COMP 3111 SOFTWARE ENGINEERING

SYSTEM REQUIREMENTS CAPTURE PART 1

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 represent 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

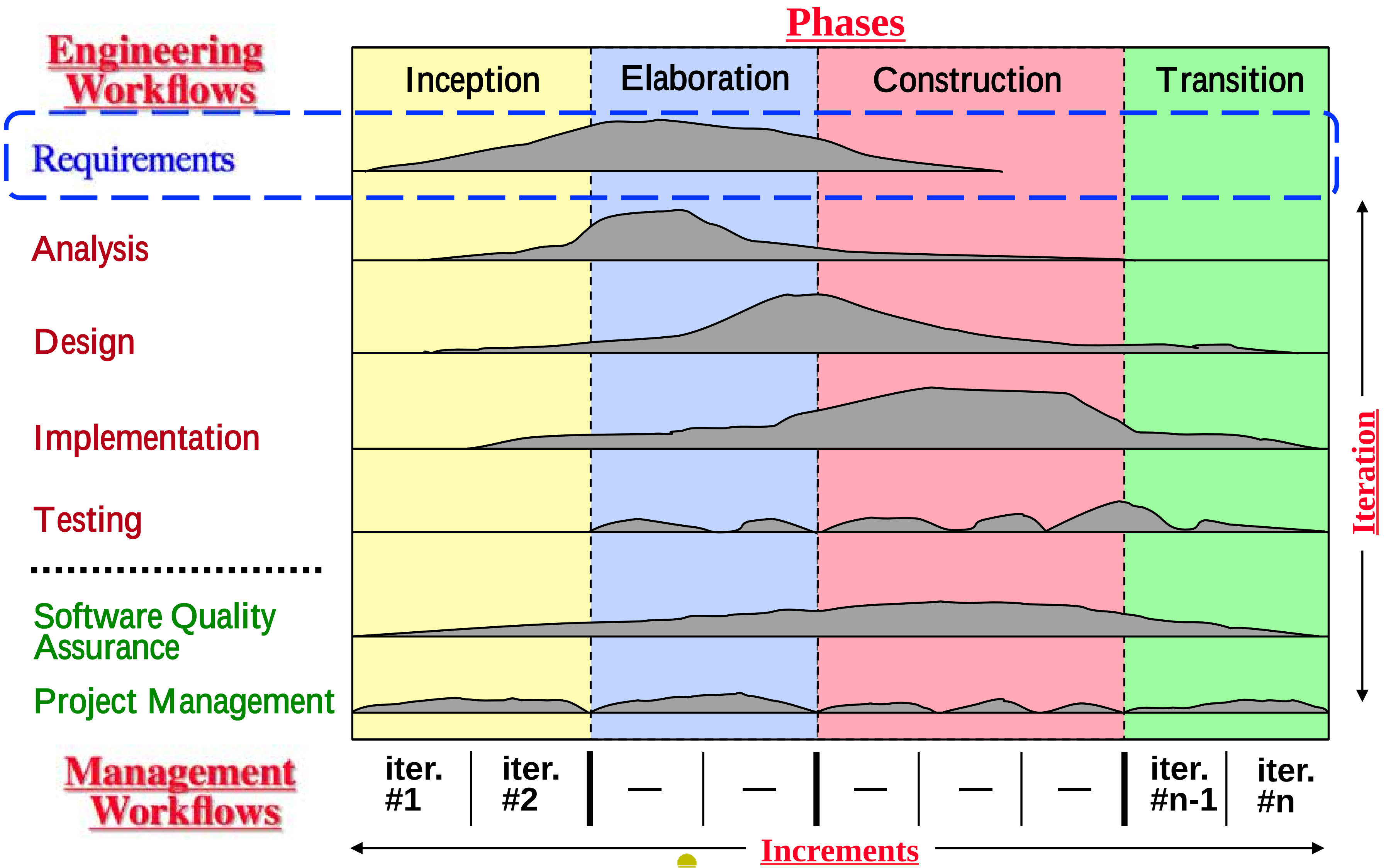
– Capture Data Requirements: Domain Modeling

– Capture Functional Requirements: Use-Case Modeling

– Capture Nonfunctional Requirements

– Validate System Requirements

REQUIREMENTS CAPTURE LIFE CYCLE ROLE



REQUIREMENTS CAPTURE LIFE CYCLE ROLE

- The system requirements are captured over several development increments—going first breadth and then depth (mainly during the inception and elaboration phases).

Requirements capture is "in focus" mainly here.

  ➔ Inception phase: identify the requirements at a high level to delimit the system, scope the project and estimate the development effort.

  ➔ Elaboration phase: capture the details of the requirements for implementation.

  – Construction phase: capture any missed requirements and/or clarify/correct existing requirements during system implementation.

  – Transition phase: there is no requirements capture unless there are changing requirements.

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 _____ the system will do *without* referring to _____ 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

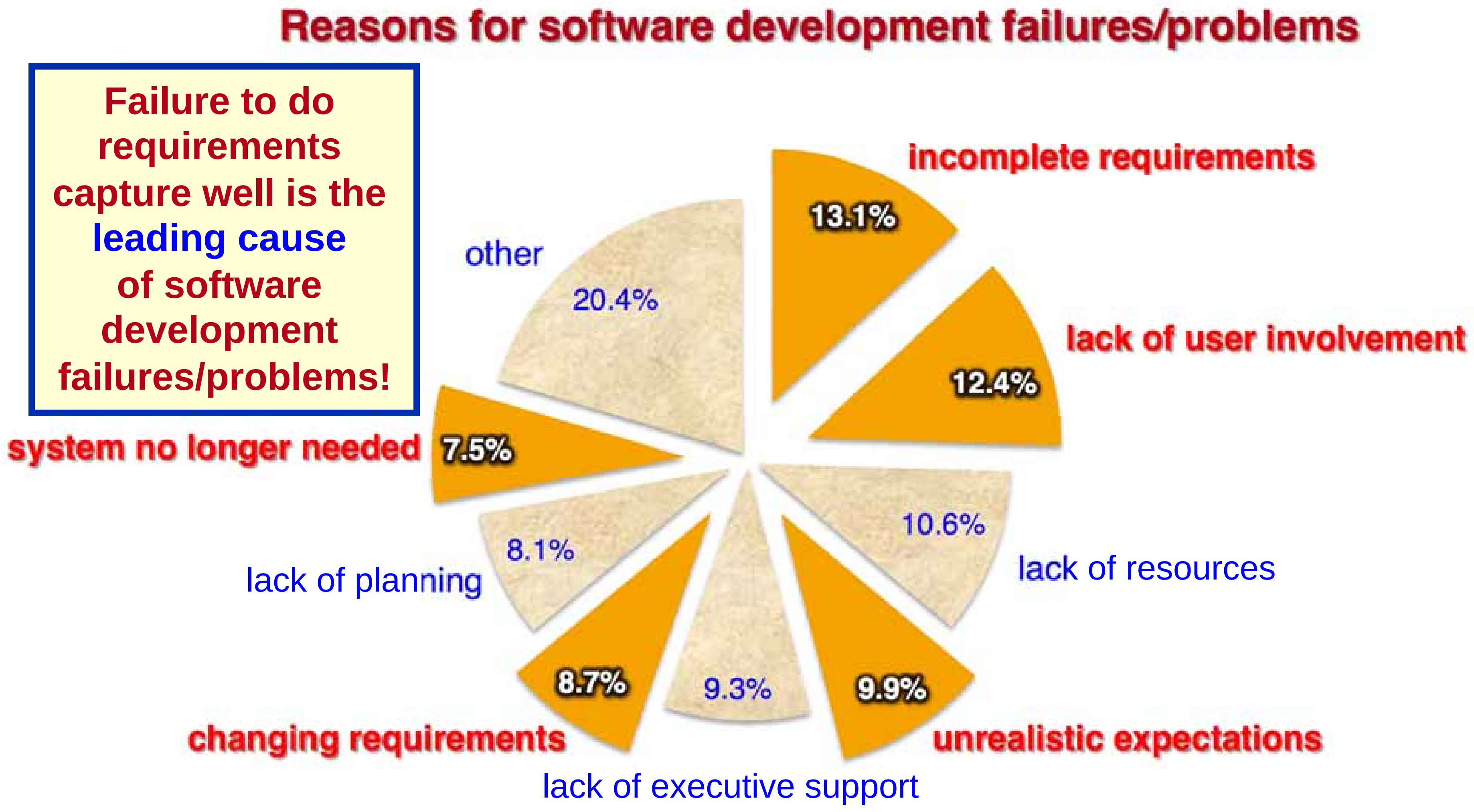
Why do we need to capture and document requirements?

- 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

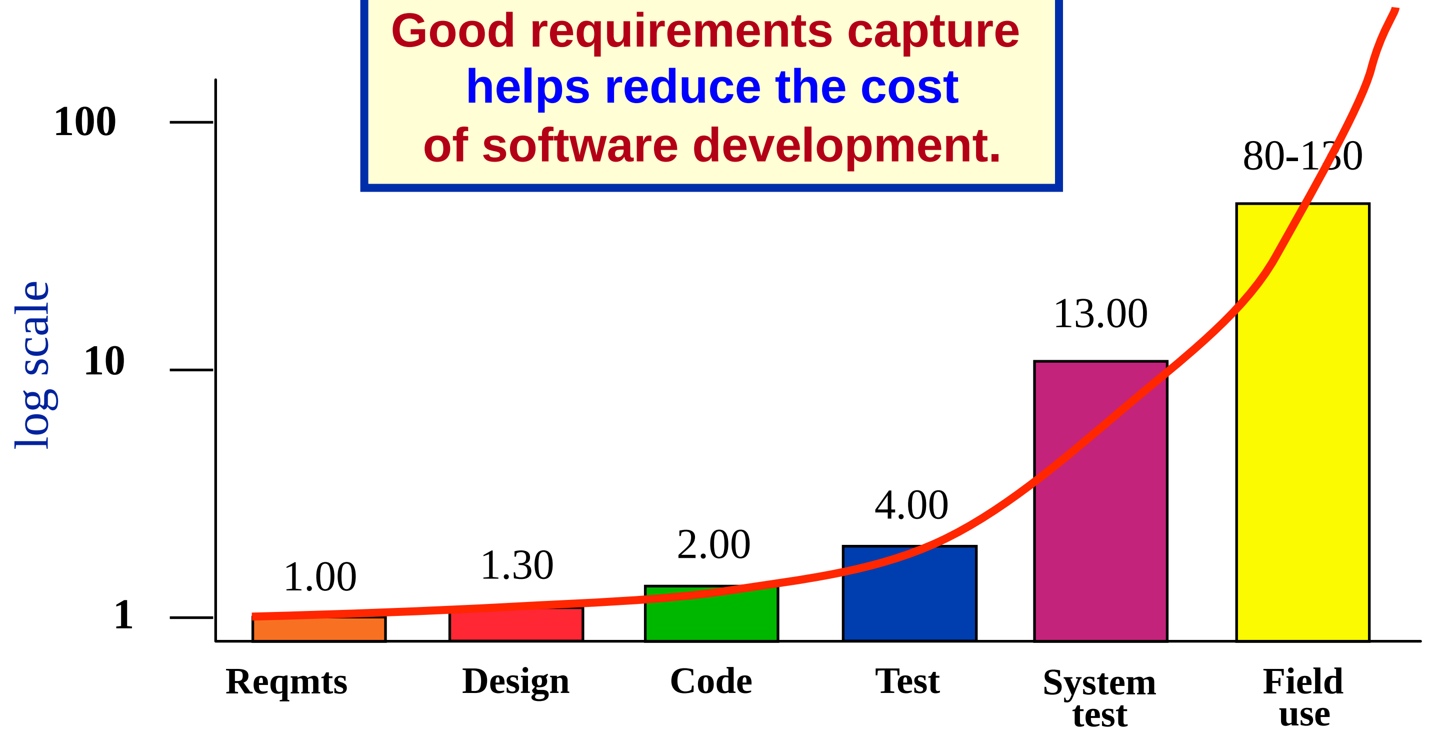
Failure to do requirements capture well is the **leading cause** of software development failures/problems!



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

Requirements capture requires the collaboration of several stakeholders usually with ____________________.

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!**

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
  - operational
  - legal
  - technical
  - organizational

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

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

Our focus

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.

DOMAIN MODELING

- Captures the most important _______ and their ___________.

  ☞ **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 _______ classes/associations.
  - ➢ Those that are *essential* and *will always exist*; *are not transient*.

☞ **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!

ASU Course Registration System

System Requirements Capture

Domain Model

Classes and Associations

The analysis and solution of this example will be posted in the Lecture Examples section of the course's web page *after* the lecture.

ASU COURSE REGISTRATION REQUIREMENTS

At the beginning of each term, students may request a course catalogue containing a list of course offerings needed for the term. Information about each course, such as instructor, department, and prerequisites are included to help students make informed decisions.

The new system will allow students to select four course offerings for the coming term. In addition, each student will indicate two alternative choices in case a course offering becomes filled or is canceled. No course offering will have more than forty students or fewer than ten students. A course offering with fewer than ten students will be canceled. Once the registration process is completed for a student, the registration system sends information to the billing system so the student can be billed for the term.

Instructors must be able to access the online system to indicate which courses they will be teaching, and to see which students signed up for their course offerings.

For each term, there is a period of time that students can change their schedule. Students must be able to access the system during this time to add or drop courses.

DOMAIN MODELING: EVALUATING CLASSES

- Are any classes irrelevant to the domain model?
- Are any classes vague (ill-defined)?
- Are any classes redundant?
- Should any classes really be attributes?
- Do any class names describe a role?
- Do any classes describe an operation?
- Do any classes describe implementation constructs?

ASU REQUIREMENTS: COURSE REGISTRATION

At the beginning of each term, students may request a course catalogue containing a list of course offerings needed for the term.

Information about each course, such as instructor, department, and prerequisites are included to help students make informed decisions.

The new system will allow students to select four course offerings for the coming term.

In addition, each student will indicate two alternative choices in case a course offering becomes filled or is canceled.

No course offering will have more than forty students or fewer than ten students.

ASU REQUIREMENTS: COURSE REGISTRATION

A course offering with fewer than ten students will be canceled.

Once the registration process is completed for a student, the registration system sends information to the billing system so the student can be billed for the term.

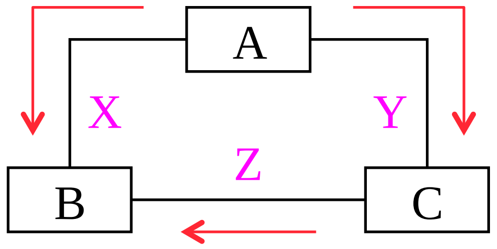
Instructors must be able to access the online system to indicate which courses they will be teaching, and to see which students signed up for their course offerings.

For each term, there is a period of time that students can change their schedule.

Students must be able to access the system during this time to add or drop courses.

DOMAIN MODELING: EVALUATING ASSOCIATIONS

- Are there any associations between eliminated classes?
- Are any associations irrelevant to the domain model?
- Do any associations describe an operation?
- Can ternary associations be decomposed into binary associations?
- Are any associations derived associations?



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?

DOMAIN MODELING: IDENTIFYING ATTRIBUTES

- Attributes usually correspond to ______ 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 ATTRIBUTES

- Are attributes closely related to the class they are in?
- Should any attributes really be classes?
- Should any attributes be in an association class?
- Have object identifiers been included as attributes?

DOMAIN MODEL 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 → should say "what" not "how" or "why"
  - role names (if needed)
  - multiplicity (if known)
  - association class (if needed)

DOMAIN MODELING EXAMPLE

The following are the requirements for a web-based system to computerize the management of the sale and rental of videos for a video shop.

- The system must be able to handle both physical and digital videos.
- It must be able to record which videos are sold and rented and by whom.
- For sold videos, the quantity sold should be recorded; for physical video rental, which copy is rented and when it is due back should be recorded.
- The system should keep track of overdue rentals of physical videos and send email notices to customers who have videos overdue.
- There will be a customer membership option for an annual fee, which will entitle a member to discounts (10%) on the sale and rental of videos.
- Members should be able to make reservations for physical video rentals either in person at the shop, by telephone or via the Web.
- A member can reserve at most five physical videos at any one time, but there is no limit on how many physical videos a member or nonmember can rent at any one time.
- As an added feature, the shop would like to allow customers (either members or nonmembers) to input, via the Web, mini-reviews (up to 100 words) and a rating (from 1, lowest, to 10, highest) of videos they have purchased or rented.

DOMAIN MODELING EXAMPLE (cont'd)

- These reviews should be anonymous if the customer so wishes (i.e., customers can specify whether they want their name to be made known when other customers browse the reviews).
- A sales clerk should be able to enter and update the following information about all customers (members or nonmembers): name, address, phone number, age, sex, and email address.
- Members are assigned a membership number by the shop when they become members and a password, which allows them to change their personal information and to buy and rent digital videos via the Web.
- The shop manager should be able to generate various reports on the sale and rental of videos.
- A sales clerk should be able to sell and rent physical videos and process the return of rented physical videos.
- When selling or renting physical videos, a sales clerk must be able to look up customer information and determine whether the customer is a member.
- A sales clerk must be able to enter basic information about a video (i.e., video id, title, leading actor(s), director, producer, genre, synopsis, release year, running time, selling price, and rental price).

DOMAIN MODELING EXAMPLE (cont'd)

From the video sale and rental shop requirements statement:

a) identify all the classes, attributes, association classes, associations, aggregations/compositions, generalizations and multiplicity constraints that are relevant to include in the domain model for the new system. ***(Only those that are explicitly given in or implied by the requirements statement should be included.)***

b) Construct a class diagram showing how the classes identified in (a) are related by associations, aggregations/ compositions and generalizations. Show the ***most likely multiplicities for all associations*****, making reasonable assumptions where necessary. If a multiplicity cannot be inferred from the requirements statement or common real-world domain knowledge, then indicate this with a "?".** ***Do not show the attributes of the classes in the class diagram.***

DOMAIN MODEL TO RELATIONAL SCHEMA REDUCTION: OVERVIEW

We need to reduce:

generalizations/specializations		
attributes	→	multivalued
classes	→	independent, dependent
associations	→	degree (e.g., unary, binary, ternary, etc.)
	→	cardinality constraints

Cardinality constraints in the UML
are reduced to
referential integrity constraints in the relational model.

RELATIONAL MODEL

The relational model represents the data for an application as a collection of tables.

Relational Model		Representation	Notation
Relation	⇔	table	$R(A_1, A_2, \ldots, A_n)$
Attribute	⇔	column	A_i
Domain	⇔	type and range of attribute values	$dom(A_i)$
Tuple / Record	⇔	row	
Attribute value	⇔	value in table cell	

RELATIONAL MODEL: PROPERTIES OF TABLES (RELATIONS)

- Records in a table are *not* considered to be *ordered*, even though they are represented in a tabular form.

- All attribute values are considered *atomic*.

  ☞ Multivalued attribute values are not allowed in tables, although they are permitted in the UML.

- The relational model requires every table to have a primary key, although relational DBMSs do not impose this requirement.

- A special *null* value is used to represent values that are:
  – *Not applicable* (phone number for a client that has no phone).
  – *Missing* (there is a phone number but we do not know it yet).
  – *Not known* (we do not know whether there is a phone number or not).

RELATIONAL MODEL: CONSTRAINTS

- **Entity integrity constraint**

  If X is a primary key of a table **R**, then X cannot contain null values.

- **Referential integrity (foreign key) constraint**

  Given two tables **R** and **S**, table **S** may reference table **R** via a set of attributes FK_R that forms the primary key K_R of table **R**.

  The attributes FK_R in table **S** are called a foreign key.

  The value of the foreign key FK_R in a record of table **S** must either be equal to the value of the primary key K_R of a record in table **R** or be entirely null.

$$\begin{tikzcd} \text{Project}(\underline{\text{proj\#}}, \text{name, budget}) & \text{Person}(\underline{\text{userId}}, \ldots, \text{manages-project}) \arrow[l, dashed, bend left] \end{tikzcd}$$

  The attribute manages-project in Person is a foreign key since it references the primary key proj# of Project.

DOMAIN MODEL TO RELATIONAL SCHEMA REDUCTION: REFERENTIAL INTEGRITY CONSTRAINTS

If table **S** contains the primary key K_R of table **R** as a foreign key FK_R, which can be specified as

FK_R references **R**

then the value of FK_R in a record of table **S** must either be equal to the value of the primary key K_R of a record in table **R** or be entirely null.

To enforce this constraint, the following actions are required.

For mandatory participation in the domain model

on delete cascade - Delete all records with foreign key values in **S** that match the primary key value of the deleted record in **R**.

For optional participation in the domain model

on delete set null - Set to null the foreign key value of all records in **S** whose foreign key value matches the primary key value of the deleted record in **R**.

GENERALIZATIONS / SPECIALIZATIONS

Option 1: Reduce all classes to tables.

- Create a table for each class (superclass and subclasses).
- For each table created for a subclass:
  - Add the primary key of the superclass into the table as a foreign key FK_S.
  - The foreign key FK_S becomes the primary key.
  - Add a foreign key constraint: FK_S references superclass table.
  - Add a referential integrity action: on delete cascade.

GENERALIZATIONS / SPECIALIZATIONS (cont'd)

Option 2: Reduce only subclasses to tables.

- Create a table for each subclass.
- For each table created for a subclass:
  - ➢ Add all the attributes of the superclass.
  - ➢ The primary key is the primary key of the superclass.

☞ **Should be used only for disjoint generalizations/specializations!**

MULTIVALUED ATTRIBUTES

For a multivalued attribute **M** in a class **S**:

- Create a table SM with an attribute **A** that corresponds to **M** and attribute(s), FK_S, corresponding to the primary key of class **S**.
- The primary key of table SM is the union of all its attributes.
- Add a foreign key constraint: FK_S references **S**.
- Add a referential integrity action: on delete cascade.

INDEPENDENT CLASSES

For each independent class **S** that has a primary key:

- Create a table R_S with all the attributes of class **S**.
- The primary key of table R_S is the primary key of class **S**.

DEPENDENT CLASSES

For a dependent class **T** that depends on another class **S** (**T** has no key):

- Create a table R_T with attributes of the dependent class **T**.
- Include attributes of association **R**, if any, in table R_T.
- Include as foreign key attribute(s) FK_S in table R_T, the primary key attributes of class **S**.
- The primary key of table R_T is the union of the foreign key attributes FK_S and the *discriminator* d_a of the dependent class **T**.
- Add a foreign key constraint: FK_S references **S**.
- Add a referential integrity action: on delete cascade.

ASSOCIATIONS

For each binary association **R**:

- Create a new table R_R.
- Include attributes of association **R**, if any, as attributes of table R_R.
- Include as foreign key attributes in table R_R the primary keys of all the classes related by association **R**.
- The primary key of table R_R is
  - *binary 1:1 association* → the primary key of either class.
  - *binary 1:N association* → the primary key of the class on the N-side of the association.
  - *binary N:M association* → the union of the primary keys of the two participating classes.
- Add a foreign key constraint for each foreign key.
- Add a referential integrity action for each foreign key: on delete cascade.

☞ **This reduction minimizes null values.**

TABLE COMBINATION

- Some tables representing associations can be combined with the table for a related class to reduce the number of tables.
  - For 1:1 associations, the table for the association can be combined with the table for either class.
  - For 1:N associations, the table for the association can be combined with the table for the class on the N-side.
  - Add a foreign key constraint for the foreign key.
  - Add a referential integrity action for the foreign key that depends on the participation constraint of the class on the N-side.