  - “The system shall be free of defects.”

Requirements Activities

- Done iteratively:
  - Requirements elicitation
    - Discover user needs
  - Requirements analysis
    - Decide scope and priorities
    - Study feasibility
  - Requirements specification
    - Detail the requirements in terms the *users* can *understand*

Users

- Who is the “user”?
  - Primary
    - End user
    - With frequent hands-on use
  - Secondary
    - Manager of end users
    - With occasional use, or via an assistant
  - Tertiary
    - Owner of the system
    - Uses output, influences or makes funding decisions

Users

- Some characteristics to consider:
  - Background
    - Literacy and language
    - Motivation to learn
    - Domain knowledge
    - Task familiarity
    - Computer skills
    - Attitude towards computers and technology

Users

- Some characteristics to consider:
  - Perceptual, motor, and tactile abilities
    - Seeing and hearing difficulties
    - Fine motor skills with input devices
  - Physical
    - Height and strength (for kiosk design)
    - Hand and finger size and strength (for mobile device design)
    - Health, age, and gender
  - Social
    - Relationships with peers
    - Culture

Users

- Kinds of use:
  - Infrequent use/novice user
    - Need wizards
    - Need clear prompts, error handling
  - Frequent use/expert user
    - Need keyboard shortcuts
    - Need customization, programmability

Users

- Some issues to consider:
  - Users cannot always express what they want
    - But they often know what they do not like
  - Users may not know what is possible
    - What is technically and economically feasible?
  - Users stick to what they know already works, or have always done
  - Users may fear job losses
    - Leads to non-constructive participation

Users

- “Innovator’s dilemma”:
  - Attributed by Clayton Christensen
  - As the user base for an application grows, there is a tendency for developers to focus on this increasingly expert (and vocal) group of users
  - The system becomes more sophisticated
  - Development becomes “optimized” for them

Users

- “Innovator’s dilemma”:
  - Potential new users need “less”
  - Experts don’t want their app “dumbed down”
  - Competitor attracts the new users with a simpler “good enough” app
  - Original app loses market share due to disruption from the low end

Understanding

- Tips:
  - Manage expectations
    - Be clear and honest about claims
    - Avoid surprises, disappointments, hype
  - Involve the user
    - Build tangible prototypes to gain feedback
    - More likely to forgive problems if they are involved
  - Establish a glossary
    - Terminology used in the application domain (not programming domain)

User Stories

Specifying Needs

- User story:
  - Written description of what a user wants to achieve with the system

Organized on index cards

Typical forms:

- *As a «user role», I want «goal».*
- *As a «user role», I want «goal», so that «reason».*

Defining User Stories

- Tips:
  - Describe *what* not *how*
    - Avoid technical details or choices of technologies, unless it is a development constraint
  - Avoid epics for near-term needs
    - Better to split up huge stories into more, smaller stories (but not too small)

Defining User Stories

- Tips:
  - Prioritize user stories
    - Discuss with the user what they find of most value, and stage development on that first
  - Can attach an effort estimate to complete
    - Normally sized to take days, not many weeks
  - Use stories to plan development tasks
    - Create work items in the iteration plan

“SMART Work Items”

S	Specific
M	Measurable
A	Achievable
R	Relevant
T	Time Boxed

- https://xp123.com/invest-in-good-stories-and-smart-tasks/

“INVEST in Good Stories”

I	Independent
N	Negotiable
V	Valuable
E	Estimable
S	Small
T	Testable

- https://xp123.com/invest-in-good-stories-and-smart-tasks/

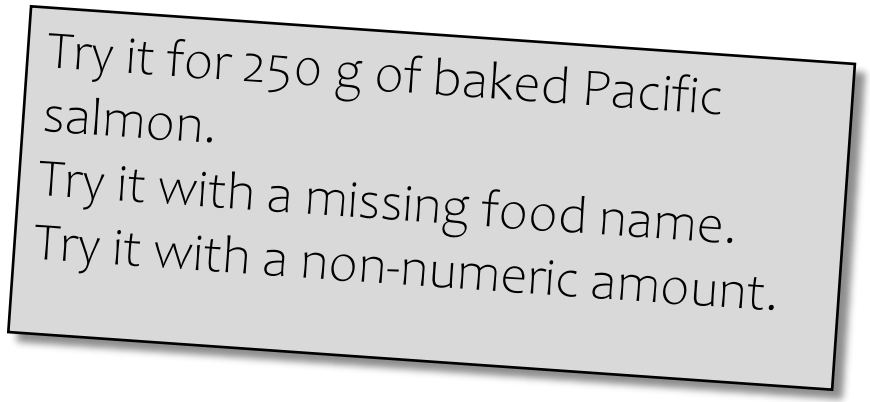
Testable User Stories

- Front of the card:

As a meal planner, I want to see nutrition information for a given amount of a given food.

Testable User Stories

- Back of the card:



Acceptance tests

- Link:
  - https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/

Use Cases

Identifying Tasks

- Study what tasks users do:
  - What is the goal and context?
  - What information is needed?
  - What are the steps?
  - Who does the user work with?
  - Why is it done this way?

Identifying Tasks

- Scenario:
  - An informal narrative
  - Personal and concrete, but not particularly general
  - Use the scenario to understand existing goals, task flow, and possible irritants

Scenario

- Example:
  - "I want to track the calories for a meal, so I consult the USDA Nutrient database. I want to look up 'Pacific salmon' so I enter that as the keywords. Item not found! So I enter 'salmon' and try again. That works, but I get 46 items, including salmonberries and even cloudberries. Why? I choose 'fish, salmon, sockeye, cooked, dry heat', then figure 2.5 x 100 g units for my item, and scan the table to see 422 kcal in the energy row."

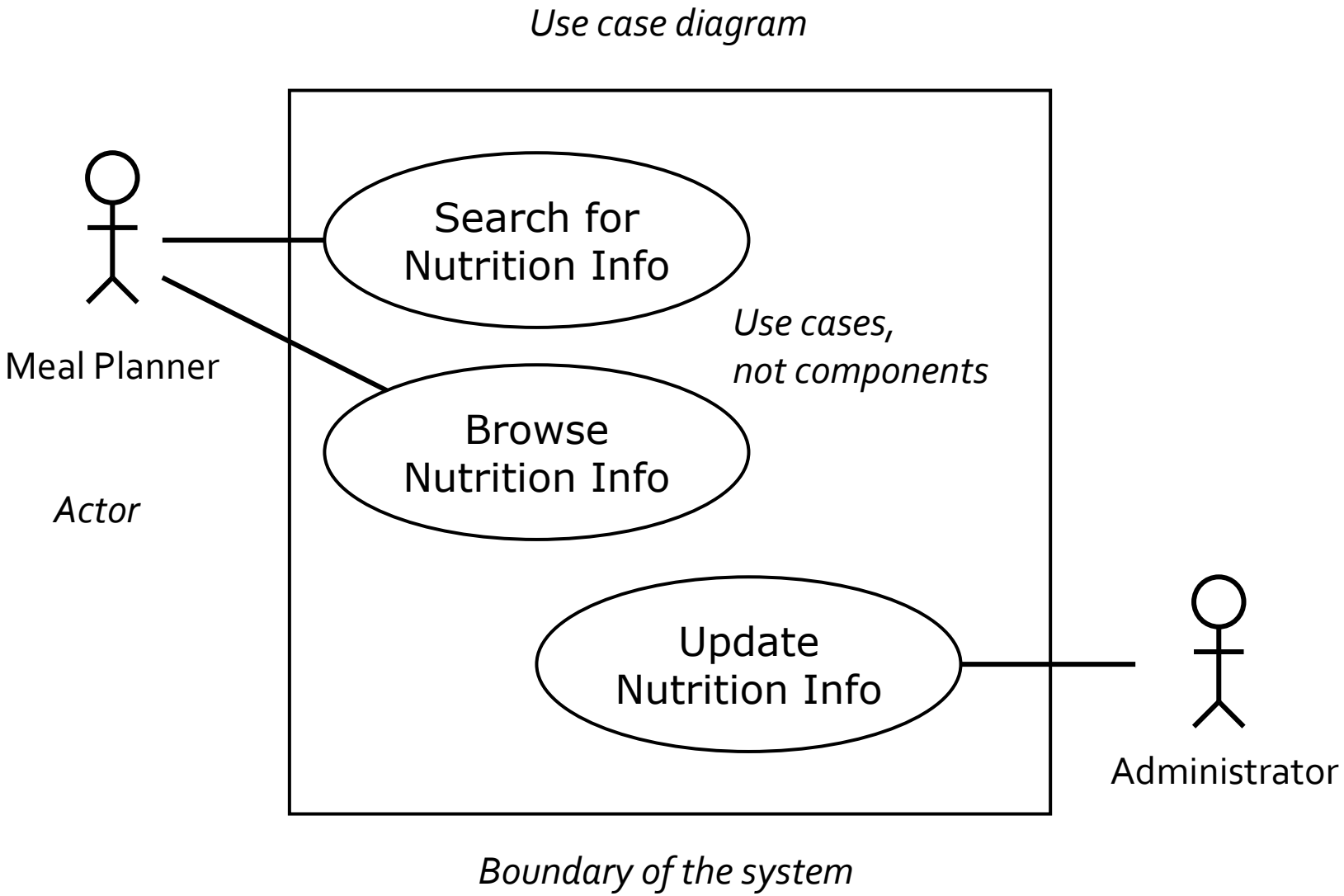
Specifying Tasks

- Use Cases:
  - Capture the goal, conditions, and steps of a coherent interaction between the actor(s) and the software system
  - More general than a specific scenario
  - Written from a “user” point-of-view

Defining Use Cases

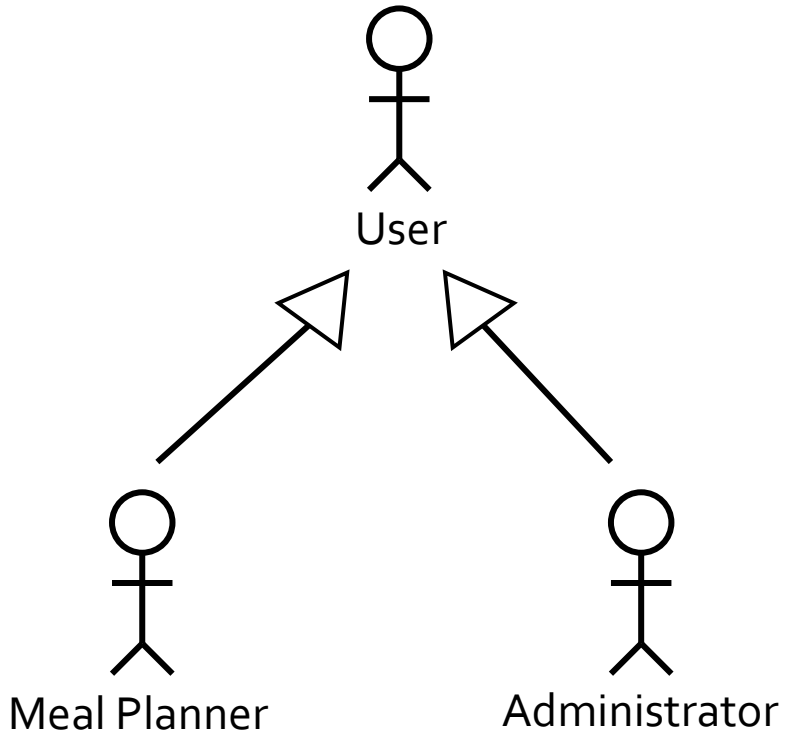
- Stages:
  - Identify the actors
    - Consider different user roles and external systems
  - Define use cases
    - Include all cases of use
  - Refine use cases
    - Consider exceptional conditions and qualities
  - Relate use cases
    - Consider inclusion and extension dependencies

Identify the Actors



Identify the Actors

- Actor generalization:



Define/Refine Use Cases

- Example:

Use Case Name	**SearchForNutritionInfo**
Participating Actors	Meal Planner (primary)
Goal	Meal Planner finds nutrition information
Trigger	Meal Planner chooses the Search option
Precondition	Meal Planner knows food name and amount.
Post-condition	On success, nutrition information displayed.
	…

Define/Refine Use Cases

- Example:

User point of view

Basic Flow	1	System prompts Meal Planner to enter keywords
	2	Meal Planner submits keywords.
	3	System lists matching foods, prompting for a selection.
	4	Meal Planner browses and selects a food.
	5	System prompts for food weight in units of 100 g.
	6	Meal Planner enters food units.
	7	System presents nutrition data for the amount of food.
		…

Avoid implementation specifics

Define/Refine Use Cases

- Example:

Exceptions 3	If there are no matching foods
3.1	System displays an error
3.2	System returns to step 1
7	If given food units is non-numeric, use 0 and proceed

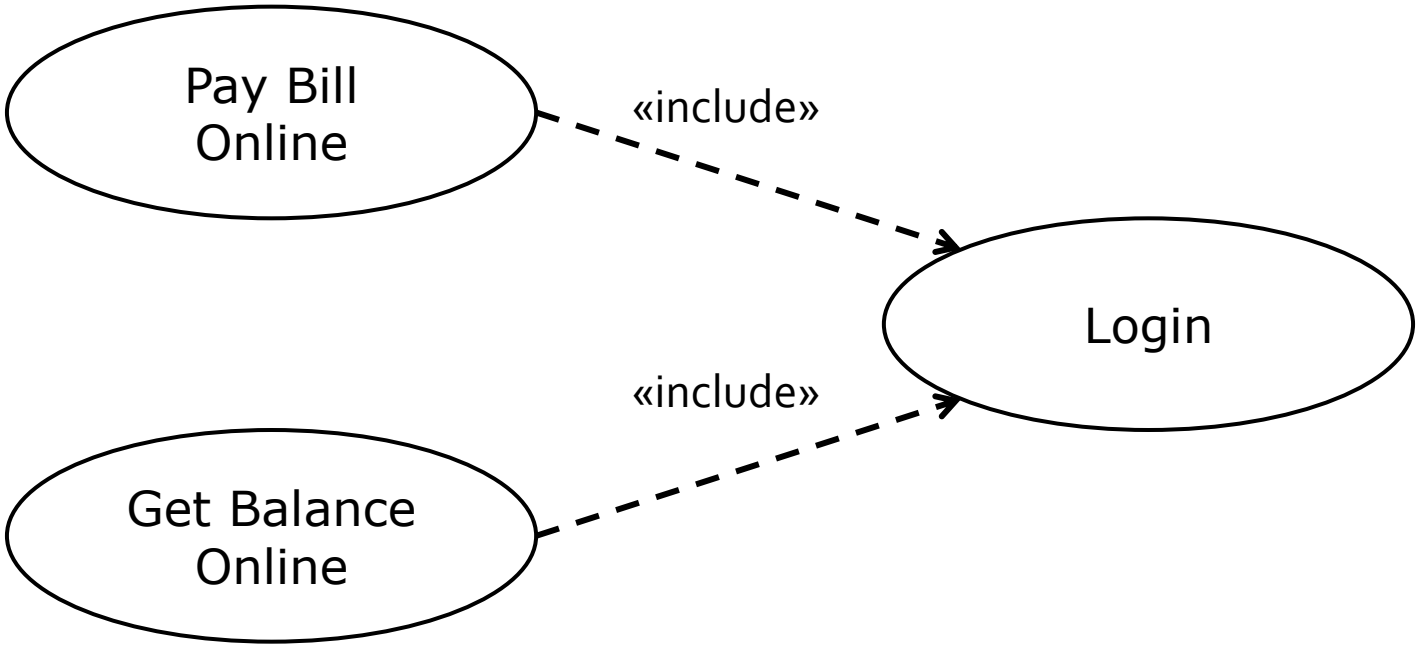
Define/Refine Use Cases

- Example:

Qualities	System responds in under 2 s for list of matching foods and for nutrition data on a specific food.
Constraints	Use USDA nutrition data.
Includes	
Extends	
Related Artifacts	
Notes	
Open Issues	

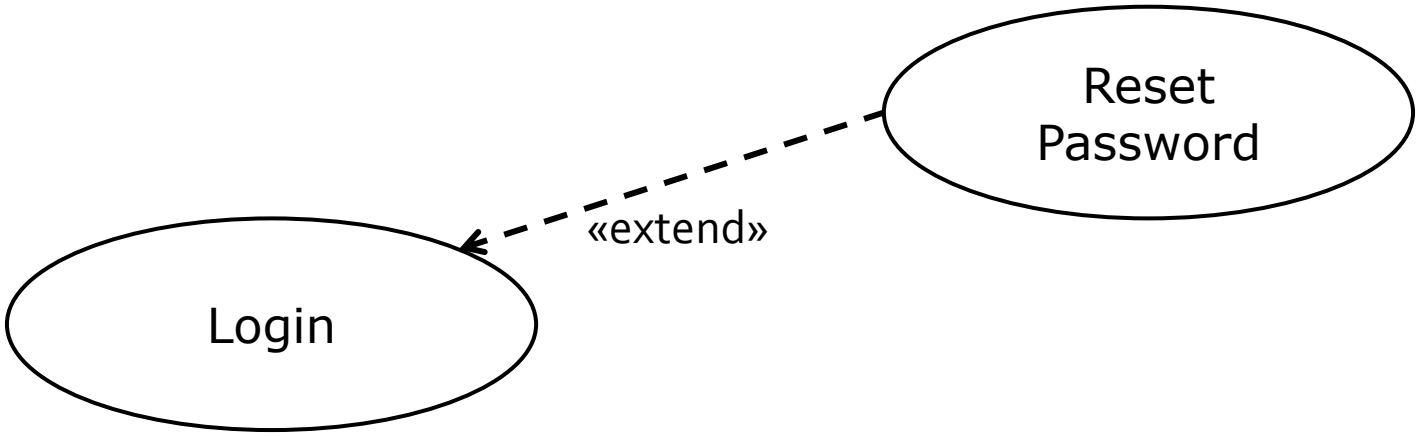
Relate Use Cases

- Inclusion:
  - A use case may include another use case (for necessary, shared behavior)



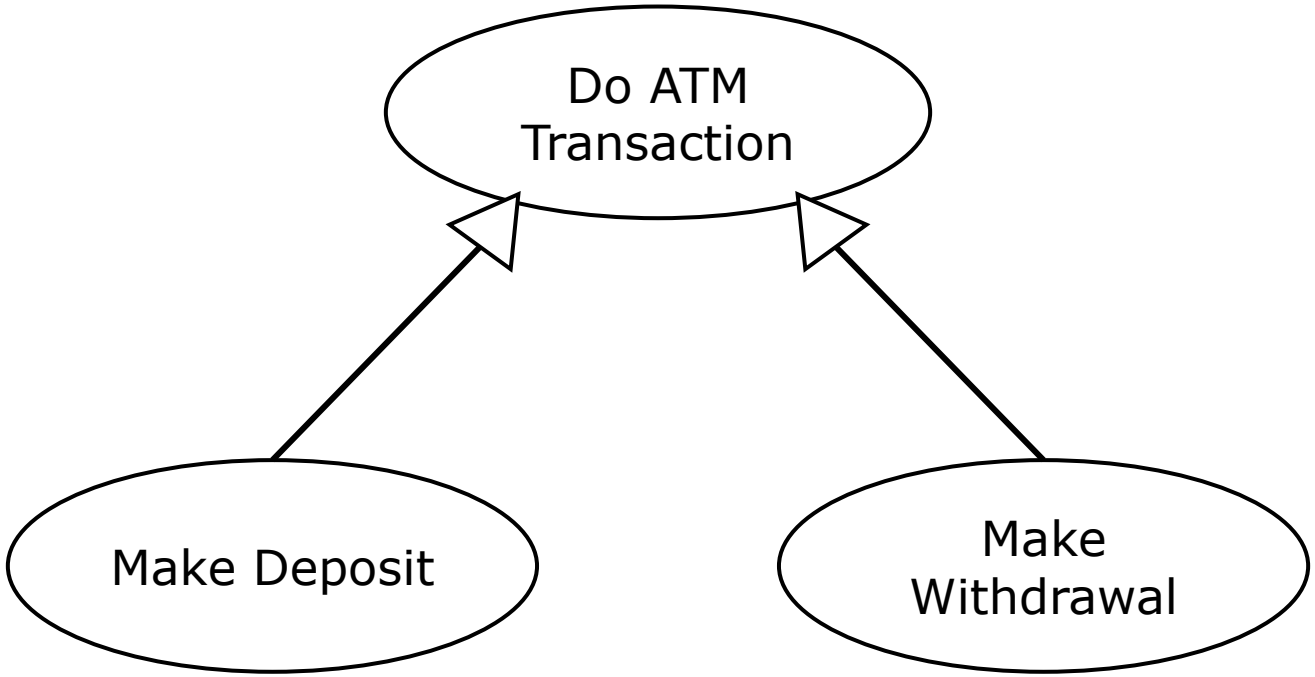
Relate Use Cases

- Extension:
  - A use case may be extended by another use case (for optional or exceptional behavior)



Relate Use Cases

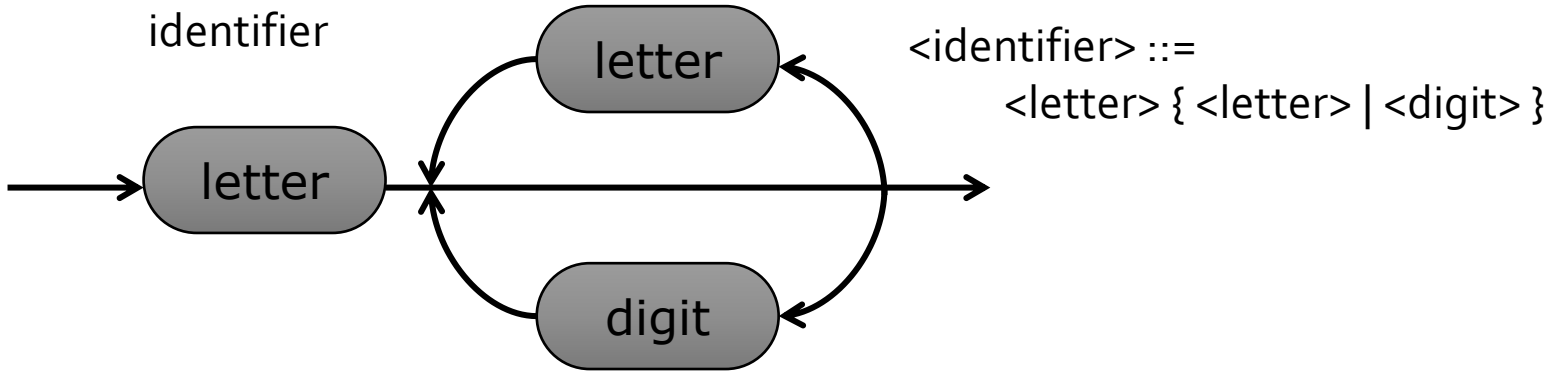
- Use case generalization:
  - A use case may be a specialization of a more general use case



Augmenting Requirements

Augmenting Requirements

- Can add other descriptions, for example:
  - Use cases to user stories
  - Data schemas
  - Sample input and output
  - User interface mockups and storyboards
  - Grammars (language syntactic/lexical structure)

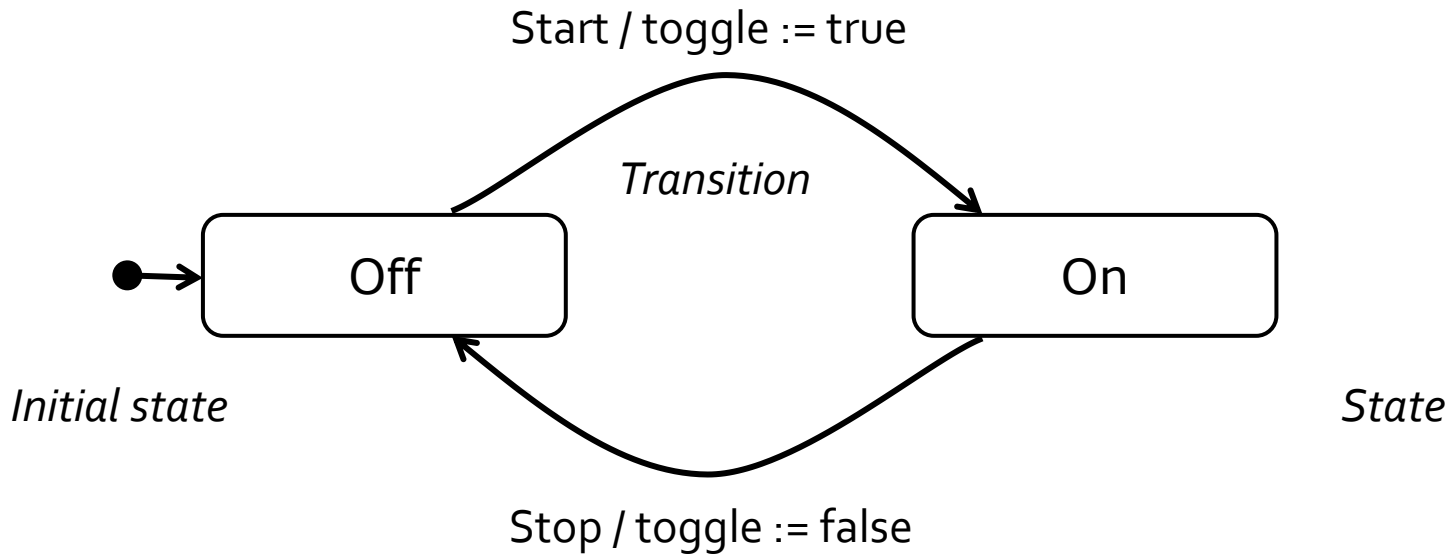


State Models

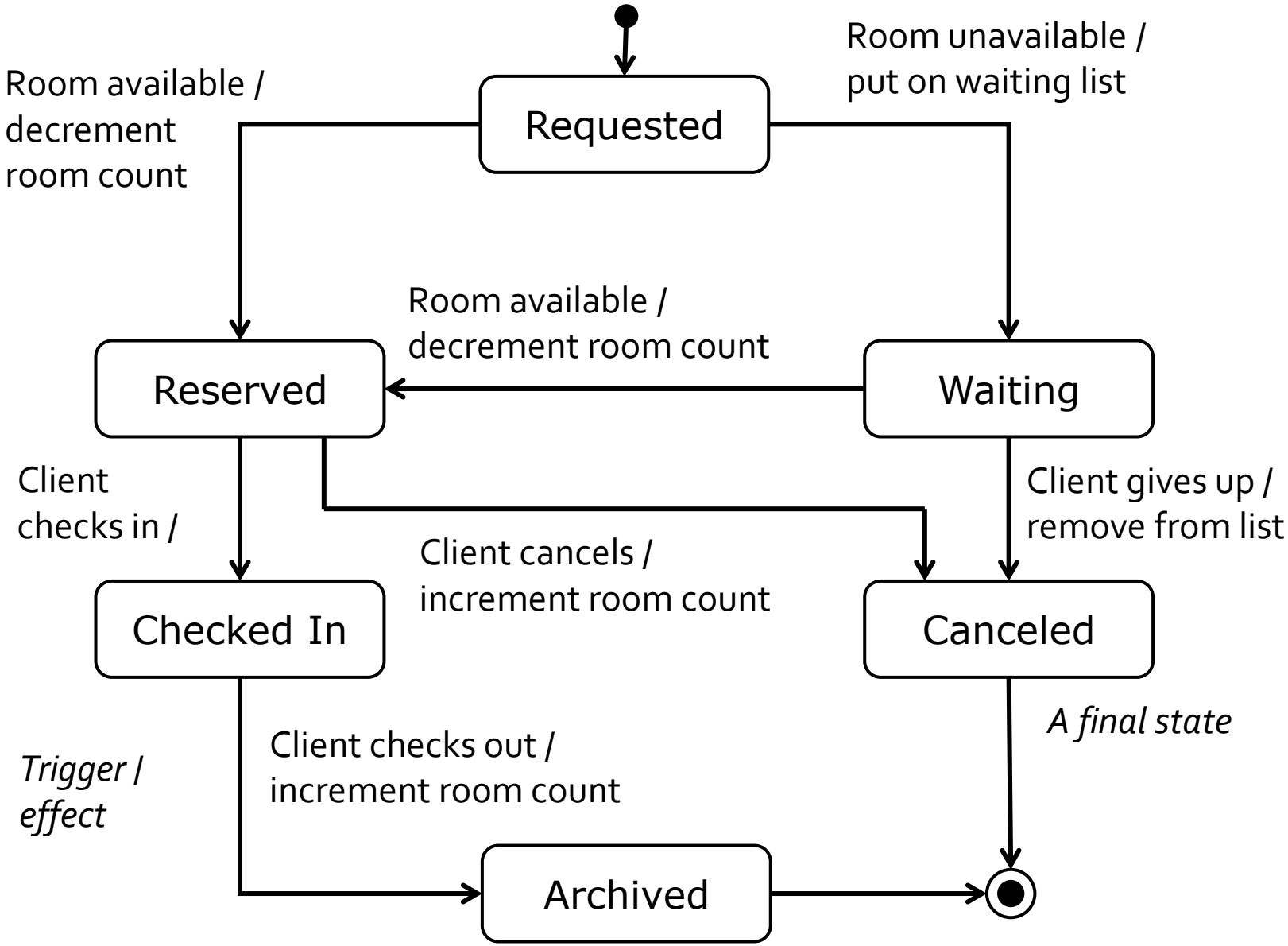
- Modeling behavior:
  - Used in formally modeling the behavior of a specific object in response to external events

UML State Diagram

- Modeling behavior:
  - *States* in which something can be in
    - A situation represented by attribute values
  - Directed *transitions* between states
    - Triggered by events, input, time, messages, etc.

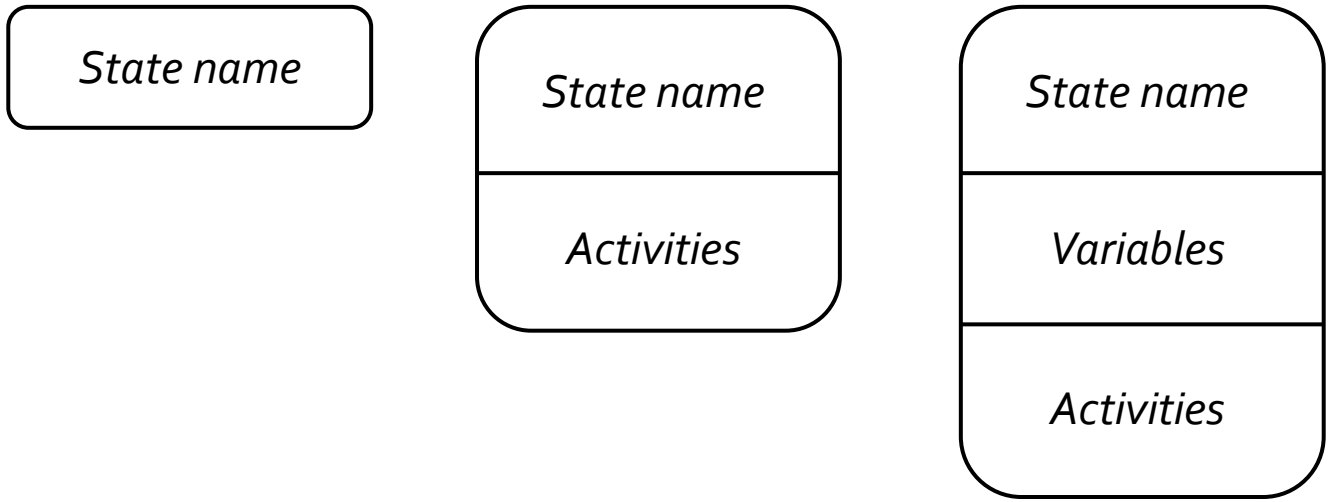


UML State Diagram



UML State Diagram

- States:



UML State Diagram

- Transitions:

trigger [*guard*] / *effect*

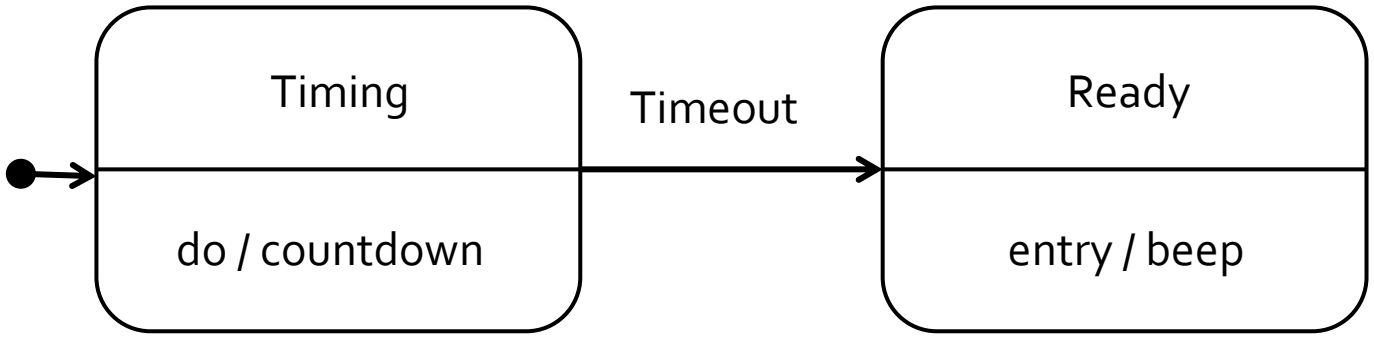
General form of transition label

  - If in a current state,
    and *trigger* occurs,
    and *guard* constraint (if any) is true …
  - Then perform state exit actions (if any),
    perform corresponding transition *effect* (if any),
    perform new state entry actions (if any);
  - Otherwise, stay at current state.

UML State Diagram

- Activities in states:

entry / *action*	perform *action* when entering state
do / *action*	perform *action* while in state
exit / *action*	perform *action* when exiting state

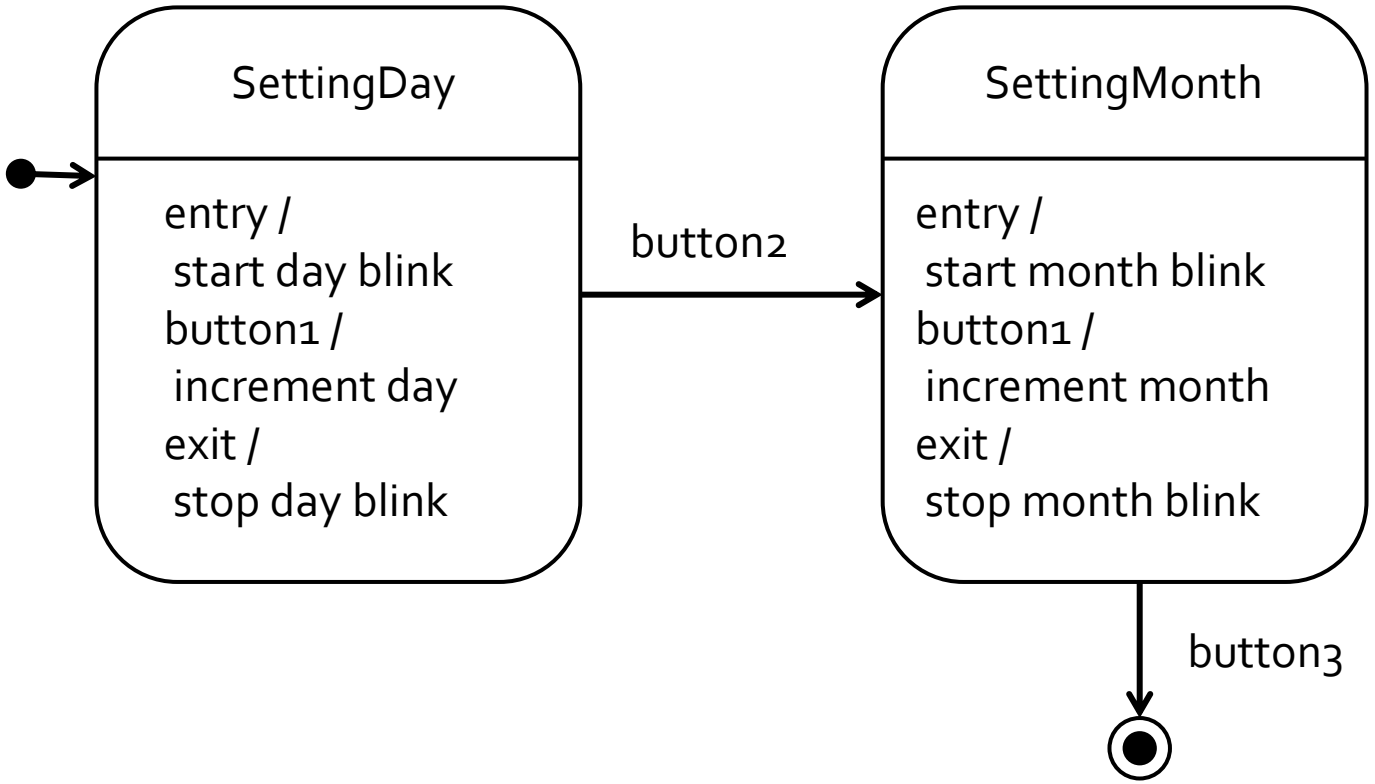


UML State Diagram

Also called an internal transition

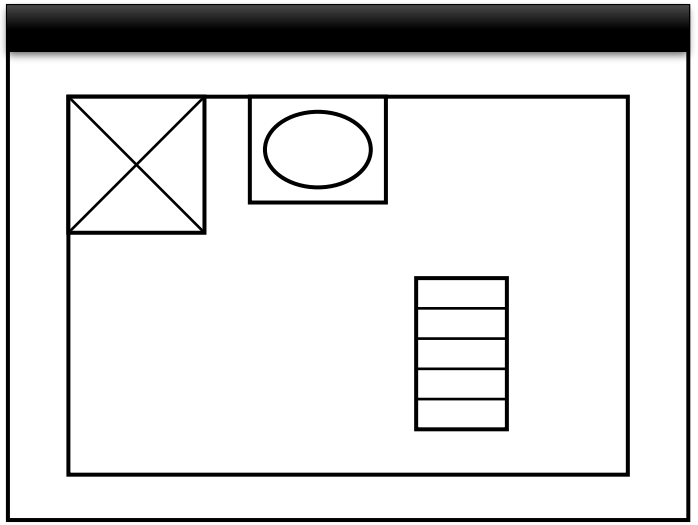
- Activities in states:

trigger / *action*	when *trigger* occurs, perform *action*



Room Planner

- Canvas:
  - To place and move items
- Mouse events:
  - Click
  - Press/drag/release
- Item type menu:
  - Choices of fixtures and furniture



Room Planner

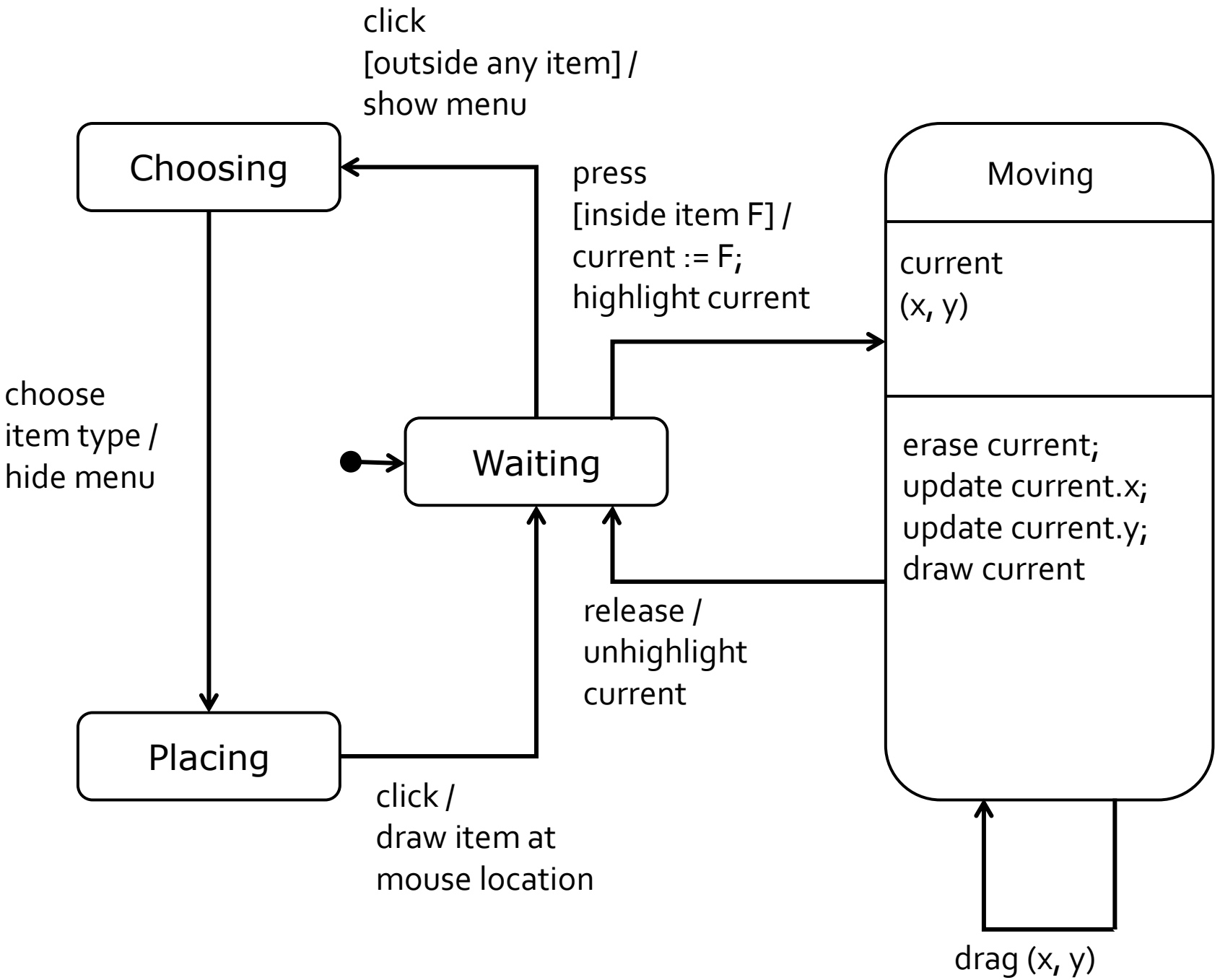
- Placing an item:
  - User clicks on canvas outside any item
    - System shows the item type menu
  - User chooses an item type from the menu
    - System hides the item type menu
  - User clicks on the canvas
    - System draws item of the chosen type at the mouse location

Room Planner

- Moving an item:
  - User presses inside an item
    - System highlights item
  - User drags item
    - System shows moving item
  - User releases mouse
    - System puts item at new location
    - System removes item highlighting

Room Planner

- States:
  - Waiting
    - Nothing happening
  - Choosing
    - Choosing item type from menu
  - Placing
    - Placing chosen shape
  - Moving
    - Moving item to new position


click
[outside any item] /
show menu
Choosing
press
[inside item F] /
current := F;
highlight current
Moving
current
(x, y)
choose
item type /
hide menu
Waiting
erase current;
update current.x;
update current.y;
draw current
release /
unhighlight
current
Placing
click /
draw item at
mouse location
drag (x, y)

UML State Diagram

- Tips:
  - Check for completeness
    - States reachable?
    - Missing transitions?
    - Events not considered?
    - Unforeseen situations?
  - Check for dangerous situations
    - E.g., exiting without having saved edits

UML State Diagram

- Tips:
  - Check for consistency
    - Similar interactions have similar effects?
    - Effects are visible and give good feedback?
  - Aid the user
    - Is undo appropriate, in each state?
    - Is cancel or escape appropriate?
    - Is invoking help appropriate?

More Information

- Books:
  - The Essence of Object-Oriented Programming with Java and UML
    - B. Wampler
    - Addison-Wesley, 2002
  - UML Distilled
    - M. Fowler
    - Addison-Wesley, 2003
  - User Stories Applied
    - M. Cohn
    - Addison-Wesley, 2004

More Information

- Books:
  - More About Software Requirements
    - K. Wiegers
    - Microsoft, 2006
  - Software Engineering: Theory and Practice
    - S.L. Pfleeger
    - Prentice-Hall, 2009

More Information

- Links:
  - Effective User Stories for Agile Requirements
    - https://www.mountaingoatsoftware.com/presentations/introduction-to-user-stories