COMP 3111
SOFTWARE ENGINEERING

LECTURE 6
SYSTEM REQUIREMENTS CAPTURE

LEARNING OBJECTIVES

1. Understand what is requirements capture and its role in the software development process.

2. Understand why requirements capture is both important and difficult.

3. Know the major activities that take place during requirements capture.

4. Learn how to use the UML to model a system's data and functional requirements.

5. Understand the importance of validating system requirements.

# SYSTEM REQUIREMENTS CAPTURE OUTLINE

System Requirements Capture Overview

– Life-cycle Role

– Importance of Requirements Capture

– Why Requirements Capture is Difficult

System Requirements Capture Activities

– Capture Data Requirements: Domain Modeling

– Capture Functional Requirements: Use-Case Modeling

– Capture Nonfunctional Requirements

– Validate System Requirements

REQUIREMENTS CAPTURE LIFE CYCLE ROLE
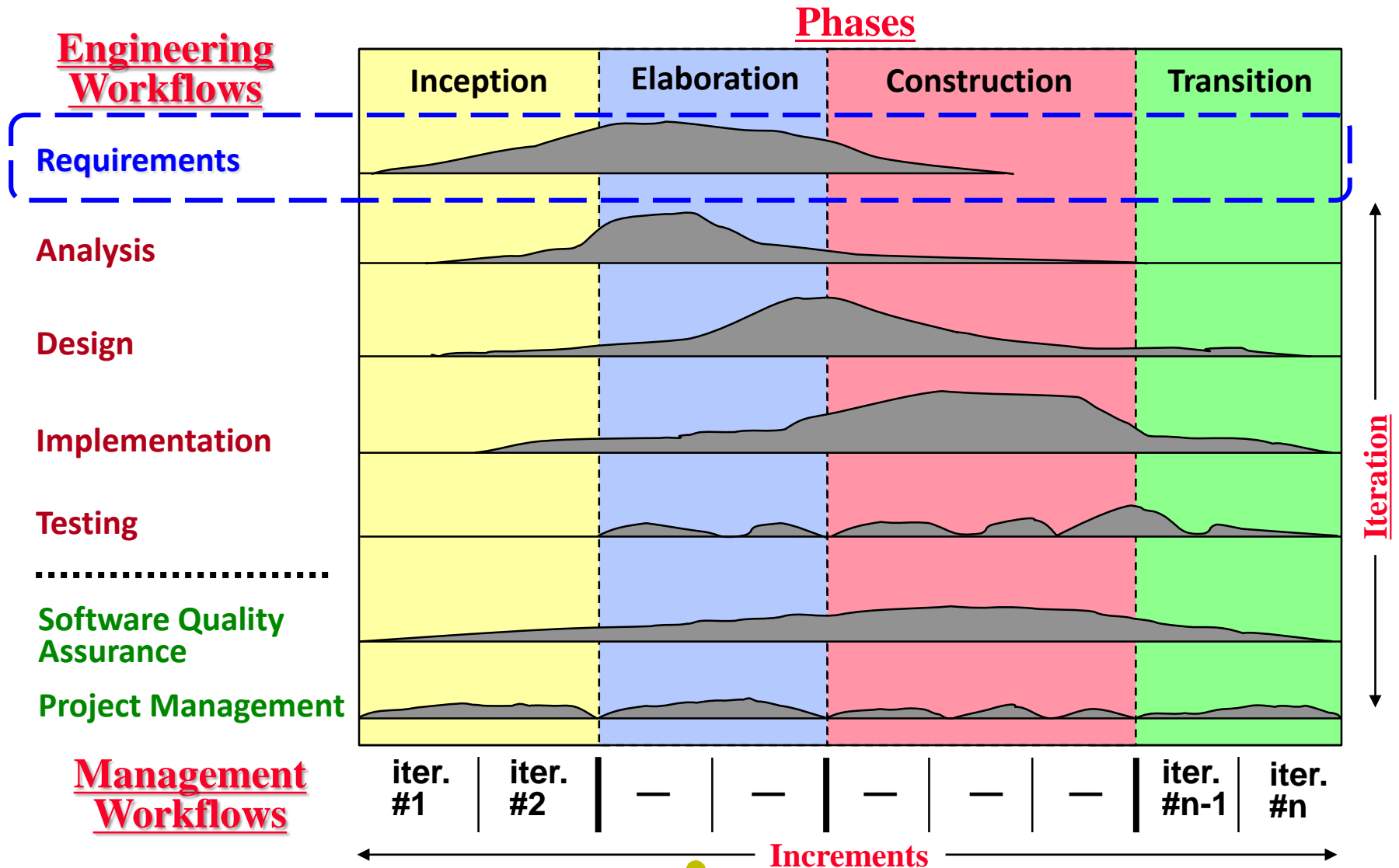
WHAT IS A REQUIREMENT?

> **A *requirement* is a feature that the system must have or a constraint that it must satisfy to be accepted by the client.**

- "Something required; something wanted or needed." *Webster's Ninth New Collegiate Dictionary*

- "a condition or capability need by a user to solve a problem or achieve an objective." *IEEE Standard 610.12*

- A complete statement of **what** the system will do *without* referring to **how** it will do it.

> **Can range from a high-level abstract statement of a service or a system constraint to a detailed mathematical specification.**

TYPES OF REQUIREMENTS

User requirements

- Statements in natural language, possibly with diagrams, of the services the system provides and its operational constraints.

- Written mainly for clients.

System requirements

- A structured document setting out detailed descriptions of the system's functions, services and operational constraints.

- Defines what should be implemented so may be part of a contract between the client and the developer.

- Written for both clients and developers.

PURPOSE OF REQUIREMENTS CAPTURE

Requirements capture **specifies the behaviour of the final software system.**

Requirements capture (gathering, elicitation, ...) involves:

- Learning about the problem that needs a solution.

- Specifying (in detail) the required features and constraints of the system in a way that the client/user understands and can approve.

☞ **REMEMBER:** Requirements **specify the problem**, **not the solution**.

The results of requirements capture also help in project planning.

IMPORTANCE OF REQUIREMENTS CAPTURE

Why do we need to capture and document requirements?
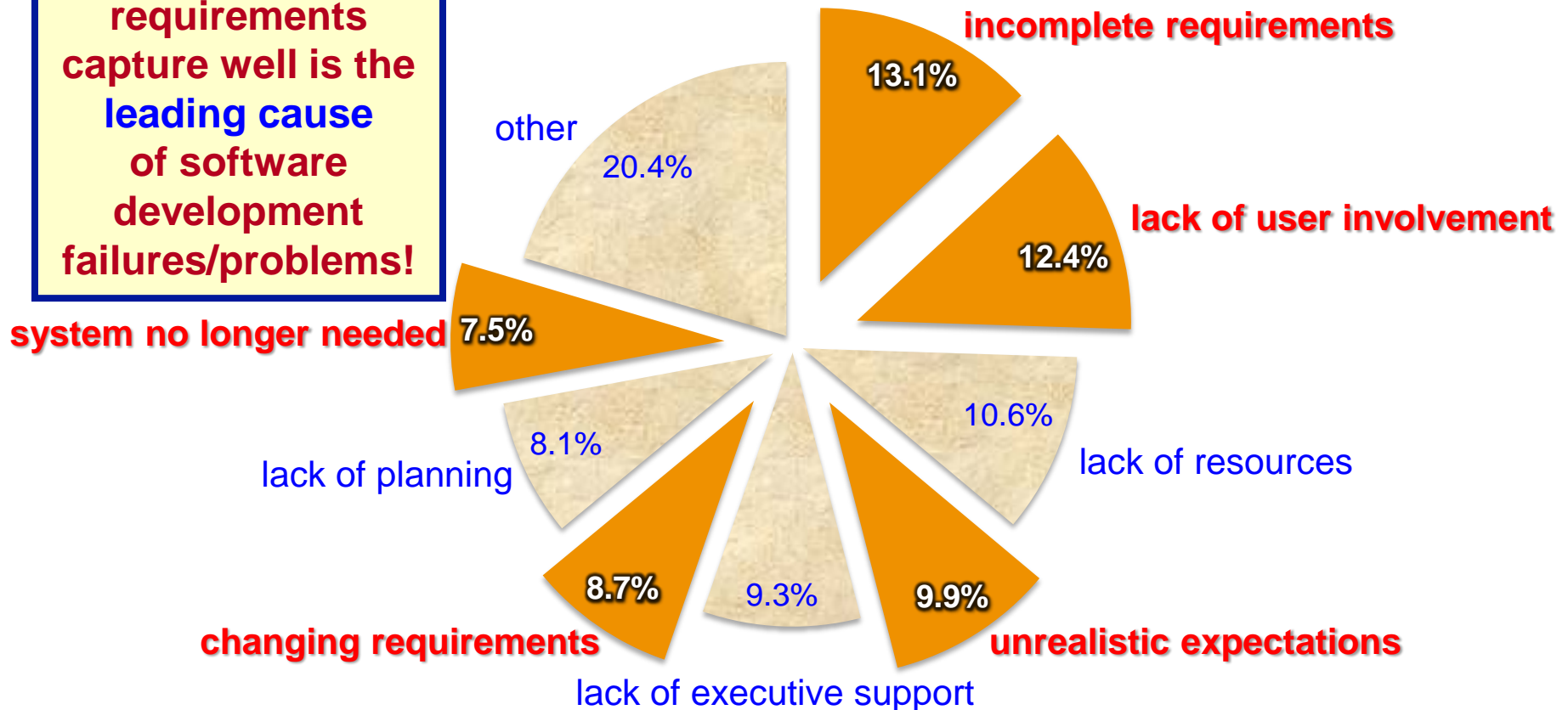
- Our brain is optimistic!

  – Think of a night dream.

  – You think you remember everything.

  – But when you try to remember details, you can't!

- The same thing happens in software development.

  – You think you know every detail.

  – But, during implementation you find many things are undefined or ambiguous.

Capturing and documenting requirements reduces errors!
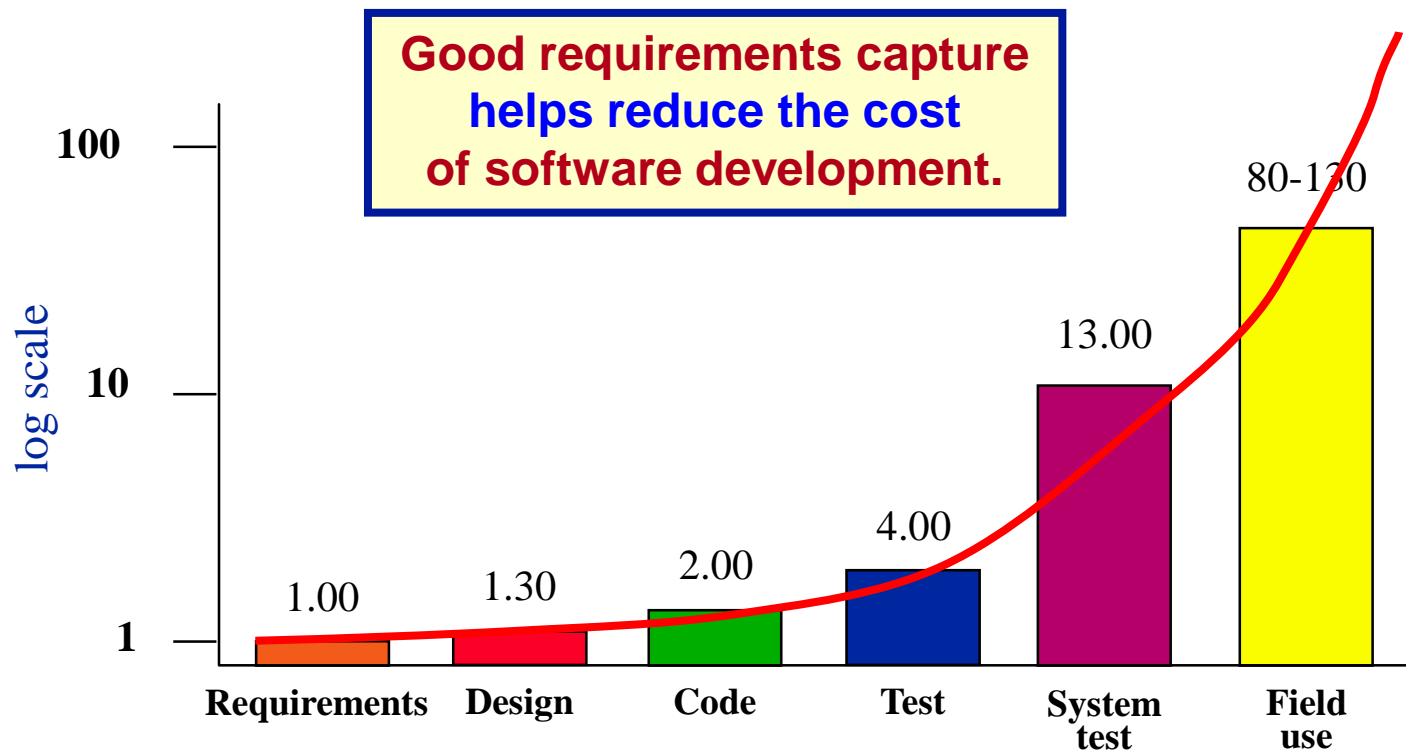
IMPORTANCE OF REQUIREMENTS CAPTURE

Reasons for software development failures/problems

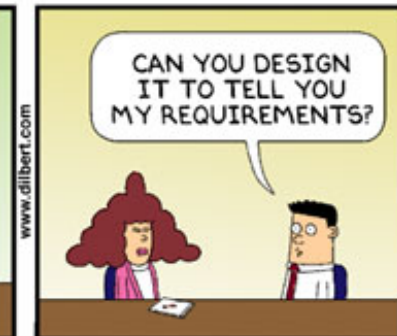**Failure to do requirements capture well is the leading cause of software development failures/problems!**

other 20.4%

incomplete requirements 13.1%

lack of user involvement 12.4%

system no longer needed 7.5%

lack of resources 10.6%

lack of planning 8.1%

changing requirements 8.7%

lack of executive support 9.3%

unrealistic expectations 9.9%

IMPORTANCE OF REQUIREMENTS CAPTURE

Cost to find and fix a defect

**Good requirements capture
helps reduce the cost
of software development.**

WHY REQUIREMENTS CAPTURE IS DIFFICULT

WHY REQUIREMENTS CAPTURE IS DIFFICULT

Requirements capture requires the collaboration of several stakeholders usually with different backgrounds.

client/users

⇧

Knowledge Gap

⇩

software engineers
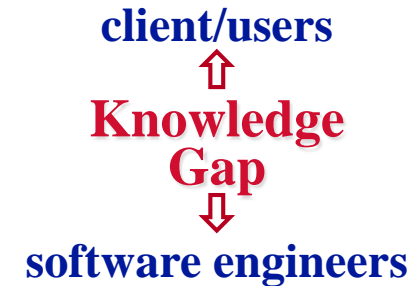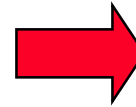
CHALLENGE: How to bridge the knowledge gap?

The software engineer needs to bridge the knowledge gap by:

– learning about the application domain and "discovering" requirements.

– transforming vague ideas into precise specifications.

– choosing an appropriate representation for specifying requirements.

☞ **Software engineer may need to "educate" the client/users!**

SYSTEM REQUIREMENTS CAPTURE OUTLINE

✓ System Requirements Capture Overview

– Life-cycle Role

– Importance of Requirements Capture

– Why Requirements Capture is Difficult

➡ **System Requirements Capture Activities**

– **Capture Data Requirements: Domain Modeling**

– **Capture Functional Requirements: Use-Case Modeling**

– **Capture Nonfunctional Requirements**

– **Validate System Requirements**

REQUIREMENTS CAPTURE ACTIVITIES

- **Understand the application domain and identify user needs**
  - collect data on system requirements and system constraints
  - define development effort scope and system design goals

- **Determine the risks of developing the system**
  - economic
  - technical
  - operational
  - organizational
  - legal

- **Capture the system requirements**
  - data requirements → domain model
  - functional requirements → use-case model
  - nonfunctional requirements → supplementary text

  Our focus

- **Validate the system requirements**
  - verify the correctness and completeness of the system requirements

SYSTEM REQUIREMENTS SPECIFICATION (SRS)

- The SRS documents the system requirements.

  – It is the official statement of what is required of the system.

  – It should include both a definition of user requirements and a specification of the system requirements.

- It is **NOT** a design document.

  – It should state **WHAT** the system should do, but not **HOW** it should do it.

NOTE: Many agile methods argue that producing a SRS is a waste of time as requirements change so quickly. However …

WAYS OF WRITING A SRS

● **Natural language**

– Uses sentences supplemented by diagrams and tables.

● **Structured natural language**

– Uses a restricted language that follows a fixed standard/template.

● **Graphical notations**   **UML + structured text**

– Uses graphical models plus text annotations.

● **Design description languages**

– Uses a language like a programming language.

● **Mathematical specifications**

– Uses notations based on mathematical concepts.

SYSTEM REQUIREMENTS CAPTURE OUTLINE

✓ System Requirements Capture Overview

– Life-cycle Role

– Importance of Requirements Capture

– Why Requirements Capture is Difficult

➔ **System Requirements Capture Activities**

☞ **Capture Data Requirements: Domain Modeling**

– **Capture Functional Requirements: Use-Case Modeling**

– **Capture Nonfunctional Requirements**

– **Validate System Requirements**

DOMAIN MODELING

● Captures the most important <u>classes</u> and their <u>associations</u>.

　☞ **Things that exist** or **events that occur** **for which data must be stored.**

● The classes and associations are found from user requirements:

　– requirements statement 　　– domain experts (users)

● The classes can be:

　– business objects (e.g., orders, accounts, etc.).

　– real-world objects and concepts (e.g., suppliers, customers, etc.).

　– events (e.g., aircraft arrival/departure, sales, reservations, etc.)

Described in a class diagram.

Provides a glossary of terms.

DOMAIN MODELING:
IDENTIFYING CLASSES AND ASSOCIATIONS

- Naturally occurring things or concepts in the user requirements:

    – classes appear as **nouns/noun phrases**

    – associations appear as **verbs/verb phrases**

    ☞ **Put all terms into singular form/active voice.**

- Identify only <u>relevant</u> classes/associations.

    ➤ Those that are *essential* and *will always exist*; <u>*are not transient*</u>.

    ☞ **This leads to a stable system.**

The decomposition of user requirements into classes and associations depends on judgment and experience and the nature of the problem.

There is usually no one correct decomposition!

DOMAIN MODELING: IDENTIFYING ATTRIBUTES

- Attributes usually correspond to <u>nouns</u> followed by possessive phrases

  e.g., password of student; student's address.

- Adjectives usually represent specific enumerated values

  e.g., Fall term; PhD degree.

- Only identify attributes that directly relate to the application domain.

> **Most attributes will not be given in a requirements statement and will have to be obtained from domain experts or application documentation.**

DOMAIN MODELING: EVALUATING CLASSES

- Are any classes irrelevant to the domain model? → eliminate
  - ☞ **A class should represent persistent data, not transient data.**

- Are any classes vague (ill-defined)? → make specific or eliminate
  - ☞ **A class should be specific and well-defined.**

- Are any classes redundant? → keep most descriptive

- Should any classes really be attributes? → consider application requirements
  - ☞ **Is the concept dependent or independent?**

- Do any class names describe a role?
  - ☞ **Roles also appear as nouns!**

- Do any classes describe an operation/action? → eliminate

- Do any classes describe implementation constructs? → eliminate

DOMAIN MODELING: EVALUATING ASSOCIATIONS

- Are any associations irrelevant to the domain model? → eliminate

- Do any associations describe an operation? → eliminate
  ☞ **Should describe a structural property, not a transient event.**

- Can ternary associations be decomposed into binary associations?
  → decompose
  ☞ **Should try to keep the model as simple as possible.**

- Are any associations derived associations? → eliminate
  ☞ **Introduces redundancy into the model.**
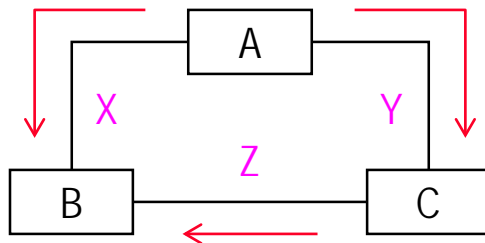
E.g., is association X ≡ associations Y and Z?

If yes, then we can derive X from Y and Z.

- Do any associations describe implementation constructs? → eliminate

DOMAIN MODELING: EVALUATING ATTRIBUTES

- Are attributes closely related to the class they are in?

  ☞ **A class should be simple and coherent.**

- Should any attributes really be classes?

  ☞ **The concept should be dependent on the class.**

- Should any attributes be in an association class?

  ☞ **Does the attribute depend on the existence of an association?**

- Have object identifiers been included as attributes?

  ☞ **Object identifiers should not be included in the domain model!**

DOMAIN MODELING: DETAIL

- For each *class* we specify its **attributes**.
  - Each class is assumed to have two standard operations, get and set, which *do not need to be specified*.

- For each *attribute* we specify
  - name (unique within a class)
  - type (e.g., string, integer, date, etc.)
  - multiplicity (if greater than 1)

- For each *association* we specify
  - name $\rightarrow$ should say "what" not "how" or "why"
  - role names (if needed)
  - multiplicity (if known)
  - association class (if needed)

SYSTEM REQUIREMENTS CAPTURE

DOMAIN MODELING EXERCISE