Use Cases
An Introduction

CS 345 Winter 2018

Chris Reedy

Use Case Approach
Actors

- An *actor* is a person, another system, or a hardware device that interacts with the system in question to achieve a useful goal.

- When an actor is a person, the actor represents a user role.
  - A given user could have multiple roles

- Actors are external to the software/system being described

Use Case Approach
Use Cases

- A *use case* is a discrete, stand-alone activity that an actor can perform to achieve some outcome of value.
  - A use case describes a sequence of interactions between a system and an actor.
  - A single use case can contain a number of similar tasks having a common goal.
  - Or, a use case is a collection of related usage scenarios

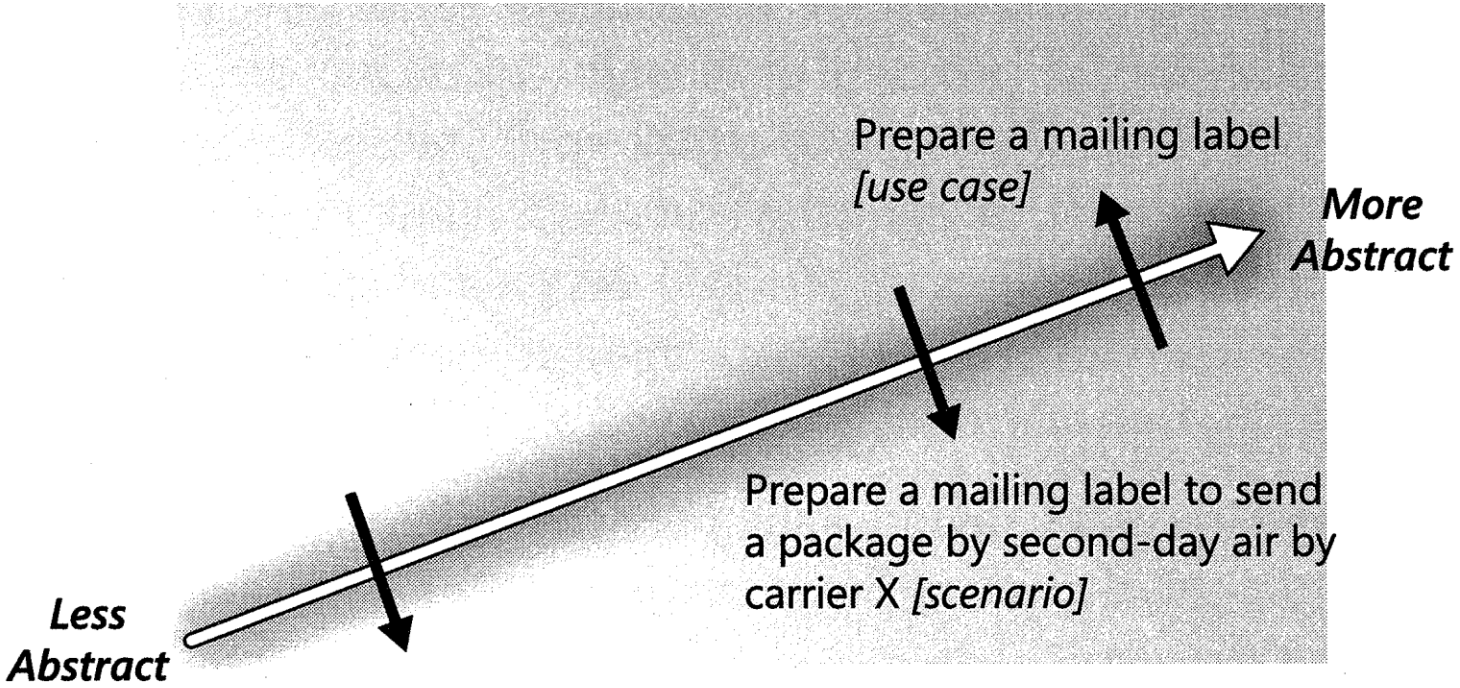
Why Use Cases?

- Shift the perspective
  - From: what functions the system should perform for the users
  - To: what the users want to accomplish using the system
- It's a lot easier for users to describe what they would do with a system.

- Example:
  - What functions an ATM provides

    versus
  - How a user uses an ATM to withdraw money
    - developers figure out the functions

Use Cases, Scenarios, and Stories

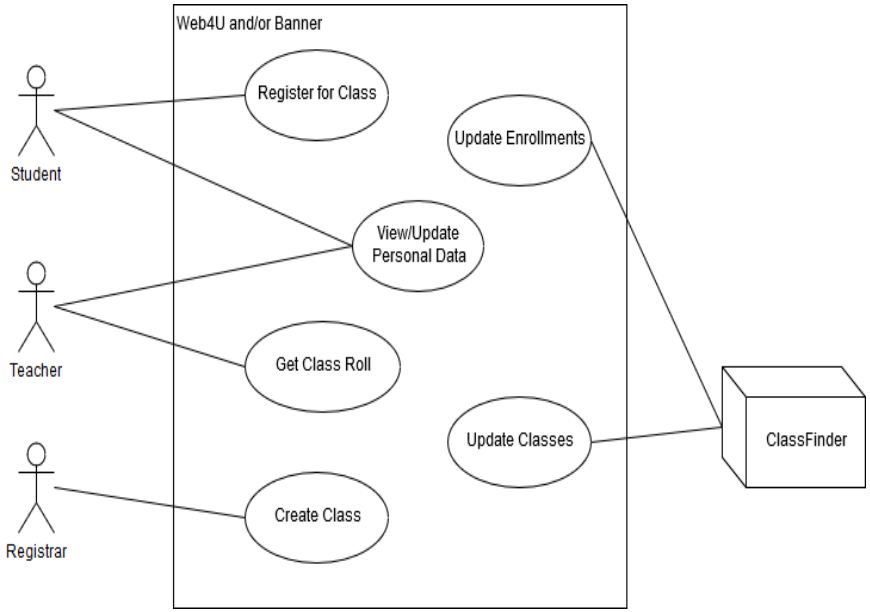


User stories are frequently used by agile software developments

Usage Scenarios

- One scenario is the *normal course* or *main course*, *basic course*, *normal flow*, *primary scenario*, or *happy path*.
- Other successful scenarios are described as *alternative courses* or *secondary scenarios*.
  - Alternative courses always represent successful task completion.

Use Case Diagrams



- The actors are around the outside
  - Primary actors are usually on the left and secondary actors on the right
  - Note that ClassFinder is an external system
- The use cases are inside the box, which represents the system boundary
- The diagram acts as a “table of contents” for the use cases

Warning

The use case diagram is not the use case.

- The diagram shows the names of the use cases and which actors interact with each use case.
- It provides no other information about the structure or content of the use case.

Uses Cases and Functional Requirements

- Software developers don't implement business requirements or use cases
  - Developers implement specific functions
- Can use cases represent all the requirements, or all the functional requirements?
  - Answer: Probably not

Benefits of Use Cases

- Users have a clearer expectations of what the system will do for them
  - In a function-centric approach, the users are forced to synthesize the system benefits based on the individual functions
- Use cases help to avoid over-specifying the system
  - Prevents “orphan” functionality – functionality that doesn’t support any user tasks
- Use cases help with prioritization
  - High priority requirements derive from high priority use cases

Some Traps to Avoid

- Including user interface design in use cases
  - Example: “system displays drop-down list” (bad) instead of “system presents user with choices” or, maybe, “user selects choice of …”
- Use cases the users don’t understand
  - Make sure use cases are written from the users’ perspective

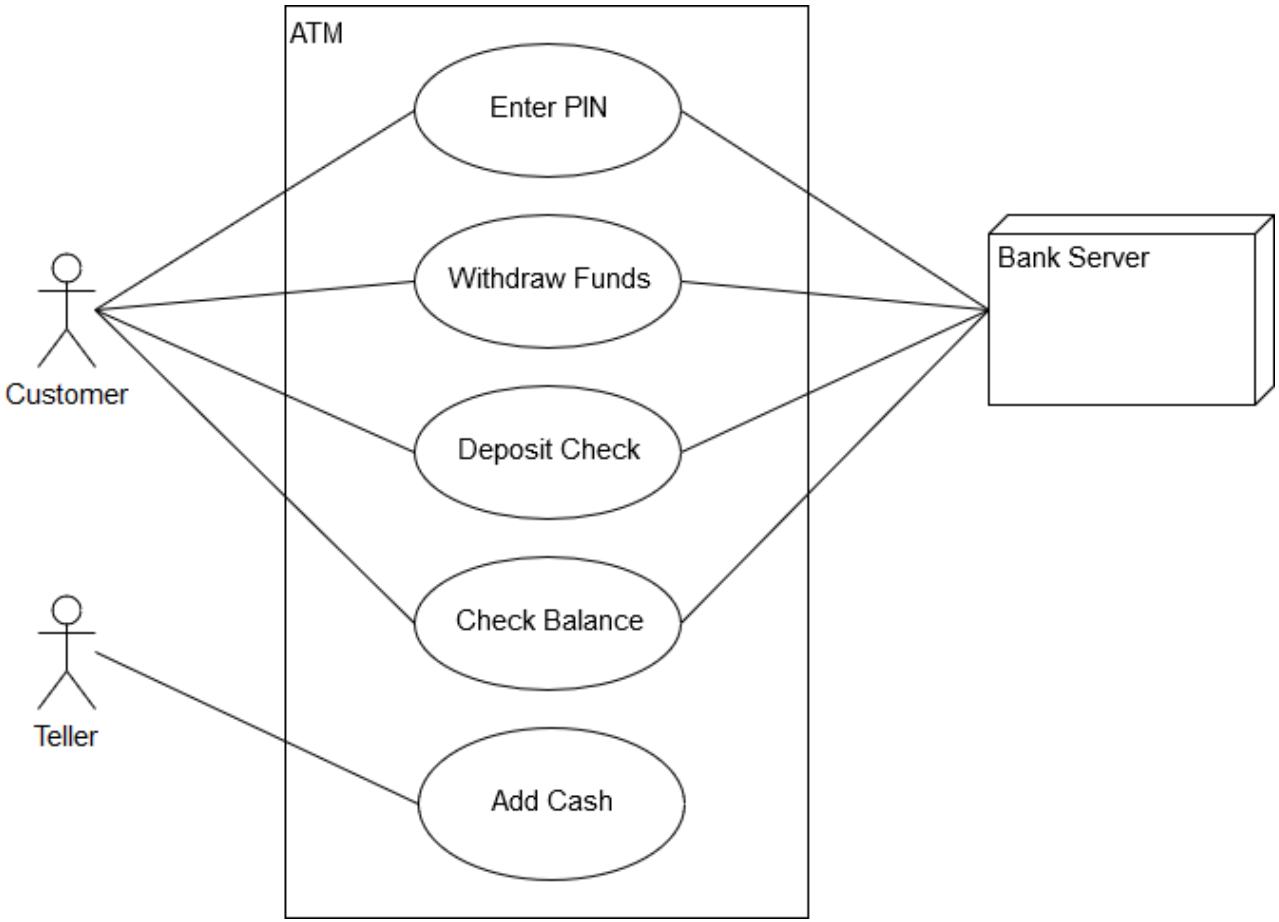
Project Type Limitations

- Use cases are great for capturing user/system interactions
  - Includes web sites, kiosks, any system (or part of a system) driven primarily by user interaction

Documenting Use-Cases For This Class!

1. Name of use case
2. Description, including what the user wants to accomplish
3. Preconditions
   - What needs to be true before the use-case is executed
4. Postconditions
   - What will be true after the use-case is executed
5. Steps

ATM Use Case Diagram



Example Use Case (1 of 2)

- Name: Withdraw Funds
- Description: Customer withdraws money from their account using the ATM machine.
- Preconditions:
  - Customer has presented card to ATM machine which has read the card
  - Customer has entered PIN number for the card
- Postconditions:
  - Customer has requested funds
  - Customer’s account has been debited by cash dispensed

Example Use Case (2 of 2)

- Steps
  1. Customer selects option to withdraw funds
  2. Customer specifies account
  3. Customer enters amount of money to withdraw
  4. ATM machine requests fund withdrawal from central server
  5. Central server responds
     a) If central server authorizes withdrawal:
        1) Dispense requested funds
        2) Debit customer's account
        3) Print receipt for withdrawal transaction
     b) If central server does not authorize withdrawal:
        1) Display appropriate error message
  6. End transaction
- Note: There are a number of possible ways to handle the alternatives in step 4. Any that convey the message are acceptable.

Finding classes and methods

- In the Use-Case
  1. Look for all things mentioned in the use case.
     - Each of these possibly represents a class
  2. Look for all the actions and operations mentioned in the use case.
     - Each of these possibly represents a method on an object

- Hard work: The above is raw material. Now you need to work through the details.

Possible Classes and Operations ATM Example

- Customer
  - getAccounts
  - validatePIN
- Money
  - getAmount
- Server
  - Request withdrawal
- Account
- Transaction
  - printReceipt