

---

# [CS3704] Software Engineering

Dr. Chris Brown  
Virginia Tech  
9/13/2023

# Announcements

---

- **Project Milestone 1 due Friday (9/22) at 11:59pm**
  - Lightning talk presentation slides
  - Proposal document

# Project Questions so far

---

- Do we need to mark or mention every process to the repo, and if the process is not that important,...Pick a process for milestone requirements
- What does the level of involvement on our end/work will look like as we continue with milestones? Slightly more involved as the semester goes on
- If we pick a project not on the given ideas list do we have to get it approved?  
Yes, in □ talk and proposal
- Do we grade teammates? Team feedback in retrospective
- Are multiple groups allowed to have the same project idea or will we need to make sure that ours is unique? Can use the same idea
- When we choose what type of workflow process or process model we want to use, where do we find the requirements for the deliverables for that type of process? I can provide examples, but will be based on process
- How long is it expected to be? Full semester, please see the schedule on GitHub

# Project Questions so far

---

- Is the project research-based\* or are we doing a full integration of an idea of ours?
- To what extent are we going to have to write code for this project?
- Will be implementing/coding any of the project or focusing primarily on the design?
- For the project, will we have to have an application as a deliverable?
- Is this essentially designing a piece of software without actually coding it?

**Coding is *not* required for the project!**

- Requirements, Design, Test plan, presentations,...

**If you implement and demo  $n$  use cases for your project, you will get extra credit and/or automatic A for the project!**

---

# Requirements Analysis

---

Requirements Analysis

Use Cases

UML Use Case Diagrams

Sequence Diagram

---

# Learning Outcomes

---

By the end of the course, students should be able to:

- **Understand software engineering processes, methods, and tools used in the software development life cycle (SDLC)**
- **Use techniques and processes to create and analyze requirements for an application**
- Use techniques and processes to design a software system
- Identify processes, methods, and tools related to phases of the SDLC
- Explain the differences between software engineering processes
- Discuss research questions and current topics related to software engineering
- Create and communicate about the requirements and design of a software application

# Requirements

**Goal:** Understand customer requirements for the software system

- The *what* of the project
- Very difficult to “get right” the first time and evolve over the course of development
  - Remember the Top 3 reasons for project failure:  
(2) Incomplete and (3) Changing Requirements
- *Software Artifacts:* requirements documents, use cases, user stories,...

# What are requirements?

---

**Definition:** Capabilities and conditions to which the system — and more broadly, the project — must conform. [Larman]

- Focusing on the **WHAT** not the **HOW**
- **Should always come first in the SDLC**



# Analysis vs. Specification

---

- **Analysis:** process of *understanding* the problem and the requirements for a solution
  - **Specification:** process of *describing* what a system will do
- *Analysis leads to Specification* – they are not the same!

# Requirements Analysis is Hard

---

- Major causes of project failures
  - Lack of user input
  - Incomplete requirements
  - Changing requirements
- **Essential requirements analysis solutions**
  - Classification of requirements Functional vs Non-functional
  - Iterative requirements analysis Iteration planning and elaboration
  - Use Cases Today!

# Running Example

---

From the Course Project Ideas List

- **Standup Bot:** A software bot to automatically schedule standup meetings between teammates.
  - *Scrum master:* Professional to ensure scrum processes for development team

## TODO: Complete a stand-up meeting!

- What I did.
- What I need to do next.
- What is blocking me.

# Use Cases

---

- **Definition:** a written description of using the system to fulfill stakeholder goals.
- **Stakeholders:** Anyone who supports, benefits from, or is affected by a software project that has direct or indirect influence on its requirements.
  - Managers, software engineers, users, clients, marketing, system administrators, testers, etc.

# Role of Use Cases

---

- Use case modeling is the most widely used approach for *requirements analysis*.
  - Useful for providing input into many subsequent software engineering activities and artifacts.
- Different levels of formality
  - Can be one paragraph for main scenario or multiple paragraphs covering steps developed iteratively
  - Shall (== *is required to*): used to indicate mandatory requirements in which no deviation is allowed
  - Should (== *is recommended that*): used to indicate
    - among several possibilities one is particularly suitable, without mentioning or excluding others
    - that a certain course of action is preferred but not necessarily required;
    - or (in the negative form) a certain course of action is not prohibited
  - May (== *is permitted to*): used to indicate a course of action permissible within the limits of the system
  - Can (== *is able to*): used for statements of possibility and capability

# Levels of Formality

---

- Brief: one-paragraph, for the main success scenario
- Casual: multiple paragraphs that cover several scenarios
- Fully dressed: all steps and variations
  - Developed iteratively during elaboration; the product of requirement analysis

# Black Box Use Cases

---

- Do NOT describe the internal workings of the system
  - Only system responsibilities
  - Focus on “what” the system should do

Good: “The system records the meeting time”

Bad: “The system generates SQL INSERT statement for the sale”

# Use Case Terms

---


- **Actor:** something with the behavior
  - Person, computer system, organization
  - Primary, supporting, offstage (interest in behavior)
- **Scenario (use case instance):** a specific sequence of actions and interactions between actors and the system.
- A use case is a collection of related success and failure scenarios that describe the actor using a system to support a goal.



# Writing a Use Case

---

**Preconditions:** What *must always* be true

- Generally, something noteworthy or interesting (i.e. not *“the user has power”*)
- Can include actors, stakeholders, etc.
- Often the postconditions of another use case
- Don't bother with it unless you are stating something noteworthy
  - *“The system has power”* is not interesting
  - *“Scrum master is identified and authenticated”* 

# Writing a Use Case (cont.)

---

**Postconditions:** State what must be true on successful completion of the use case—either the success scenario or alternative ones

- Success guarantee
- Meeting is requested by authorized user, Date is correctly calculated, Bot has access to developer schedules, Scrum Meeting recorded, Calendar invitation is generated and sent to team,...

# Main success scenario (Basic Flow)

---

- Defer all conditional and branching statements to an Extension section
- Records three kinds of steps:
  1. An interaction between actors
  2. A validation (usually by the system)
  3. A state change by the system

**Main Success Scenario:**

1. Scrum master accesses bot with meeting to schedule
2. Scrum master requests to schedule new meeting
3. Developers accept time proposed by bot based on schedule availability
4. System records the meeting, presents calendar invitation to development team

# Extensions (or Alternative Flows)

---

- Often comprise the majority of text
- Indicate all the other scenarios or branches, both success and failure
- Notated with respect to its corresponding steps 1...N in the main success scenario.

# Example: Stand-Up Bot

---

## Use Case: Create a meeting

### 1 Preconditions

User must have google calendar api tokens in system.

### 2 Main Flow

User will request meeting and provide list of attendees [S1]. Bot will provide possible meeting times and user confirms [S2]. Bot creates meeting and posts link [S3].

### 3 Subflows

[S1] User provides /meeting command with @username,@username list.

[S2] Bot will return list of meeting times. User will confirm time.

[S3] Bot will create meeting and post link to google calendar event.

### 4 Alternative Flows

[E1] No team members are available.

...

# Special Requirements

---

- If a non-functional requirement relates especially to a user case, record it with the use case

## *Special Requirements:*

- Bot must respond within 30 seconds 90% of the time.
- System must robustly recover when access to Google Calendar API fails

# Technology and Data Variations List

---

- Technical variations in “how” something must be done
  - Early design decisions or constraints
- Technical constraints imposed by stakeholders about input/output technologies.
  - Try to ***avoid*** premature design decisions unless they are obvious or unavoidable

Ex) Data scheme variations are necessary for understanding

# Technology and Data Variations List

---

## **Sample Technology and Data Variations examples:**

- a. Item identifier is entered by keyboard or voice input.
- b. Date may be in EST, EDT, UTC,... time zone.
- c. Calendar invitation is sent to Google Calendar, Microsoft Outlook Calendar,...



# Recap: Questions to Answer

---

- Who are the actors?
- What is the casual use case description?
  - Typically one success scenario
- What are alternative scenarios?
- What are dependent non-functional requirements?

# Unified Modeling Language

---

- Definition: A visual language for specifying, constructing, and documenting the artifacts of systems
  - Standard diagramming notation for drawing pictures related to software
  - Includes 13 types of diagrams

# UML Categories

---

- **Structural UML diagrams**
  - Specifies the structure of the objects, classes or components
    - Class diagram
    - Object diagram
    - ...
- **Dynamic UML diagrams**
  - Represent object interactions at runtime
    - Use case diagram
    - Sequence diagram
    - ...

# UML In Class

---

In Class, we will discuss:

- **Use case diagram (Requirements)**
- **Sequence diagram (Requirements & Design)**
- Class diagram (Requirements & Design)
- Package diagram (Design)

# Use Case Diagrams

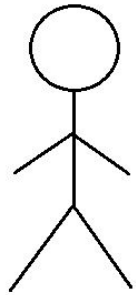
---

- A representation of interactions between actors and the system
  - It shows relationship between actors, use cases, and the system
    - the scope of the system
    - the external actors
    - how actors use the system
- Also can vary in formality
- It is *secondary* to text documentation

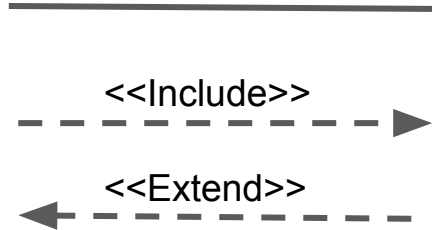
# Use Case Diagrams

---

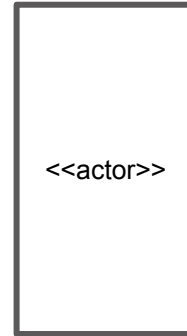
- Graphical depiction of use cases
  - Defines *actors'* interactions with the *system*



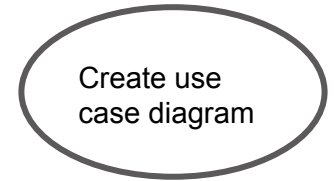
Actor



Interactions



System

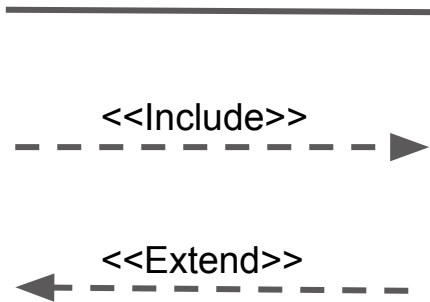


System Behavior

# Use Case Diagrams (cont.)

---

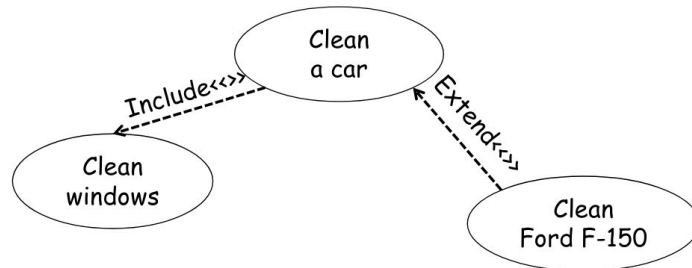
## Interactions



**Association:** relation between an actor and a use case

**Includes dependency:** a base use case includes a sub use case as component

**Extends dependency:** a use case extends the behavior of a base use case.



# Use Case Diagrams (cont.)

---

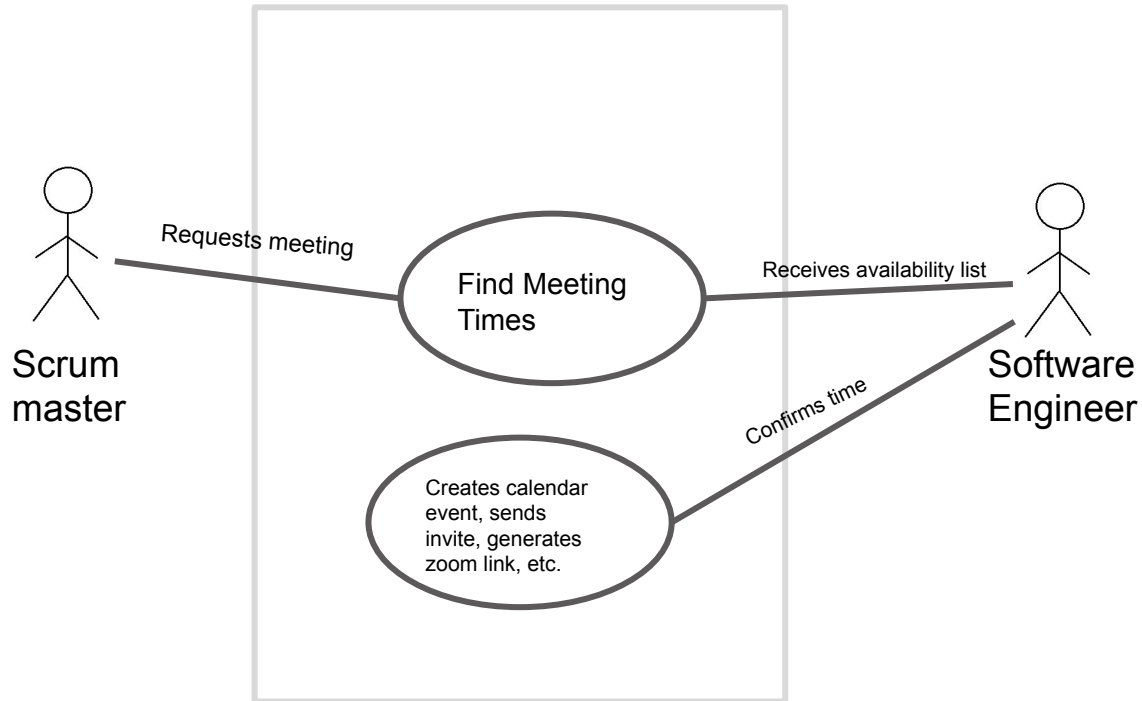
When creating use case diagrams, consider:

- What are the actors?
- What are the use cases in the system?
- What is the relationship between use cases?

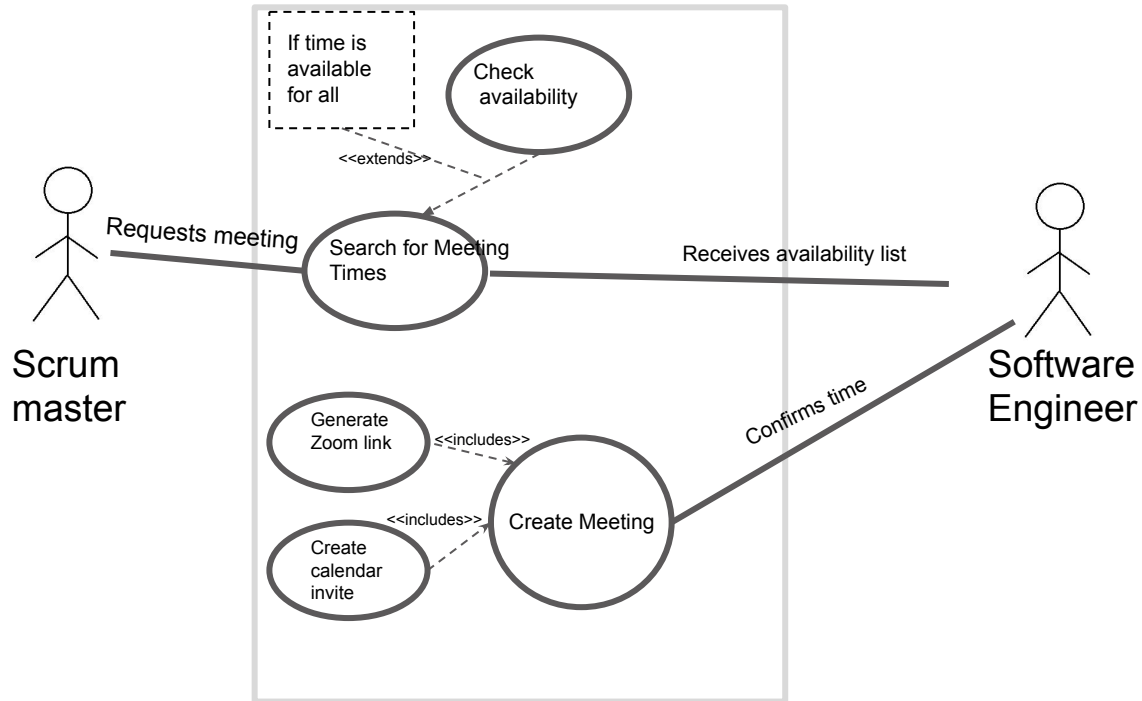


# Example: Stand-Up Bot

---



# Example: Stand-Up Bot (cont.)



# Sequence Diagrams

---

- A picture that shows, for a given use case, the events that external actors generate, their order, and inter-system events
  - Basic flow + frequent/complex alternatives
- All systems are treated as a black box, focusing on *WHAT* instead of *HOW*
- Generated from inspection of a use case
  - Illustrate input and output events related to the system, Emphasize events across boundaries between actors and systems
- Input to OOD

# UML Sequence Diagram

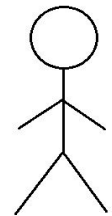
---

- A notation to illustrate actor interactions and operations initiated by them
- Only the interaction between users and the system is modeled in system sequence diagram
  - Allows for more advanced computational interaction (*i.e.*, conditional statements, loops, etc.)

# UML Sequence Diagram (cont.)

---


- Graphical Depiction



Actor

A rectangular box representing an object in a UML sequence diagram.

Object

A tall, narrow rectangular box representing an activation bar in a UML sequence diagram.

**Activation box:**  
represents time  
needed to for object to  
complete a task

**[condition]**

message/data →

← Return message

# Use Case vs. Sequence Diagrams

---

Sequence diagrams can:

- Present the *behavior* of the code
- Describes how—*and in what order*—a group of actors or objects work together.

# Revisit Stand-Up Bot Use Case

---

## Use Case: Create a meeting

### 1 Preconditions

User must have google calendar api tokens in system.

### 2 Main Flow

User will request meeting and provide list of attendees [S1]. Bot will provide possible meeting times and user confirms [S2]. Bot creates meeting and posts link [S3].

### 3 Subflows

[S1] User provides /meeting command with @username,@username list.

[S2] Bot will return list of meeting times. User will confirm time.

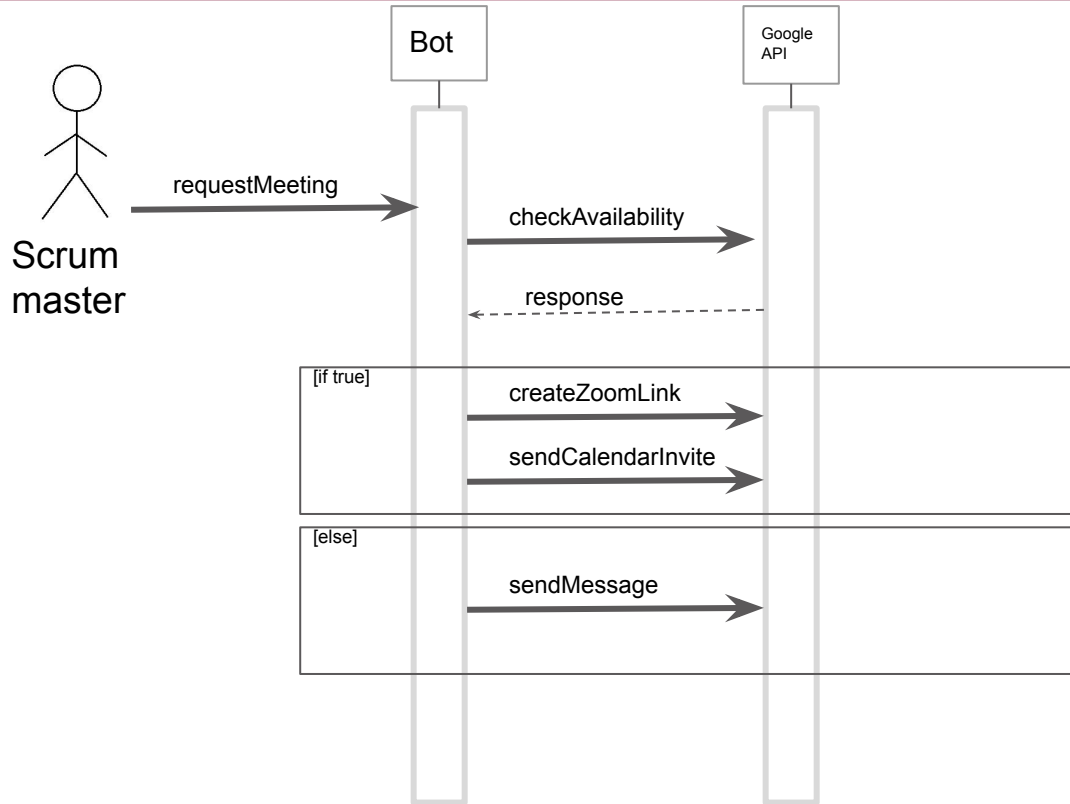
[S3] Bot will create meeting and post link to google calendar event.

### 4 Alternative Flows

[E1] No team members are available.

...

# Example: Stand-Up Bot





# Attendance Check

---

Please complete the following quiz:

<https://forms.gle/vGWikKdfcjmjhaLz9>

# Next Class

---

- **Requirements Workshop on Friday (9/15)**
  - Work with project group
  - To be turned in with PM2
- **PM1 due Friday (9/22)**
  - Lightning Talk slides
  - Design proposal

# Domain Models

---

- A visual representation of conceptual classes or real-situation objects showing:
  - Domain objects or conceptual classes
  - Relationship between conceptual classes
  - Attributes of conceptual classes
- Illustrated with a set of UML *Class diagrams*

# Role of Domain Models

---

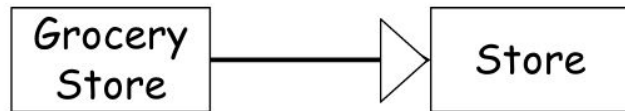
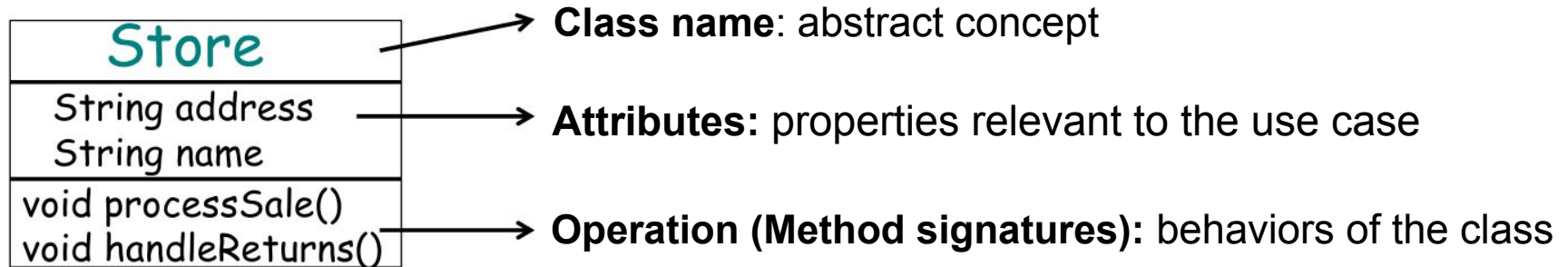
- Build upon use cases
  - More emphasis on relationship
- Basis for design and implementation
  - Starting to move towards requirements specification, and design
- The most important and classic model in OO analysis

# Class Diagram

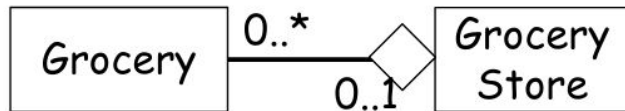
---

- A visual representation of main objects and their relations for a system.
- Elements
  - Classes containing: Attributes, Operations
  - Various relationships: Association, Aggregation, Composition, Generalization

# Class Diagram Example



**Generalization:** “is-a” relationship. A sub-class inherits all attributes and operations of its super class.



**Aggregation:** “has-a” relationship. The container and elements can exist independently from each other

# Building a Class Diagram

---

- Step 1: Identify conceptual classes
- Step 2: Decide attributes
- Step 3: Identify associations between classes

**Note:** Step 1 and 2 may occur together, iteratively

# 1. Identify Conceptual Classes

---

- Reuse or modify existing partial models created by experts
  - “recipes” for well-known problems and domains (e.g., accounting, stock market, ...)
- Consider common categories
- Identify nouns and noun phrases from fully dressed use cases
  - Ex) **ScrumMeeting**



## 2. Decide Attributes

---

- Properties of the conceptual classes relevant to the problem domain
  - Nouns and noun phrases that the requirement suggest or imply a need to remember
- *E.g.*, date, time, location, members, etc. are all relevant to a **ScrumMeeting** conceptual class

### 3. Identify Associations

---

- Relationship between instances of conceptual classes
- Think of it as a mathematical relation
  - Typically a binary relation:  $R \subseteq S1 \times S2$ 
    - $S1$  = set of instances of the first class
    - $S2$  = set of instances of the second class

# Typical Associations

---

- A is a physical/logical part of B
- A is physically/logically contained in B
- A is recorded/reported/captured in B
- A is a description of B
- A uses or manages B
- A is related to a transaction B
- A is owned by B...

# Key Points on Class Diagram

---

- UML is just annotation
- UML class diagram means different things in different contexts
  - Conceptual perspective: description of the domain model
  - Specification perspective: description of software abstractions or components
  - Implementation perspective: description of Java classes
- A small set of UML class diagram elements
  - Classes: Attributes, Operations
  - Relationship: Generalization, Aggregation, Composition, Association