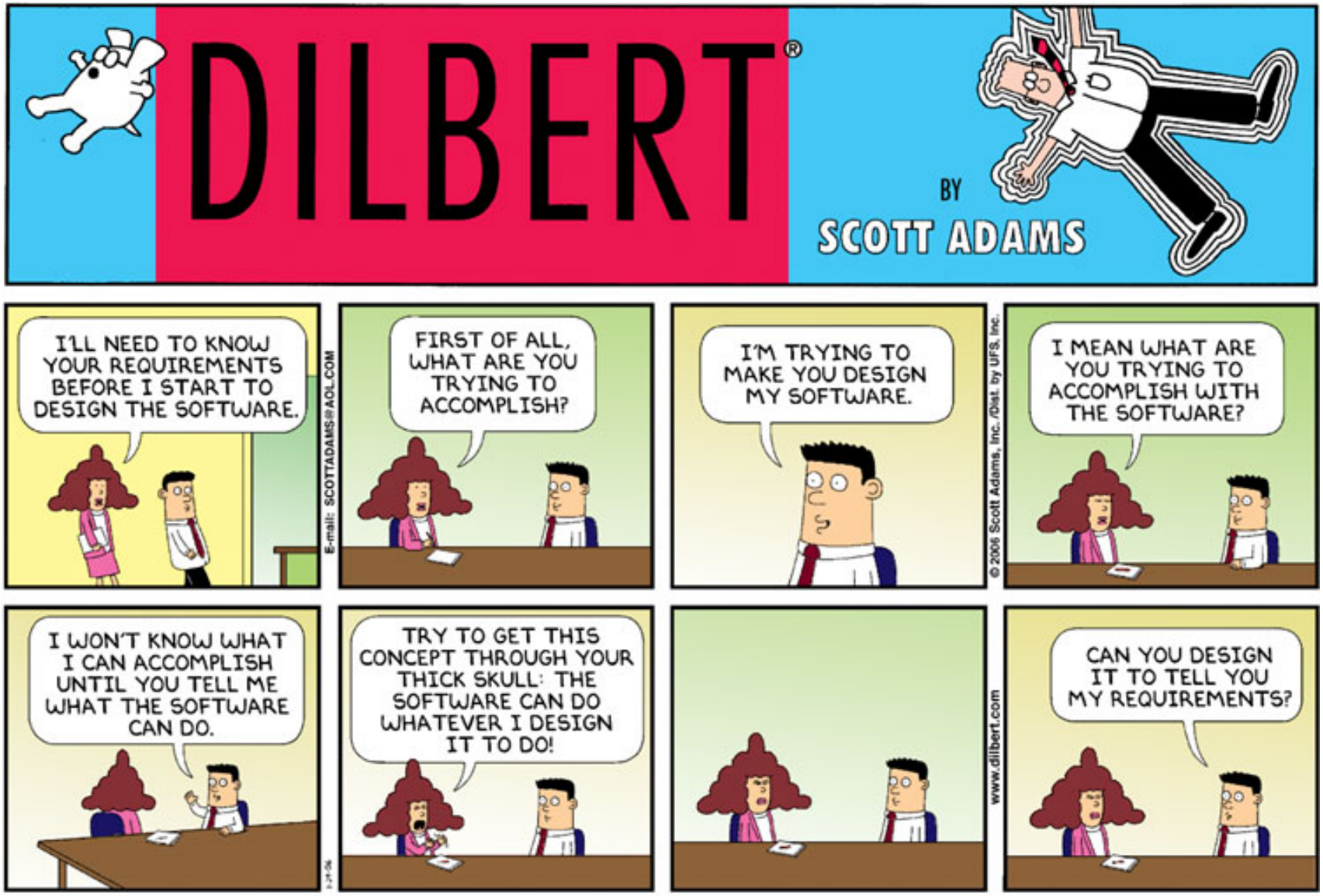
Requirements Capture

What exactly do we need to build?

Key Questions

- What is requirements? Why so many lectures?
- What do people do for requirement capture?
- Domain model? Use case?



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 729*
- A complete statement of **what** the system will do *without* referring to **how** it will do it.

PURPOSE

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

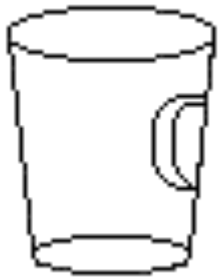
Requirements capture (gathering, elicitation, ...)

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

☞ **REMEMBER:** A requirement **specifies the problem**, **not the solution**.

REQUIREMENTS EXAMPLE: HOT DRINK

- Need the capability to drink hot liquids
- Requirements
  - The thing shall contain up to 8 ounces of hot liquid.
  - The thing shall have a handle.
  - **No details how to make the thing!**



http://www.jiludwig.com/What_vs_How.html

Declarative vs. Imperative (SQL vs. Java)

- Retrieving a list of names
- SQL
  - "select firstname, lastname, birthday from entries where firstname="charles" order by birthday"
- Java

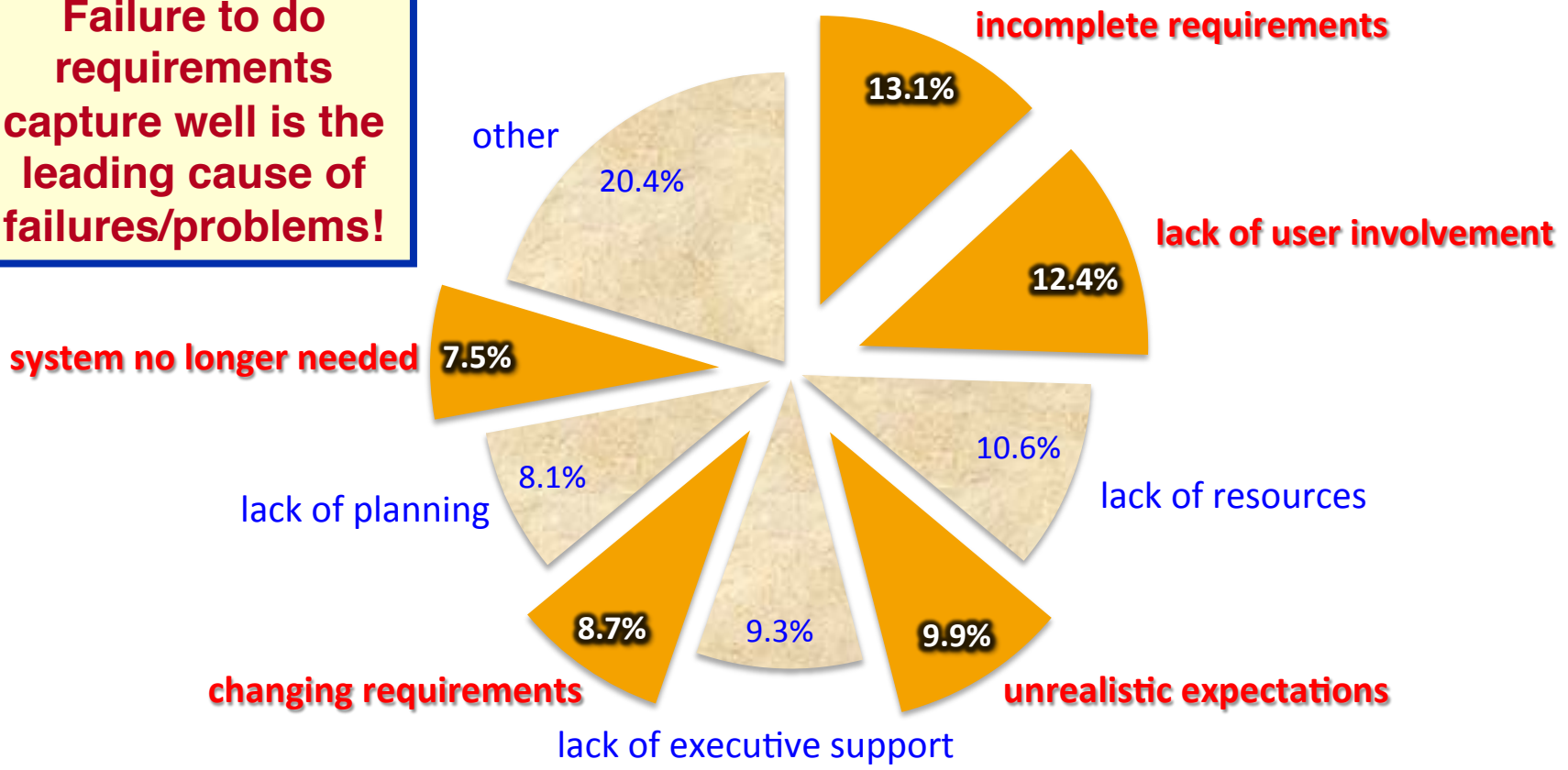
```
String[] retrieve(){
    SortedList list = new SortedList(new Comparator {
              public int compare(String bday2, String bday2){
           return bday2.compareTo(bday2);
               }
     );
     for(entry:entries){
         if(entry.firstname.equals("charles"))
        list.insert(entry);
     }
     return list.toArray();
}
```

THE IMPORTANCE OF REQUIREMENTS

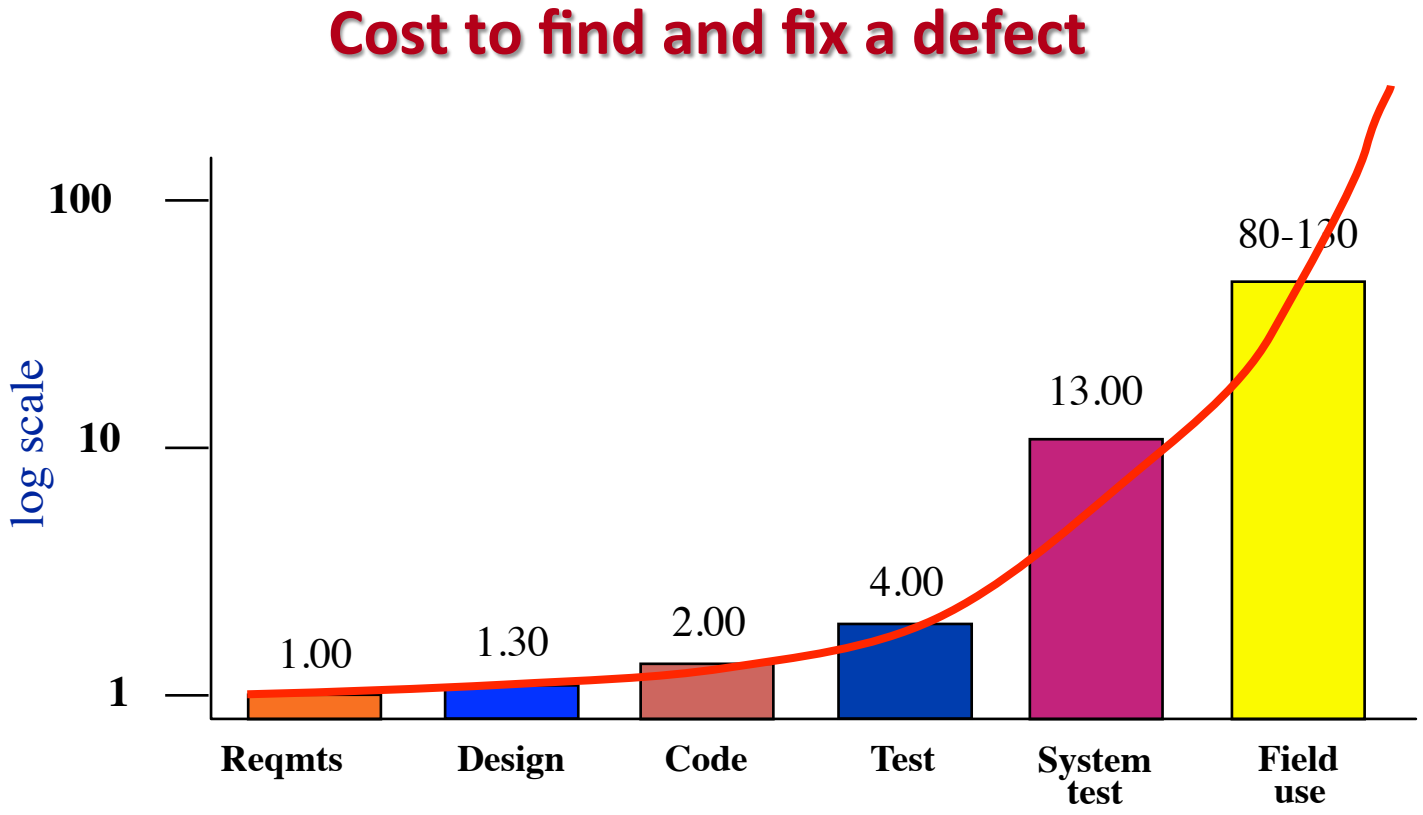
Requirements capture is important

Reasons for failure of or problems with software development

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

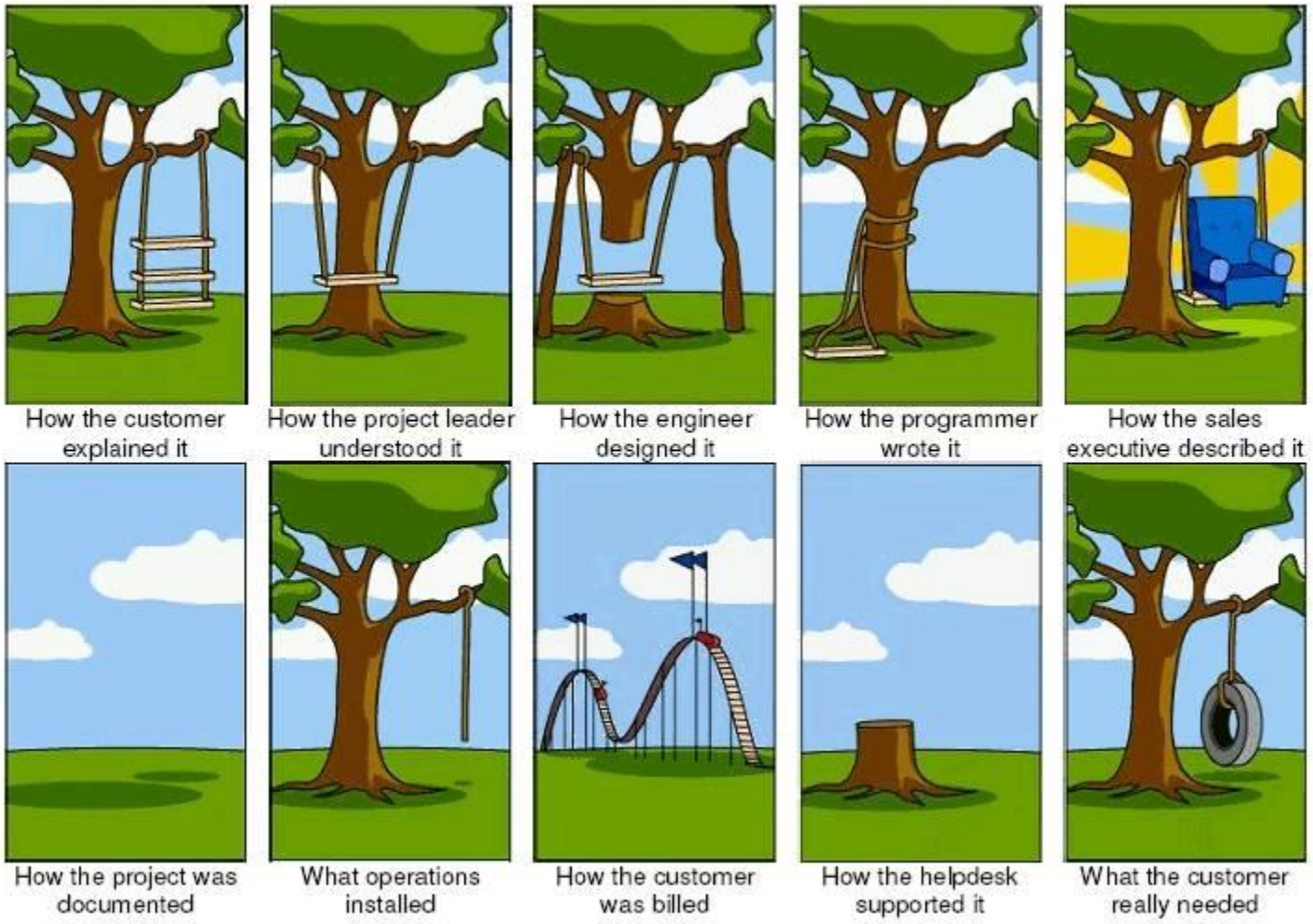


Requirements capture is important



Good requirements capture helps reduce the cost of software development.

Why requirements are difficult to capture?



Capture requirements – The UP way

- **Data requirements** (persistent data) → **domain model**
  - Describe the important concepts of the application domain.
    - Allows the development of a glossary of terms (data dictionary) for communicating among everyone on the project.

    Example: The system shall store information about products.

- **Functional requirements** (processing) → **use-case model**
  - Describe the interactions between the system and its environment independent of its implementation.

    Example: The system shall allow searching for products by brand.

- **Nonfunctional requirements** (implementation constraints) → **text**
  - Describe aspects of the system that may impose restrictions on the implementation (e.g., response time, throughput, security, implementation language, OS platform, hardware, interfaces, etc.)

    Example: The system should be up 99.999% of the time.

Domain Modeling

- Domain modeling captures the most important classes and their associations in the context of the system.

  ☞ **Things that exist or events that occur in the system's environment.**

- The classes and associations are found from:
  - problem statement.
  - domain experts (users).

  ☞ **It is imperative that this be done well so as to establish a solid foundation and to allow reuse by other systems.**

- 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.)

Provides a glossary of terms.

Described in a class diagram → static system requirements.

Identify classes and associations

- Naturally occurring things or concepts in the application domain:
  - classes often appear as **nouns/noun phrases**
  - associations often appear as **verbs/verb phrases**

☞ **Circle or highlight in the problem statement.**

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

- Identify only relevant classes/associations.
  - Those that are *essential* and *will always exist*; *are not transient*.

☞ **This leads to a stable system.**

The decomposition of a problem 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 on the course's Lecture Examples web page *after* the lecture.

ASU: COURSE REGISTRATION PROBLEM STATEMENT

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.

Professors 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.

ASU: POSSIBLE CLASSES

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

classes: Term, Student, CourseCatalogue, CourseOffering

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

classes: Course (Information), Instructor, Department, Prerequisite, InformedDecision

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

classes: System

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

classes: AlternativeChoice

ASU: POSSIBLE CLASSES

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

classes: *no new classes*

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

classes: *no new classes*

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.

classes: RegistrationProcess, Information, BillingSystem

Professors 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.

classes: Professor

ASU: POSSIBLE CLASSES

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

classes: PeriodOfTime, Schedule

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

classes: *no new classes*

ASU: POSSIBLE CLASSES

Term
Student
CourseCatalogue
CourseOffering
Course(Information)
Instructor
Department
Prerequisite
InformedDecision
System
AlternativeChoice
RegistrationProcess
Information
BillingSystem
Professor
PeriodOfTime
Schedule

ASU: POSSIBLE ASSOCIATIONS

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

associations: Student *Requests* CourseCatalogue
CourseCatalogue *Contains* CourseOffering
Term *Needs* CourseOffering

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

associations: CourseCatalogue *Includes* Course(Information)
CourseCatalogue *Helps* Student
Student *Makes* InformedDecision

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

associations: System *Allows* Student
Student *Selects* CourseOffering

ASU: POSSIBLE ASSOCIATIONS

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

associations: Student *Indicates* AlternativeChoice
? *Fills* CourseOffering
? *Cancels* CourseOffering

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

associations: CourseOffering *Has* Student

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

associations: *no new associations*

ASU: POSSIBLE ASSOCIATIONS

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.

associations:
Student *Completes* RegistrationProcess
System *Sends* Information *To* BillingSystem
BillingSystem *Bills* Student

Professors 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.

associations:
Professor *Accesses* System
Professor *Indicates* CourseOffering
Professor *Teaches* CourseOffering
Professor *Sees* Student
Student *SignsUpFor* CourseOffering

ASU: POSSIBLE ASSOCIATIONS

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

associations: Term *Has* PeriodOfTime
Student *Changes* Schedule

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

associations: Student *Accesses* System
Student *Adds* CourseOffering
Student *Drops* CourseOffering

ASU: POSSIBLE ASSOCIATIONS

Student *Requests* CourseCatalogue
CourseCatalogue *Contains* CourseOffering
Term *Needs* CourseOffering
CourseCatalogue *Includes* Course(Information)
CourseCatalogue *Helps* Student
Student *Makes* InformedDecision
System *Allows* Student
Student *Selects* CourseOffering
Student *Indicates* AlternativeChoice
? *Fills* CourseOffering
? *Cancels* CourseOffering
CourseOffering *Has* Student
Student *Completes* RegistrationProcess
System *Sends* Information *To* BillingSystem
BillingSystem *Bills* Student
Professor *Accesses* System
Professor *Indicates* CourseOffering
Professor *Teaches* CourseOffering
Professor *Sees* Student
Student *SignsUpFor* CourseOffering
Term *Has* PeriodOfTime
Student *Changes* Schedule
Student *Accesses* System
Student *Adds* CourseOffering
Student *Drops* CourseOffering

KEEPING THE RIGHT CLASSES

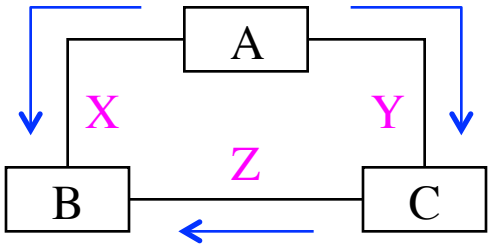
- Are any classes irrelevant to the domain model?
- Are any classes vague (ill-defined)?
- Are any classes redundant?
- Should any classes really be attributes?
- Do any class names describe a role?
- Do any classes describe an operation?
- Do any classes describe implementation constructs?

ASU: Keeping the right classes

Term	→	attribute (of CourseOffering)
Student	→	O.K.
CourseCatalogue	→	irrelevant (physical system generated entity)
CourseOffering	→	O.K.
Course(Information)	→	O.K.
Instructor	→	redundant (same as Professor)
Department	→	O.K.
Prerequisite	→	role (of Course)
InformedDecision	→	irrelevant (mental process of students)
System	→	implementation construct
AlternativeChoice	→	redundant (same as CourseOffering)
RegistrationProcess	→	operation (activity of using the system)
Information	→	vague (need to specify more clearly)
BillingSystem	→	irrelevant (external system)
Professor	→	O.K.
PeriodOfTime	→	vague (related to processing constraints)
Schedule	→	redundant (same as CourseOffering)

KEEPING THE RIGHT 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?

ASU: KEEPING THE RIGHT ASSOCIATIONS

Student *Requests* ~~CourseCatalogue~~	→	class eliminated
~~CourseCatalogue~~ *Contains* CourseOffering	→	class eliminated
~~Term~~ *Needs* CourseOffering	→	class eliminated
~~CourseCatalogue~~ *Includes* Course(Information)	→	class eliminated
~~CourseCatalogue~~ *helps* Student	→	class eliminated
Student *Makes* ~~InformedDecision~~	→	class eliminated
Student *Selects* CourseOffering	→	operation
~~System~~ *Allows* Student	→	class eliminated
Student *Indicates* AlternativeChoice	→	operation
? *Fills* CourseOffering	→	operation
? *Cancels* CourseOffering	→	operation
CourseOffering *Has* Student	→	**O.K.**
Student *Completes* ~~RegistrationProcess~~	→	class eliminated
~~System~~ *Sends* ~~Information~~ *To* ~~BillingSystem~~	→	class eliminated
~~BillingSystem~~ *Bills* Student	→	class eliminated
Professor *Accesses* ~~System~~	→	class eliminated
Professor *Indicates* CourseOffering	→	operation
Professor *Teaches* CourseOffering	→	**O.K.**
Professor *Sees* Student	→	operation
Student *SignsUpFor* CourseOffering	→	operation
Term *Has* ~~PeriodOfTime~~	→	class eliminated
Student *Changes* Schedule	→	operation
Student *Accesses* ~~System~~	→	class eliminated
Student *Adds* CourseOffering	→	operation
Student *Drops* CourseOffering	→	operation

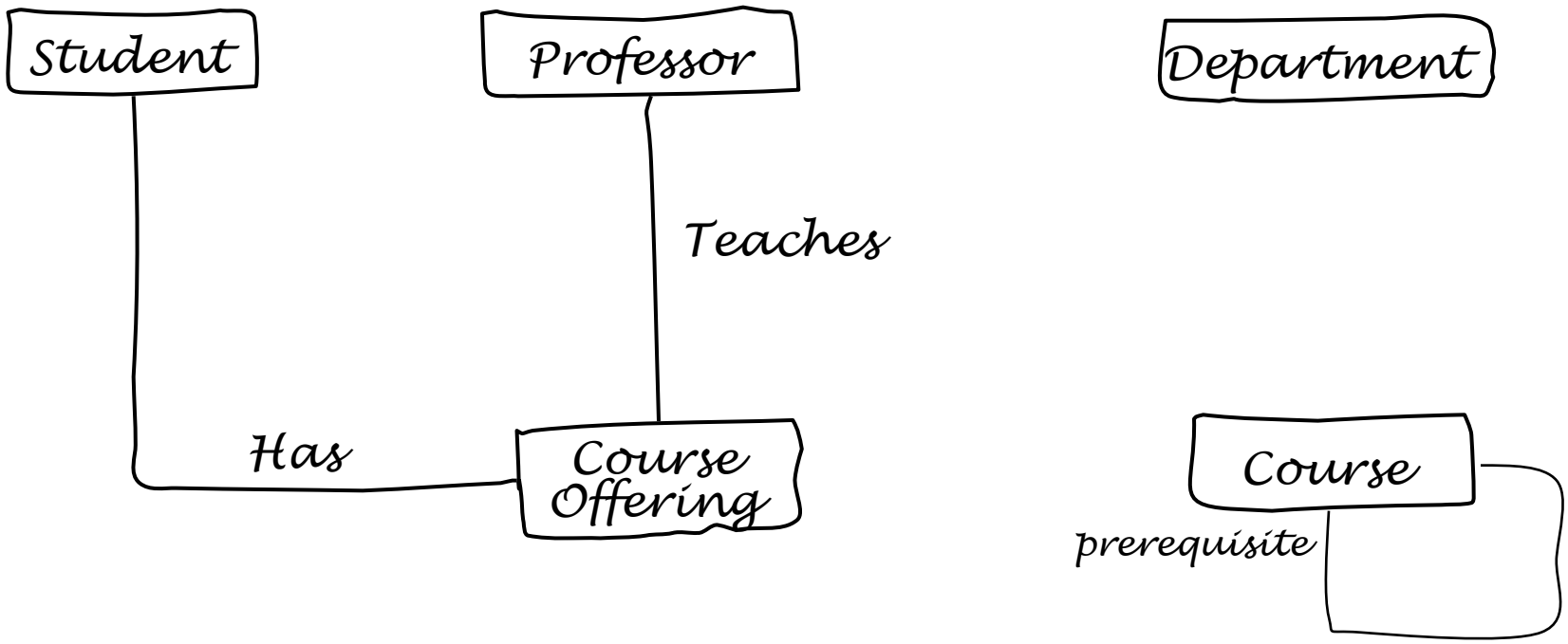
Initial class diagram

Classes

Student Department
Course Professor
CourseOffering

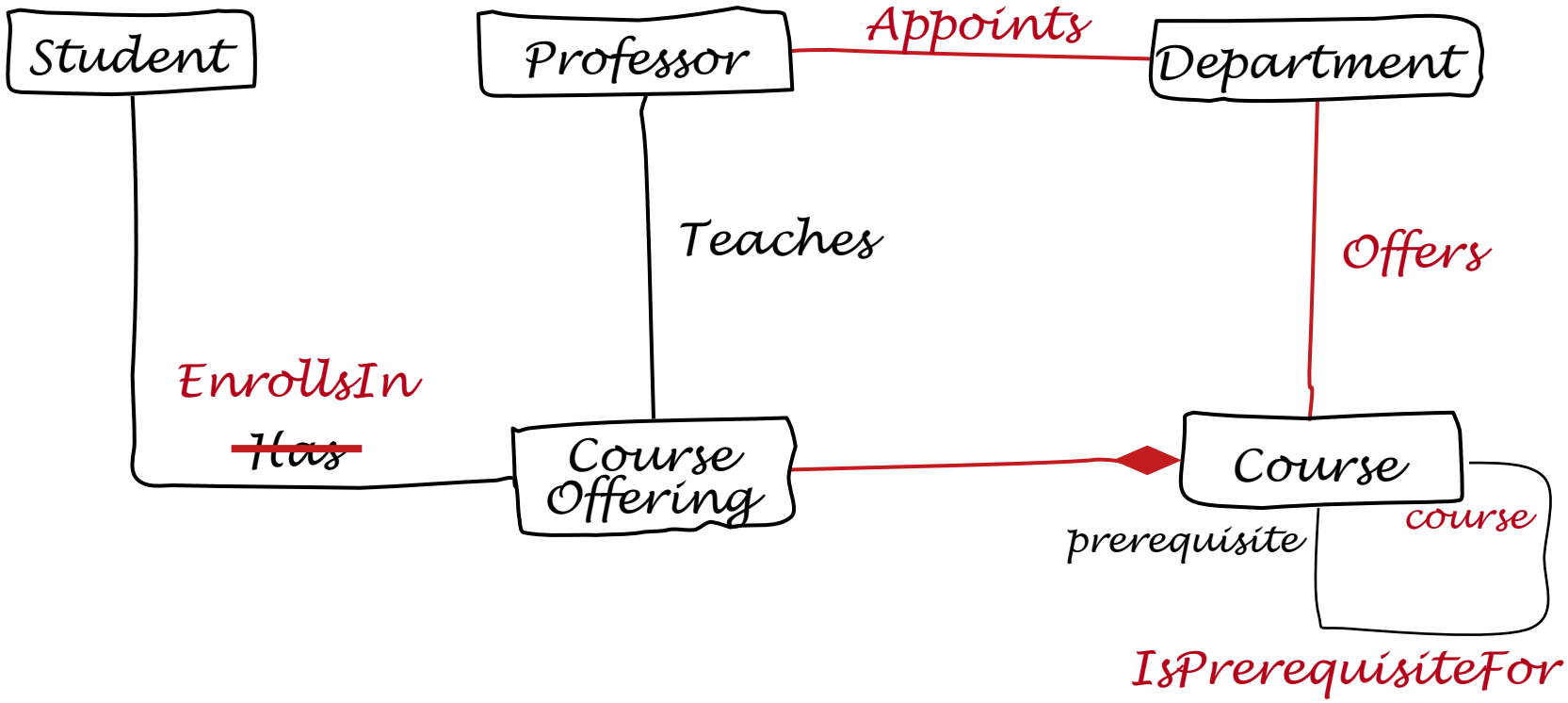
Associations

CourseOffering *Has* Student
Professor *Teaches* CourseOffering



First refinement

Association information in red is inferred from domain knowledge/experts.



IDENTIFYING ATTRIBUTES

- Attributes usually correspond to nouns 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 problem statement and will have to be obtained from domain experts or application documentation.

ASU: Attributes

Student
userId :String password : String userGroup : Integer surname : String otherNames : String address : String DOB : Date dateOfAdmission : Date levelOfStudy : String modeOfStudy : String yearOfStudy : Integer

Professor
userId : String password : String userGroup : Integer surname : String otherNames : String address : String DOB : Date qualification : String dateOfAppointment : String

Department
code: String title : String

Course
code : String title : String description: String credits : Integer

EnrollsIn
grade: Decimal

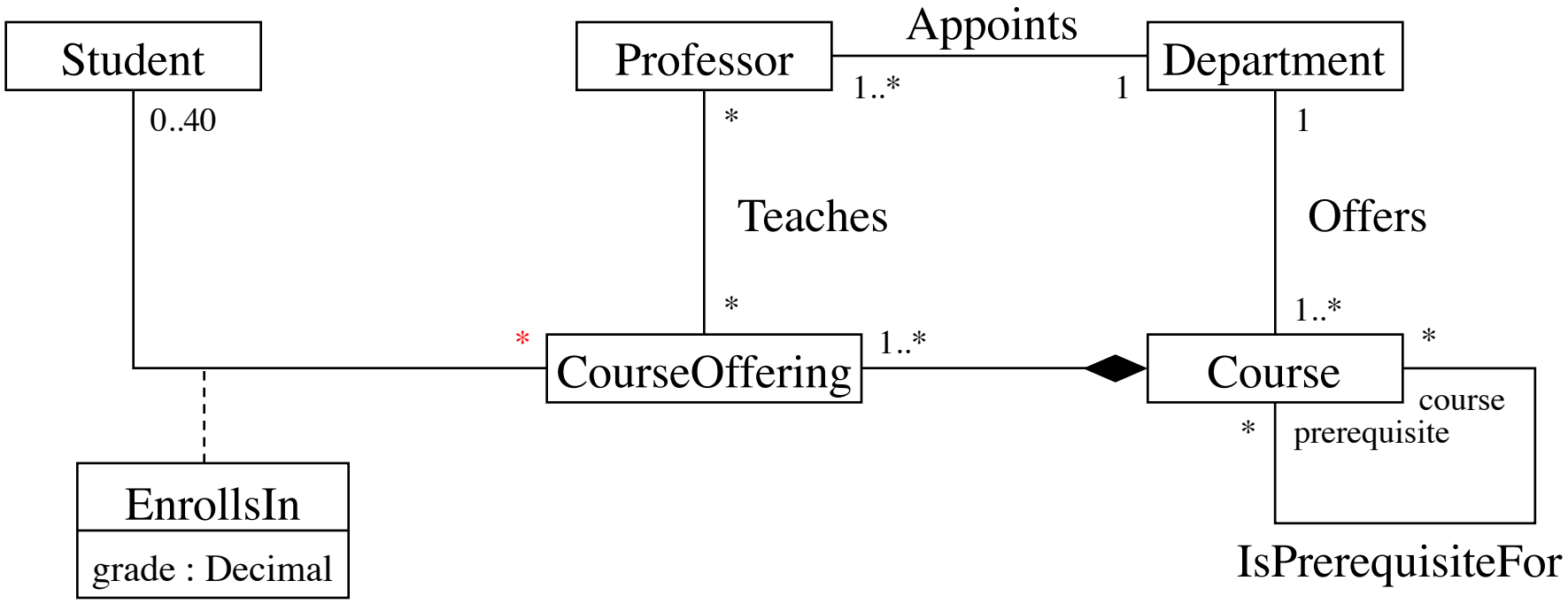
CourseOffering
offeringId : String term : String year : Integer

Most of these attributes are obtained from the domain experts.

Keeping the right 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?

ASU:Association multiplicity



GENERALIZATION

- *If it is meaningful to do so*, we classify classes according to the similarity of their attributes, operations and associations.

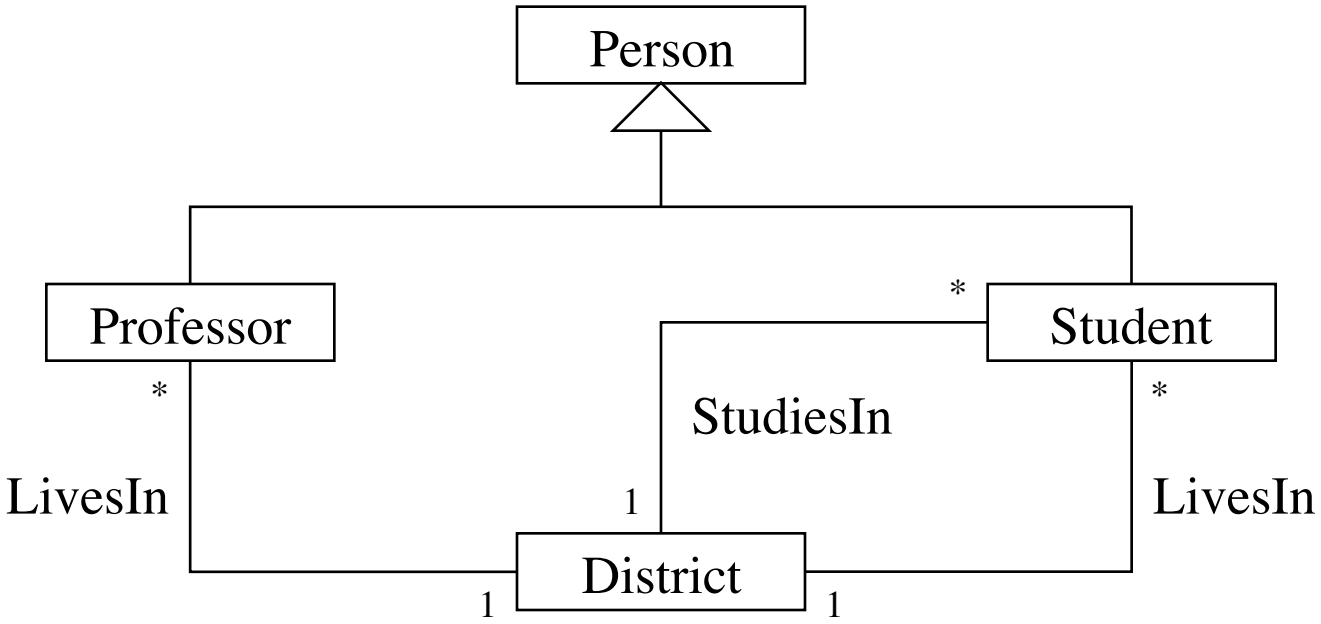
  ☞ **Look for "kind-of" statements that are true in the real world.**

- Do not nest too deeply:
  - 2-3 levels → **OK**
  - 4-6 levels → **maybe**
  - 10 levels → **too deep!**

Goal: Simplicity of representation and modeling clarity.

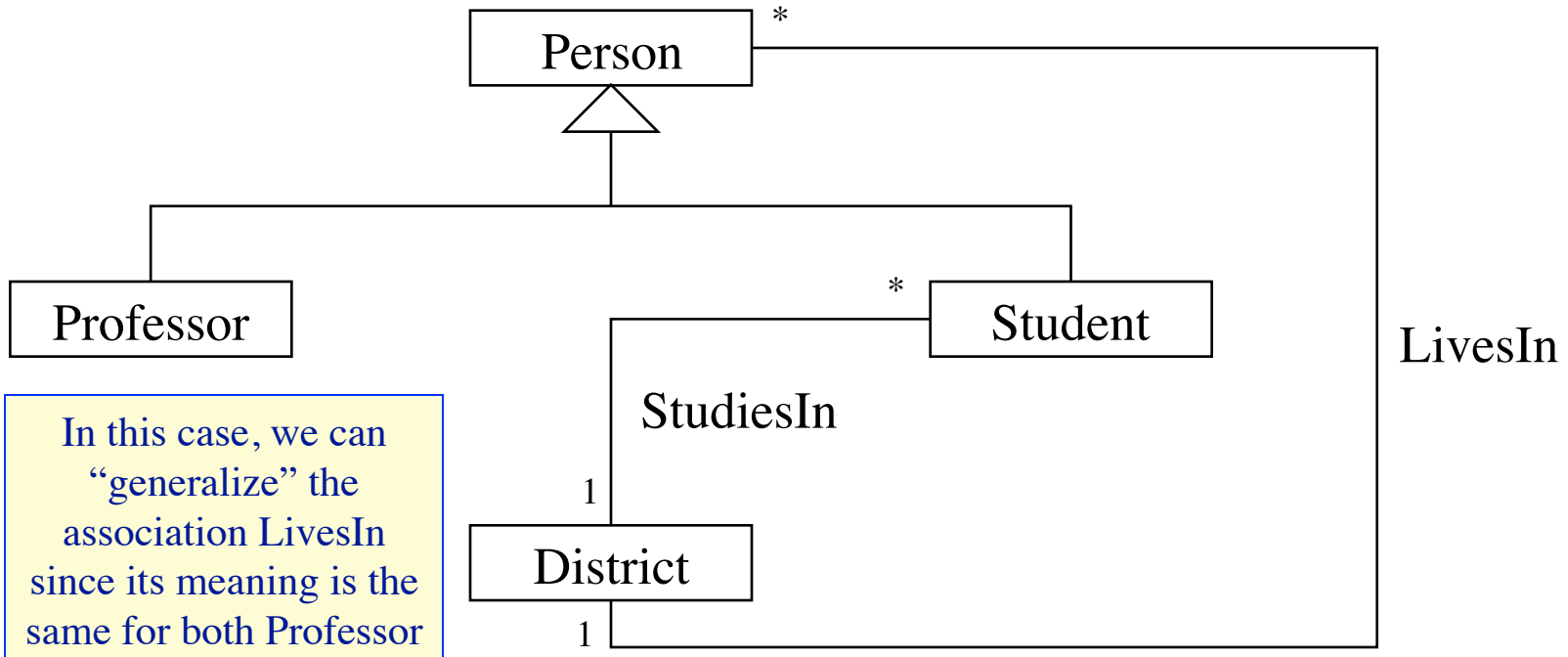
Generalization (cont'd)

☞ **We need to decide how to handle associations in which subclasses participate.**



Generalization (cont'd)

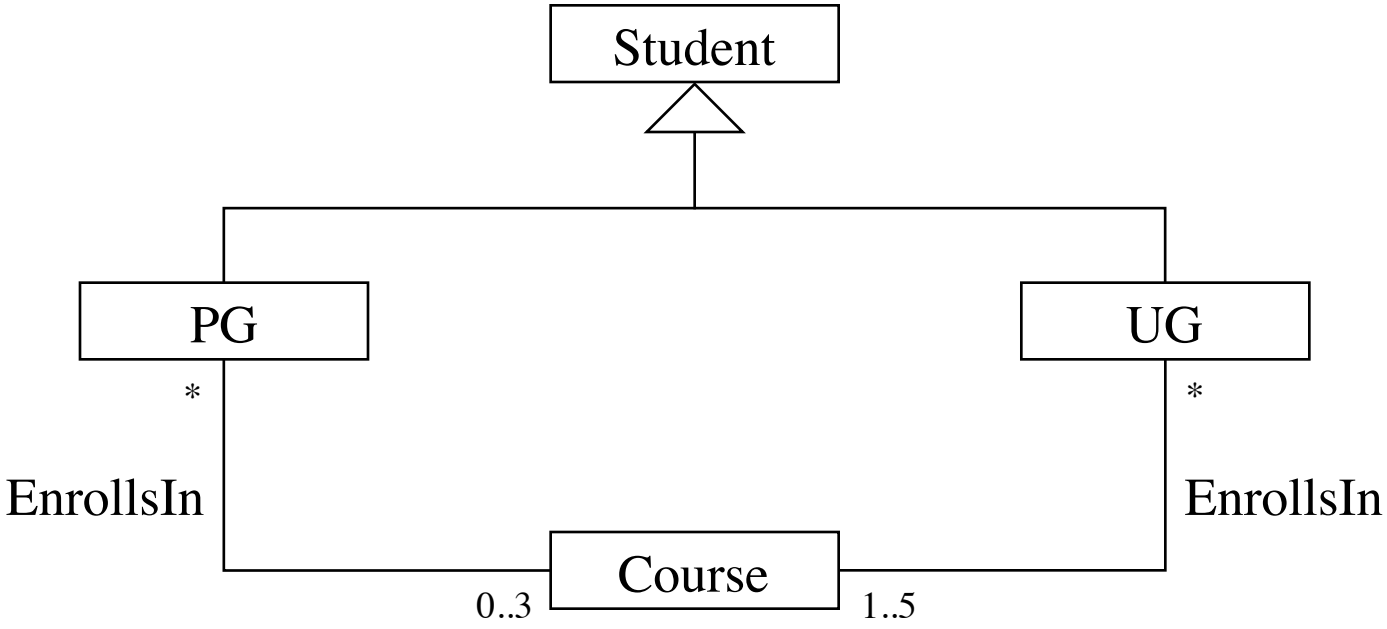
☞ **We need to decide how to handle associations in which subclasses participate.**



In this case, we can "generalize" the association LivesIn since its meaning is the same for both Professor and Student.

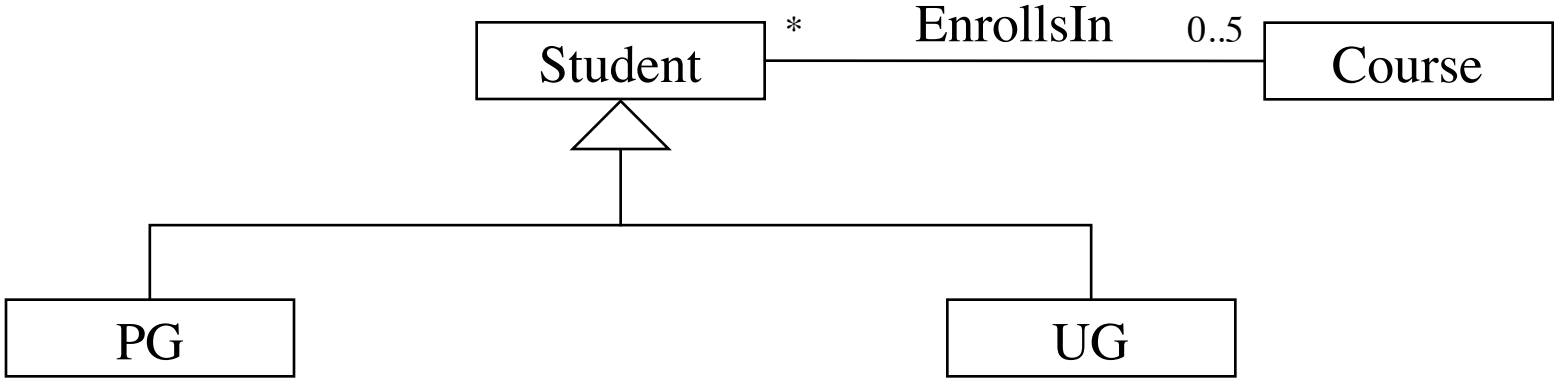
Generalization (cont'd)

☞ **Sometimes it is necessary to "adjust" the multiplicity of an association.**



Generalization (cont'd)

☞ **Sometimes it is necessary to "adjust" the multiplicity of an association.**



In this case, although we can "generalize" the EnrollsIn association, we need to use the *minimum* of the min-card and the *maximum* of max-card of both original EnrollsIn associations in order to preserve the semantics of the original class diagram.

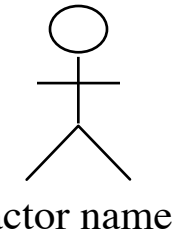
Use-case modeling

- Captures the system behavior from the *users' point of view*.
- Developed in cooperation with the domain model.
- Helps in:
  - capturing data and functional requirements.
  - planning iterations of development.
  - validating the system.
- The dynamic model gets started with use-case analysis.
  ☞ **USE CASES DRIVE THE ENTIRE DEVELOPMENT EFFORT.**

☞ **The use-case model is developed incrementally.**

All required functionality is described in the use cases.

Actors



An *actor* represents something outside the system that interacts *directly* with it.

Can be a person or another system.

Provides input to or receives output from the system.

A role a user can play → multiple roles per user; multiple users per role.

☞ Actors are usually the source for discovering use cases.

An actor is a stereotype of a UML class:

an actor is a classifier; a specific user or system is an instance.

ASU Course Registration System

System Requirements Capture

Use-case Model

Actors

ASU: ACTORS

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.

Professors 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.

ASU: ACTORS

Student

A person who is registered to take classes at the university.

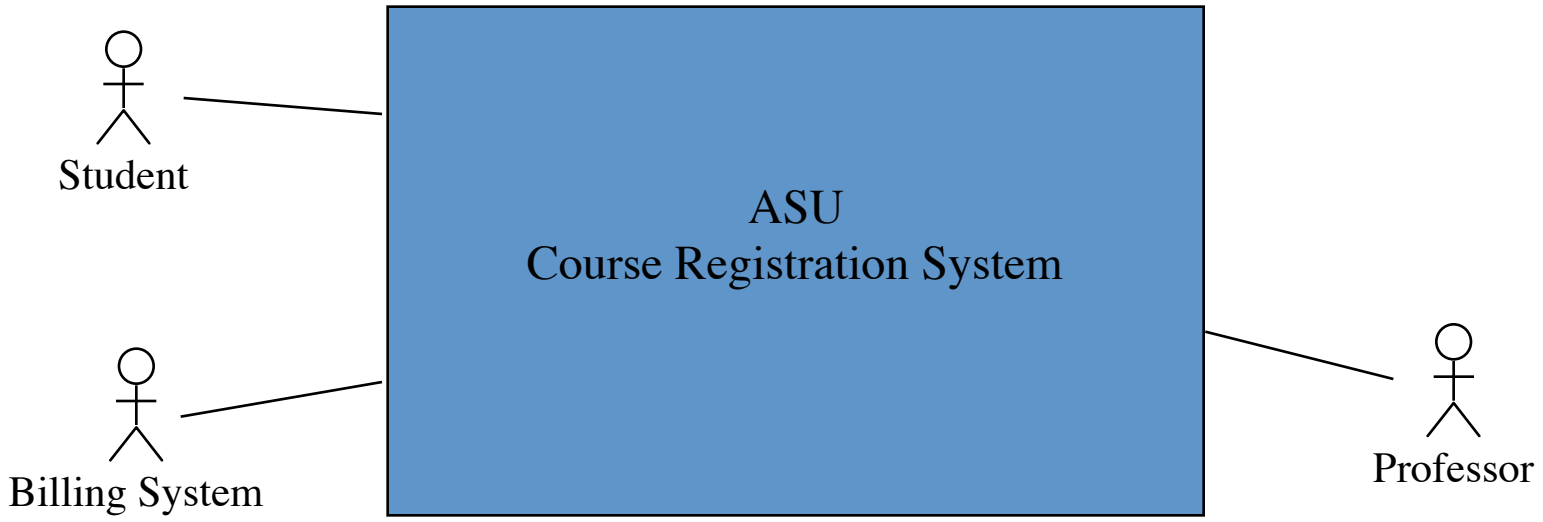
Billing System

An external system responsible for billing students.

Professor

A person who is part of the teaching staff of the university.

ASU: USE-CASE MODEL SO FAR



ASU Course Registration System

System Requirements Capture

Use-case Model

Use Cases

ASU: USE CASES

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

functionality: Someone – *prepare course catalogue*

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

functionality: Part of *prepare course catalogue* functionality.

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

functionality: Student – *select course offering*

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

functionality: Same as *select course offering* functionality.

ASU: USE CASES

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

functionality: None (This is mainly a domain model requirement.)

A course offering with fewer than ten students will be canceled

functionality: Someone – *cancel course offering*

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.

functionality: Billing System – *receive billing information*

Professors 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.

functionality: Professor – *select courses to teach*
Professor – *request enrollment list*

ASU: USE CASES

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

functionality: Student – *change course offering*

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

functionality: Student – *add course offering*
Student – *drop course offering*

ASU: ACTORS & THEIR FUNCTIONAL REQUIREMENTS

Registrar

Discovered from domain experts.

A person who is responsible for maintaining the curriculum information inclusive of students, professors and courses. The Registrar uses the Course Registration System to prepare the course catalogue for the coming term and to manage course offerings.

Student

A person who is registered to take classes at the university. A student uses the Course Registration System to register for courses in the current or a future term.

Billing System

An external system responsible for billing students. After a student successfully registers for a term, the Course Registration System sends the billing information to the billing system.

Professor

A person who is part of the teaching staff of the university. A professor uses the Course Registration System to make choices on the courses to teach and to request course enrolment lists.

ASU: USE CASES — REQUIRED FUNCTIONALITY

Registrar – *prepare course catalogue*

Registrar – *cancel course offering*

Student – *select course offering*

Student – *change course offering*

Student – *add course offering*

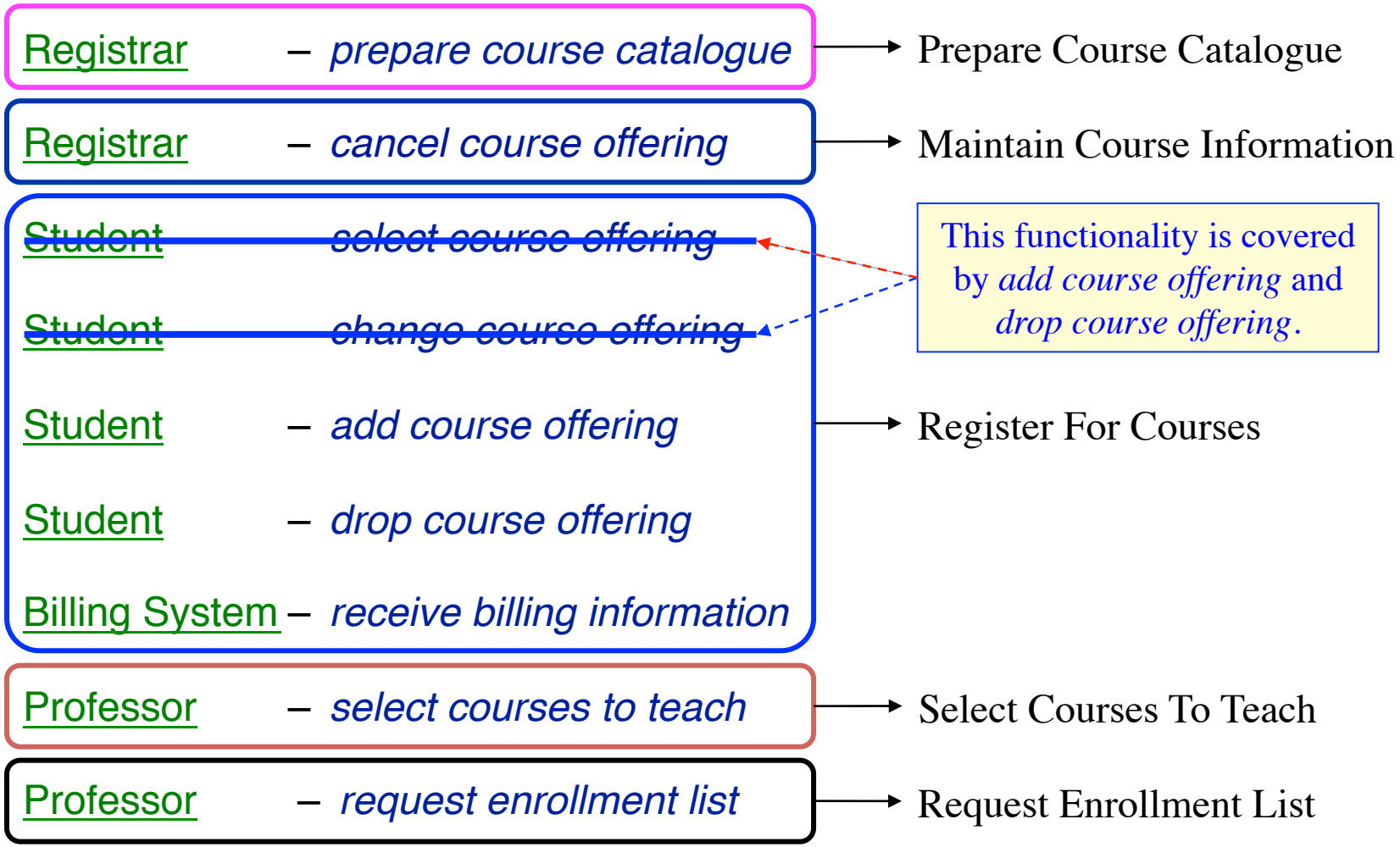
Student – *drop course offering*

Billing System – *receive billing information*

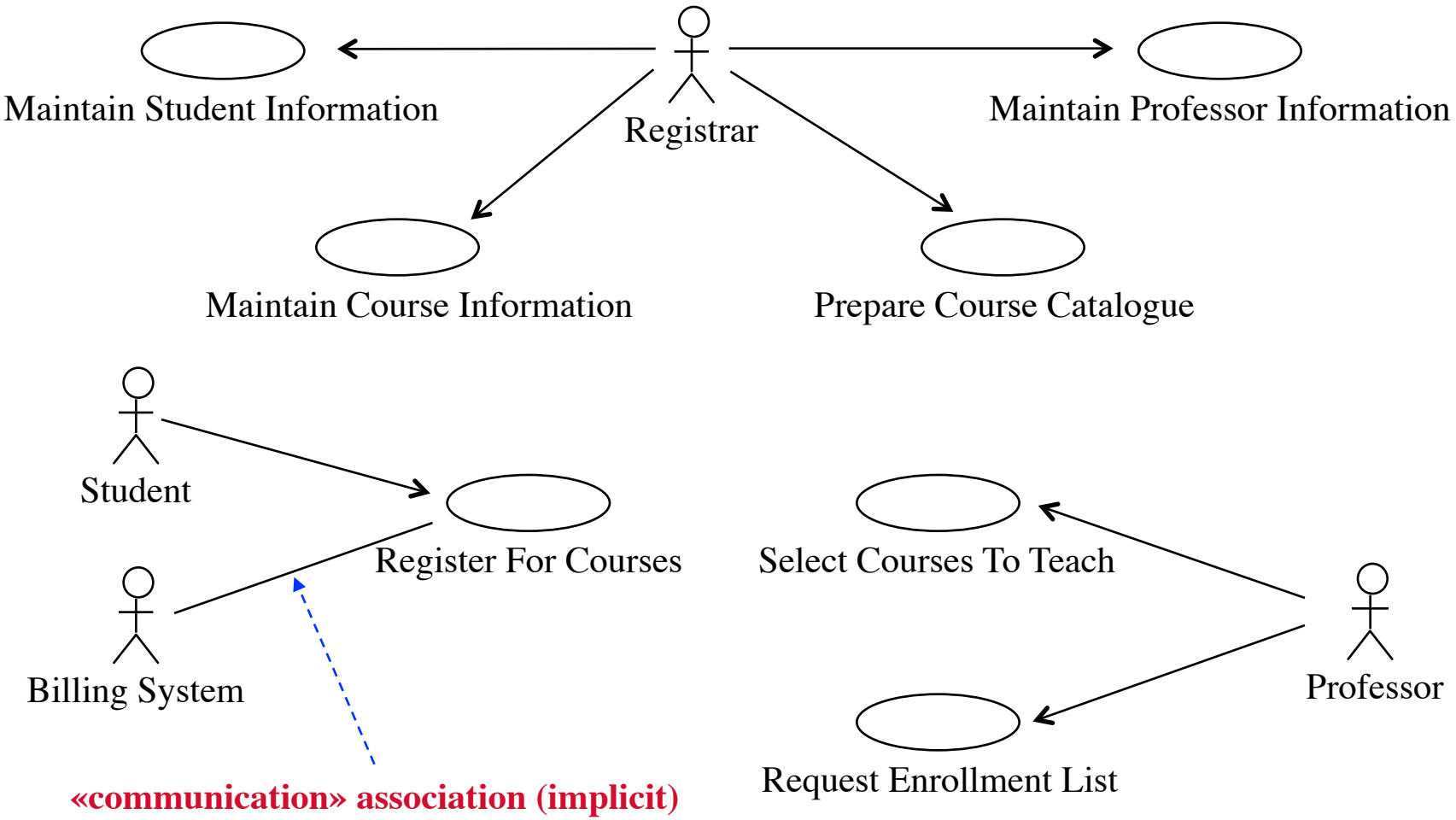
Professor – *select courses to teach*

Professor – *request enrollment list*

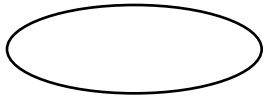
ASU: USE CASES — FUNCTIONALITY ANALYSIS AND GROUPING



ASU: USE-CASE MODEL



Use case



use-case name

A *use case* is a specific way of using the system by performing some part of its functionality.

Describes the interaction that takes place between an actor and the system and what the system must do.

☞ Considered from the actor's viewpoint.

Constitutes a *complete sequence* of events/actions.

Always initiated by an actor.

- Initially, only consider normal sequence of events/actions.

☞ **Ignore alternative events/actions.**

ASU: INITIAL USE-CASE SPECIFICATION (For reference)

Register For Courses

This use case describes the process by which a student registers for a course offering. It provides the capability to create and modify a student's study schedule for the coming term.

Flow of Events

1. Add course offering.
2. Drop course offering.
3. Send billing information.

Initially, we just give a brief description of the purpose of a use case and an outline of the steps required to perform the use case.

The steps are refined and described in more detail as we discover more about the functionality required.

USE-CASE DETAILED SPECIFICATION (For reference)

- **use case name** (present-tense, verb phrase in active voice)
- **brief description**
- **participating actors** (can show using a use-case diagram fragment)
- **preconditions** (if any)
- **flow of events** (the required *normal* sequence of actions)
- **postconditions** (if any)
- **alternative flows** (if any), which describe:
  - optional actions performed *in addition to* the normal actions
  - variant actions performed *to replace* some normal actions
  - exceptional actions performed *to handle abnormal situations* (e.g., invalid input, system errors, etc.)
- **special (nonfunctional) requirements** (if any)

Identifying actors(For reference)

- Who or what uses the system?
- What roles do they play in the interaction?
- Who gets and provides information to the system?
- What other systems interact with this system?
- Does anything happen at a fixed time?
- Who installs, starts and shuts down, or maintains the system?

☞ **It is possible to have both a domain model class and an actor that represent the same thing.**

☞ **Time can be an actor. (BUT be very careful!)**

☞ **Input/output devices are never actors!**

For each actor, briefly describe the role it plays when interacting with the system.

WHAT IS A GOOD USE CASE? (For reference)

A use case typically represents a major piece of functionality that is complete from beginning to end.

A use case must deliver something of value to an actor.

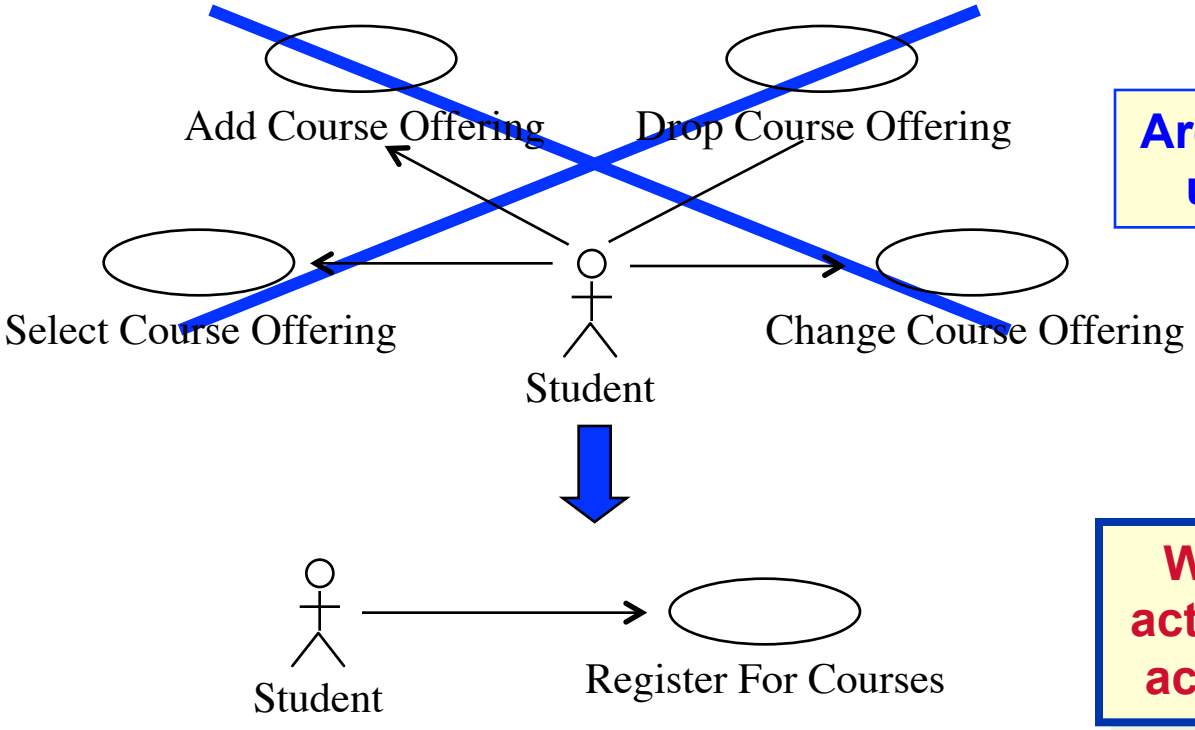
Generally, it is better to have longer and more extensive use cases than smaller ones.

We want real and complete use cases, not several sub-cases of a use case.

WHAT IS A GOOD USE CASE? (For reference)

A use case typically represents a major piece of functionality that is complete from beginning to end.

A use case must deliver something of value to an actor.



Are these good use cases?

What is the actor trying to accomplish?

Scenario (For reference)

A *scenario* is a concrete, focused, informal description of a *single use* of the system from the viewpoint of a *single actor*.

☞ **It is an actual attempt to carry out a use case.**

Note: There are two viewpoints of use-case modeling:

1. top-down: start with use cases, refine with scenarios.
2. bottom-up: start with scenarios, abstract use cases.

☞ **In reality, the use-case specifier uses both viewpoints.**

A use case is a stereotype of a UML class:
use case is a classifier; scenario is an instance

Identifying use cases and scenarios (For reference)

- What are the tasks that an actor wants the system to perform?
- What information does an actor access (create, store, change, remove or read) in the system?
- Which external changes does an actor need to inform the system about?
- Which events does an actor need to be informed about by the system?
- How will the system be supported and maintained?

☞ **State a use case name from the perspective of the user as a present-tense, verb phrase in active voice.**

☞ **Provide a description of the purpose of the use case and an outline of its functionality.**

☞ **Use application domain terms in descriptions (i.e., from the glossary/data dictionary).**

Nonfunctional requirements

A *nonfunctional requirement* places constraints on a use case or on the system.

It is identified by asking questions about the system's:

Design Qualities –reliability, supportability, maintainability, etc.

Performance – speed, throughput, response time, accuracy, etc.

Interface – user interface (learnability, usability); external system interface (formats, timing).

Hardware – implementation platform, memory size, storage capacity.

Implementation – standards, languages, error handling.

Security – system access; data access; physical access.

Physical environment – abnormal conditions; distributed operation

Documentation – what is required and for who?

Management – system back up, installation, maintenance.

Specifying nonfunctionl requirements

Nonfunctional requirements are specified as supplementary requirements on use cases or on the system as a whole.

- Some nonfunctional requirements will be implemented as administration use cases.
  - Login
  - System start up
  - System shut down
  - System backup

***Administration use cases* are those that deal with non-functional requirements such as security or system operation and maintenance.**

Calendar: Possible classes

The user can schedule an event with the starting time, end time, the event description, the event location, the frequency, if a reminder is needed, how much time ahead the reminder should be triggered, and additional description for the event.

Nouns: User, Event, Starting time, end time, description, location, frequency, reminder, additoinal description

Attributes: Starting time, end time, description, location, frequency, reminder, additoinal description

Calendar: Possible classes

The time specification should be in intervals of 15 minutes. The event frequency can be one-time, daily, weekly, or monthly. Should a reminder be needed, it should be triggered and displayed to the user at or less than the specified time interval before the scheduled time of the event. The reminder should only be given ONCE.

This requirements include all constraints: what conditions should be satisfied.

Calendar: Possible classes

The reminder information, the event location, and the event description can be optionally specified. Other information must be specified. The location must be selected from a list of predefined locations in the system.

The successfully scheduled event should cause GUI to change accordingly. An entry should show in the daily view and the day when the event is scheduled should change color on the month view.

Nouns: Let us not consider GUI elements for now

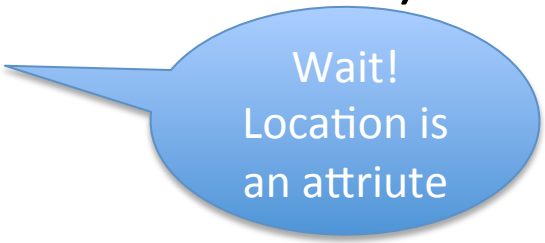
Calendar: Possible classes

The location must be uniquely identified by its name. The user must select from a set of predefined locations. The locations can be added through a separate interface

Nouns: name

List all the nouns

- User (OK)
- Event, (OK)
- Starting time, (Attribute of event)
- End time, (Attribute of event)
- Description, (Attribute of event)
- Location, ~~(Attribute of event)~~ (OK)
- Frequency, (Attribute of event)
- Reminder, (Attribute of event)
- Additoinal description , (Attribute of event)
- Name, (Attribute of location)



Calendar: Possible associations

The user can schedule an event with the starting time, end time, the event description, the event location, the frequency, if a reminder is needed, how much time ahead the reminder should be triggered, and additional description for the event.

Associations: User schedules event
Event has reminder
? triggers reminder（Time?)

Calendar: Possible associations

The time specification should be in intervals of 15 minutes. The event frequency can be one-time, daily, weekly, or monthly. Should a reminder be needed, it should be triggered and displayed to the user at or less than the specified time interval before the scheduled time of the event. The reminder should only be given ONCE.

Associations: Calendar displays reminder
Event needs reminder
Calendar gives reminder

Calendar: Possible associations

The reminder information, the event location, and the event description can be optionally specified. Other information must be specified. The location must be selected from a list of predefined locations in the system.

Associations: User specifies remind, location, description
User select location.

The successfully scheduled event should cause GUI to change accordingly. An entry should show in the daily view and the day when the event is scheduled should change color on the month view.

Associations: Event change GUI
Entry show in daily view
Event change color

Calendar: Possible associations

The location must be uniquely identified by its name. The user must select from a set of predefined locations. The locations can be added through a separate interface

Associations: Name identifies location
User select location
User add location

List all the associations

- User schedules event (Operation)
- Event has reminder (Attribute)
- Time triggers reminder (OK)
- Calendar displays reminder (OK)
- Event needs reminder (Attribute)
- Calendar gives reminder (Redundant)
- User specifies remind, location, description (Operation)
- User select location. (Operation)
- Event change GUI. (Constraint/Condition)
- Entry show in daily view. (Constraint/Condition)
- Event change color. (Constraint/Condition)
- Name identifies location. (Constraint/Primary key condition)
- User select location (Operation)
- User add location (Operation)

Conclusions

- We found a new class Location
- Domain model classes do not map to implementation in a 1-to-1 fashion
- More classes added later