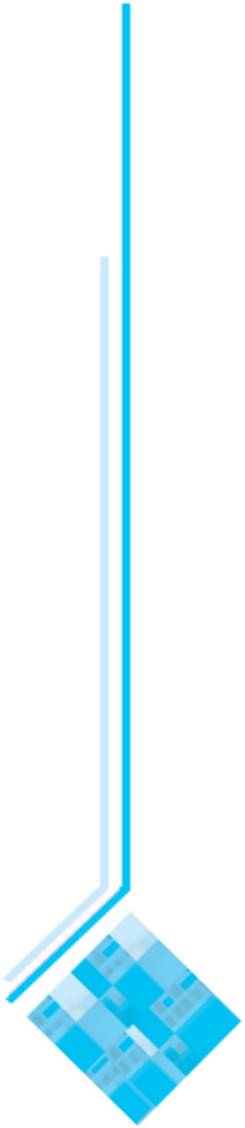


Teaching Introductory Software Engineering

ASEE&T 2011

Timothy C. Lethbridge
University of Ottawa
tcl@site.uottawa.ca
http://www.site.uottawa.ca/~tcl

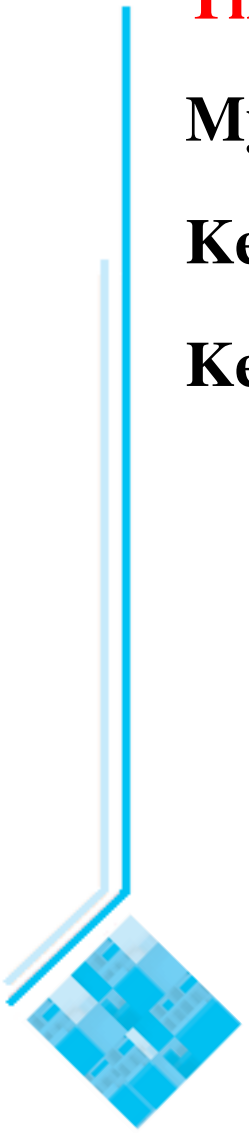
Agenda

The course I teach

My experiences and how they shaped my teaching

Key lessons: Keeping attention and fostering affinity for SE

Keeping teaching focused: Areas I suggest to emphasize



The Course I Teach: SEG2105 Introduction to SE (link)

A 40-hour course taught in year two of four at the University of Ottawa

- Students' background is two Java courses

Students are in a mix of different degree programs

- Software Engineering
- Computer Science
- Computer Engineering
- Arts, social science, other engineering: in a minor

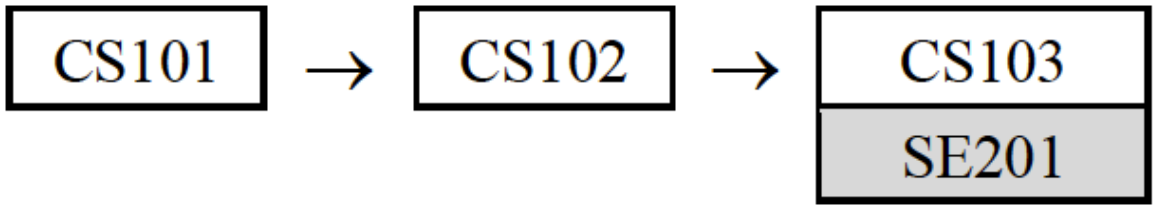
Registration: 70 students per course section

My course is very similar to Course SE201 of SE2004

http://sites.computer.org/ccse/SE2004Volume.pdf **p. 100**

Introduction to Software Engineering

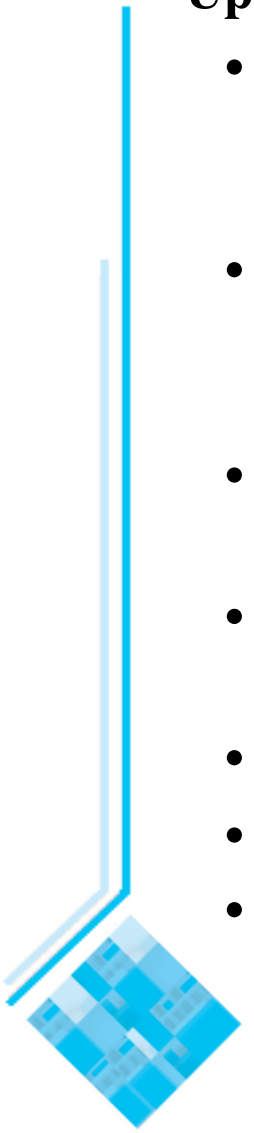
Principles of software engineering: Requirements, design and testing. Review of principles of object orientation. Object oriented analysis using UML. Frameworks and APIs. Introduction to the client-server architecture. Analysis, design and programming of simple servers and clients. Introduction to user interface technology.



SE201 Learning objectives

Upon completion of this course, students will have the ability to:

- Develop clear, concise, and sufficiently formal requirements for extensions to an existing system, based on the true needs of users and other stakeholders
- Apply design principles and patterns while designing and implementing simple distributed systems-based on reusable technology
- Create UML class diagrams which model aspects of the domain and the software architecture
- Create UML sequence diagrams and state machines that correctly model system behavior
- Implement a simple graphical user interfaces for a system
- Apply simple measurement techniques to software
- Demonstrate an appreciation for the breadth of software engineering



Agenda

The course I teach

My experiences and how they shaped my teaching

Key lessons: Keeping attention and fostering affinity for SE

Keeping teaching focused: Areas I suggest to emphasize

My Experiences 1

I have taught Introduction to Software Engineering since 1991

- Textbooks in the early years
  - —Pressman, Sommerville, Pfleeger

The 'rote knowledge' in the big textbooks went over students' heads

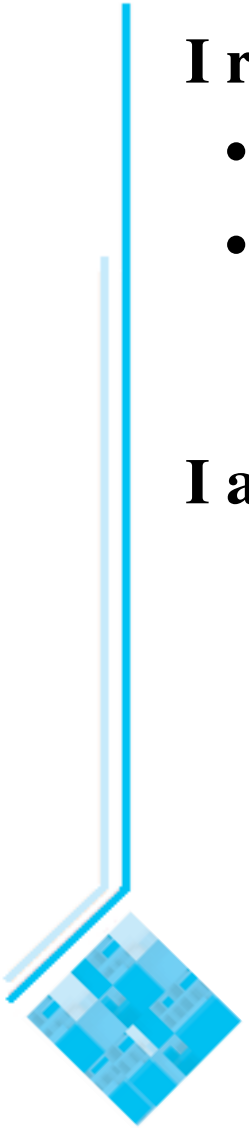
- E.g, teaching modeling using a few examples taught them very little

My Experiences 2

I recorded

- Which teaching approaches seemed to work
- Bad answers and misconceptions I encountered on exams

I adapted what I learned into the present materials



I Wrote My Own Book in 2001 That Incorporates My Experiences

Lethbridge and Laganiere, "Object-Oriented Software Engineering: Practical Software Development Using UML and Java", 2nd Edition, McGraw Hill, 2004

http://www.lloseng.com

Agenda

The course I teach

My experiences and how they shaped my teaching

Key lessons: Keeping attention and fostering affinity for SE

1. Build on what students know
2. Outcomes to avoid
3. Getting and keeping students' attention
   - Shock and awe
   - Mixed mode teaching with 'live' tools &problem solving
4. Integrating knowledge through experience
5. Ensuring students feel an affinity for SE

Keeping teaching focused: Areas I suggest to emphasize

A Key to Good Teaching: Understand What Students Already Know and Build On It

Students starting my course are moderately competent programmers (may not be true if SE started in year 1)

- They know programs have bugs
  - —This frustrates them
- They will be motivated to make better programs faster

Students have all used bad software

- Slow, unusable, crashes etc.
  - —Motivate students to avoid this

Students know very little about process, testing, modeling ...

Outcomes to Avoid

Students learn vocabulary only

- Exam questions that simply ask them to define terms

Students learn what, but not why and how

- E.g. syntax of UML but without an ability to apply it practically

Students think SE is boring and look forward to getting back to 'real programming'

Students learn techniques they will never apply

Key Teaching Methods That Work 1: Getting and Keeping Students' Attention

"Shock and Awe":

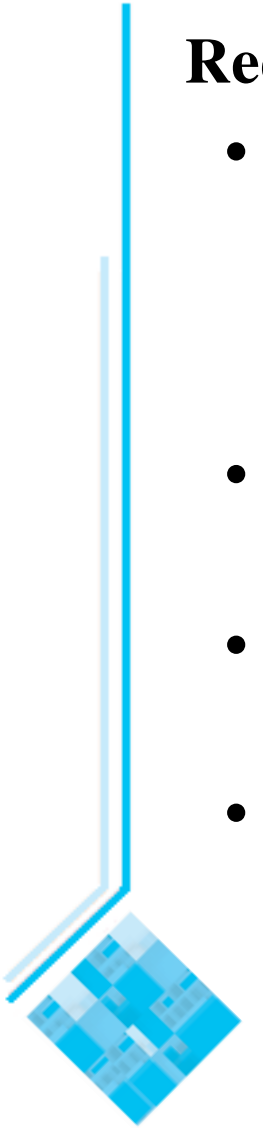
- Disasters caused by software
  - —Therac 25, London Ambulance, Ariane 5
- Recent items in the news, often related to security
  - —Playstation Network hacker attaches
- Massive wastes of money caused by doing things badly
  - —E.g. Air Traffic control
  - —Useful URLs:
    - Lessons From History
    - Project failures cost Billions
    - Risks Forum Digest

Key Teaching Methods That Work 1: Getting and Keeping Students' Attention – cont.

Mixed mode presentations

- Powerpoint presented with energy
  - —It's only "evil" if the presenter is boring
- Blackboard, whiteboard
  - —For design, modeling, testing
  - —Students help guide what appears
- Live use of modeling tools, showing generated code

Mixed-Mode Teaching Demo Using Board and UmpleOnline: http://try.umple.org



Requirements to model:

- A theatre has a series of productions; each production has a set of performances, and tickets are sold for performances. Performances also have a set of production staff and actors.
- A seat in the theatre is identified by row and seat number.
- A subscriber can purchase tickets to a set of performances.
- The theatre records the name, address, phone number and email address of all people.

Key Teaching Methods That Work 2: Integrating Knowledge Through Experience

Practical labs

- Measuring performance
- Modifying and existing system in small increments
- Generating code from a model

A project that includes all steps including requirements, design, testing and coding

- Coding is a level of design and is integral to SE

Key Teaching Methods That Work 3: Ensuring Students Feel an Affinity for SE

Relate topics to students' own experience

- Bad software they have used
- Their difficulty programming

Pride in being:

- an engineer and/or
- a computer professional

Point out interesting challenges

- This is not a dry and boring topic

Anecdotes from personal experience

- Stories they can relate to and empathize with

Agenda

The course I teach

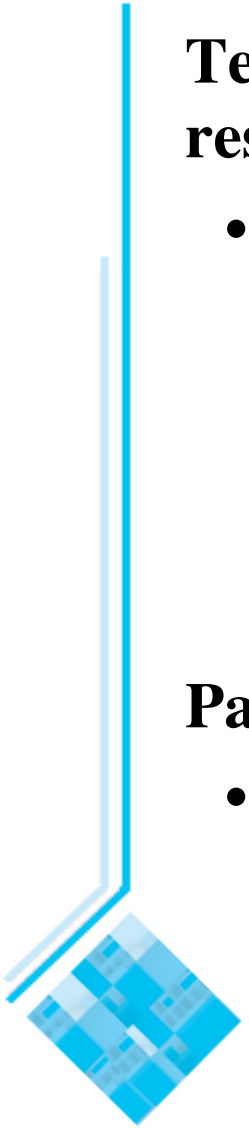
My experiences and how they shaped my teaching

Key lessons: Keeping attention and fostering affinity for SE

Keeping teaching focused: Areas I suggest to emphasize

1. Professionalism
2. Modeling class and state diagrams
3. Design principles
4. Design patterns
5. Agility
6. Reusability

Don't Try to Teach All Aspects of Each Area



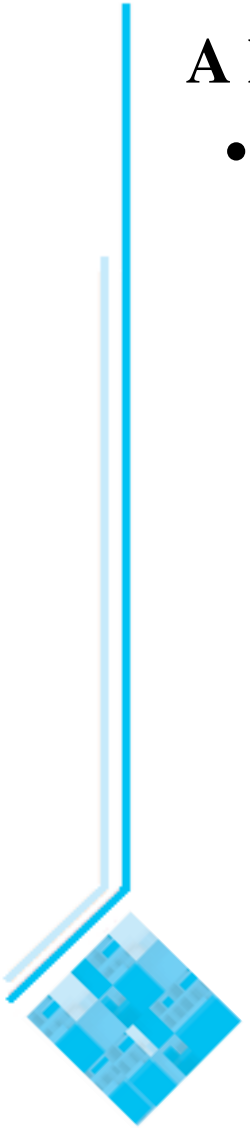
Teach a central subset in depth, with awareness of the rest

- E.g.
  - —Only the most useful patterns
  - —Key design principles
    - Divide and conquer, low coupling, high cohesion
  - —Subset of UML syntax and semantics

Pareto principle: 80-20 rule

- You don't have to teach them everything,
  - —Just the 20% that covers 80% of the ground
  - —Apply this recursively to subtopics

Areas of Focus 1: Professionalism



A key factor that distinguishes good engineering

- Key takeaway knowledge:
  - —It's a legally recognized profession in many jurisdictions
  - —You must take *responsibility* for safe, secure operation of the system
  - —Examples of things that are not acceptable:
    - Bugs
    - Poor usability
    - Undocumented, unmaintainable code
  - —Understanding clients correctly is difficult but key
  - —Efficient use of resources is key to engineering

Areas of Focus 2: Modeling Class and State Diagrams

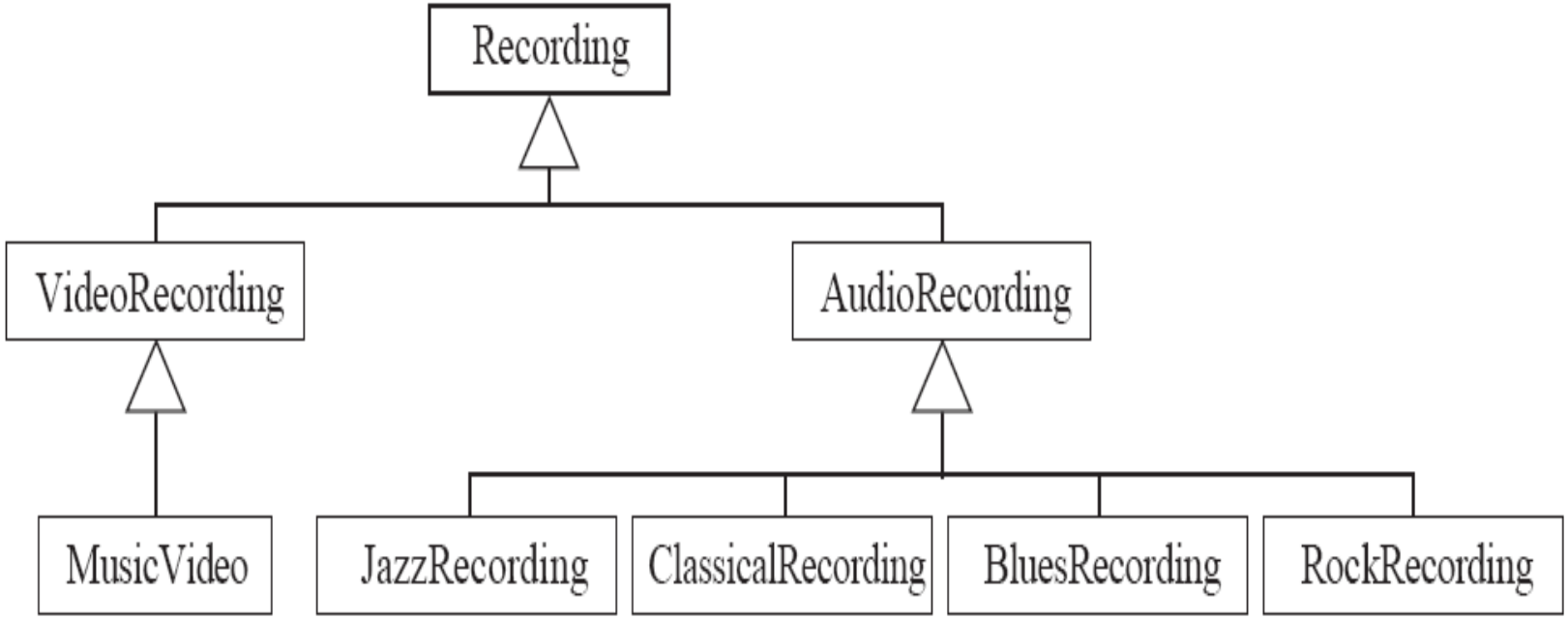
Most current practitioners are poor modelers

- Lack of understanding of semantics and pragmatics
- They just draw "pretty diagrams" with semantic errors

How to teach properly?

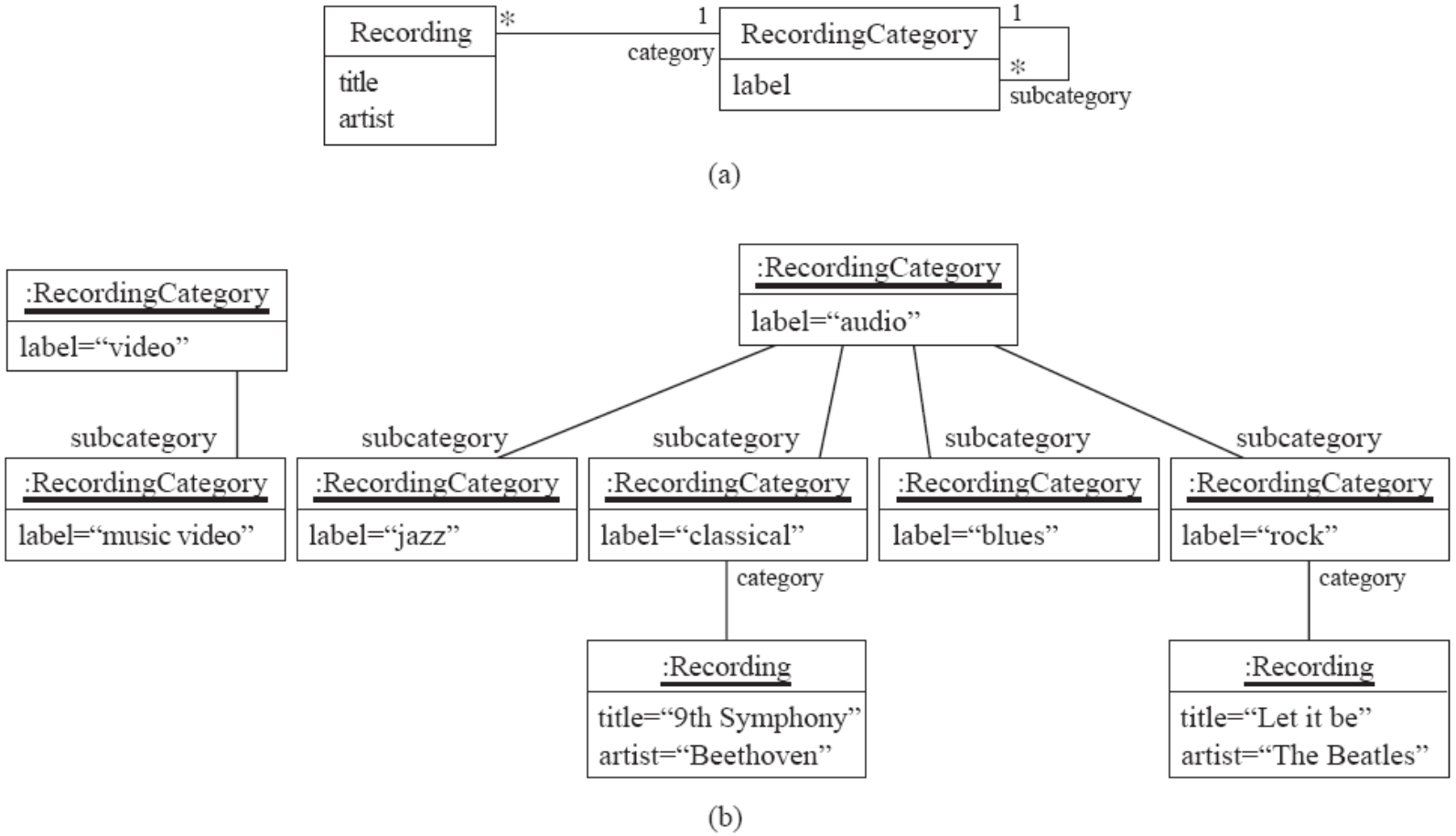
- Point out typical mistakes (antipatterns)
- Board work, where students point out solutions

Example: Avoiding unnecessary generalizations



Inappropriate hierarchy of classes, which should be instances

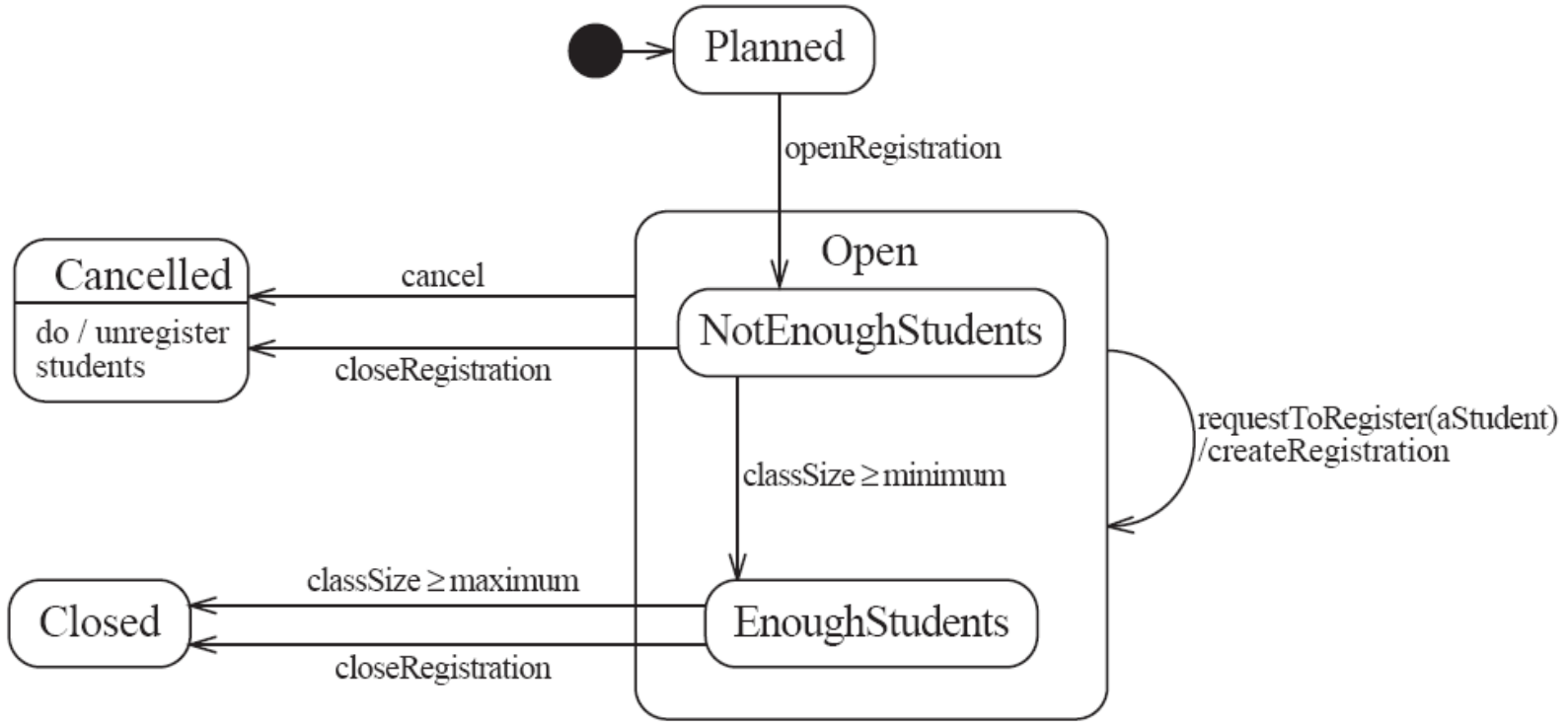
Avoiding unnecessary generalizations (cont)



Improved class diagram, with its corresponding instance diagram

Example: Class conversation about adding details to state diagrams

How to add 'drop course'



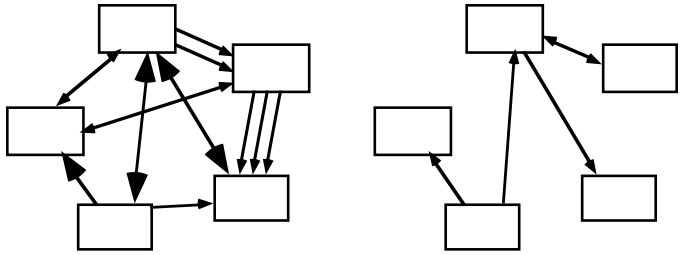
Areas of Focus 3: Design Principles

How to cohesion as 'organizedness'

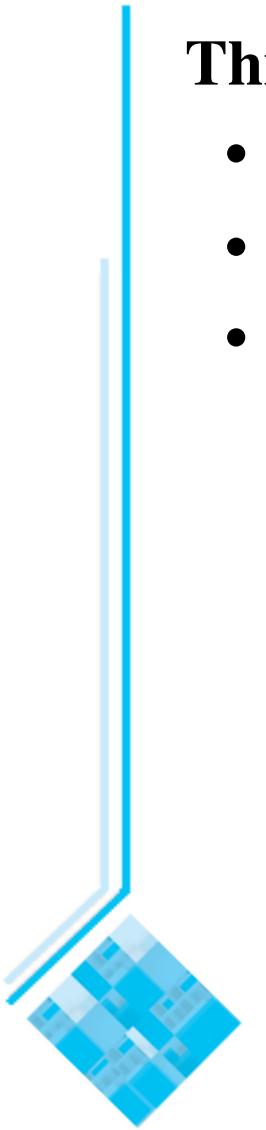
- Analogy: Organizing your house
  - —Temporal cohesion: A room for everything used in the morning; another room for evening things
  - —Functional cohesion: All the equipment and ingredients needed for a recipe kept together, and everything else kept out

More on design principles

Coupling as interdependencies



Areas of Focus 4: Design Patterns

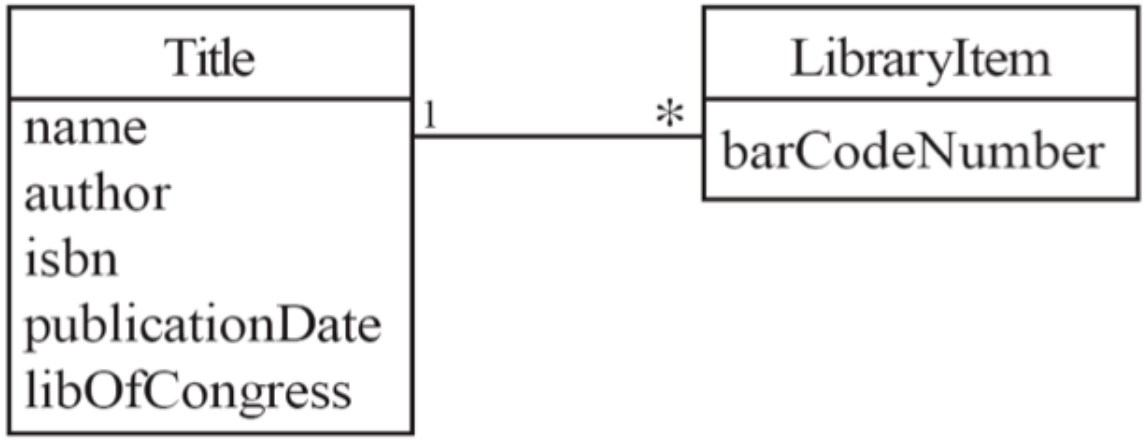


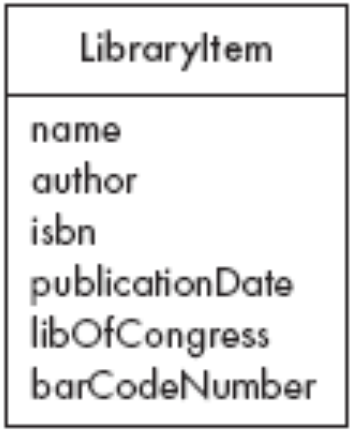
Three types of Patterns:

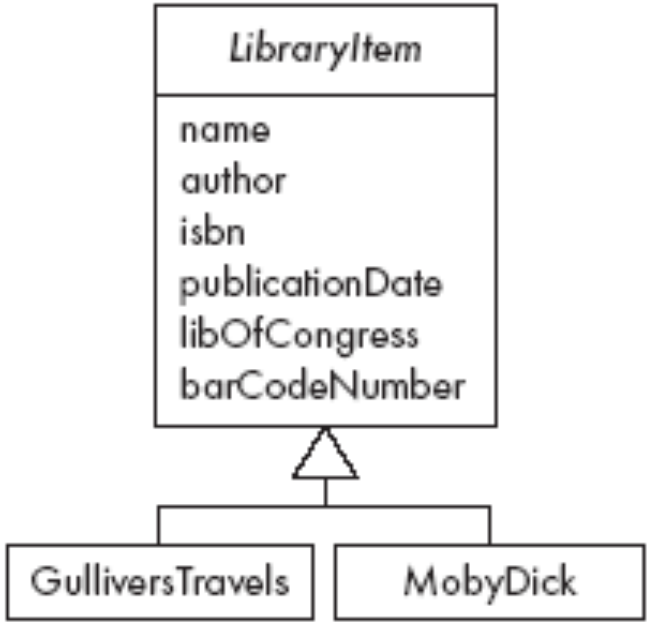
- Analysis Patterns
- Gang of Four
- Architectural

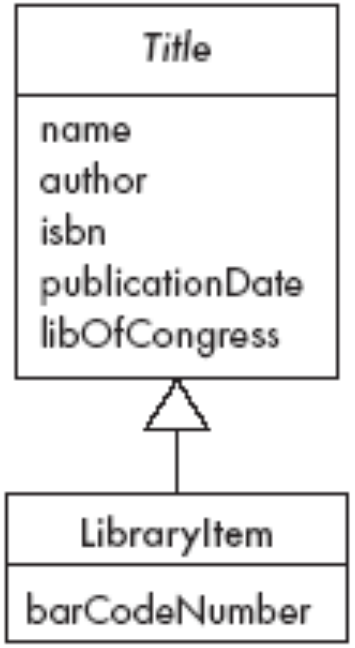
Examples of Teaching Patterns and Antipatterns

Abstraction-occurrence pattern



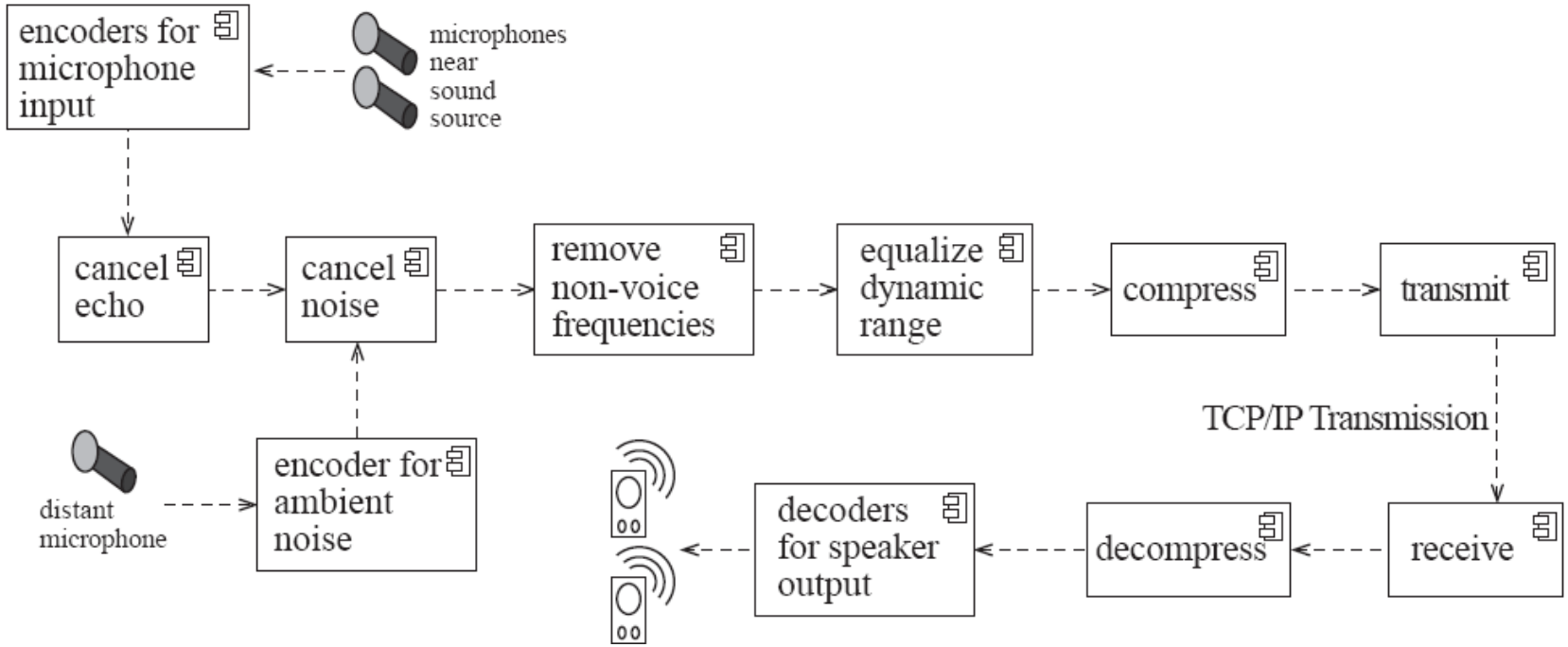






Example of Architectural Patterns

Show an entire system designed using "pipe and filter"



Architectural Patterns vs. Design Principles

	1	2	3	4	5	6	7	8	9	10	11
Multi-layers	■	■	■	■	■	■	■	■	■	■	■
Client-server	■	■	■	■		■	■		■	■	
Broker	■				■	■	■		■		■
Transaction processing	■	■	■				■				■
Pipe-and-filter	■	■	■	■	■	■	■			■	■
MVC	■	■	■			■	■			■	
Service-oriented	■	■	■	■	■	■		■	■	■	■
Message-oriented	■	■	■	■	■	■	■			■	■

Area of Focus 5: Agility

Other methods should be downplayed because they fail too often

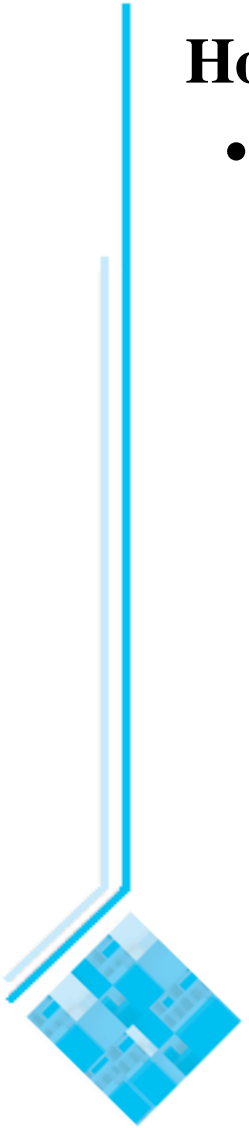
Key concepts emphasized:

- Test driven development
- Small increments to requirements delivered quickly
- End-user involvement

How to teach?

- Story about the origin of "Waterfall"
- Failures of waterfall
- Small increments in the labs, with test cases

Area of Focus 6: Reusability



How to teach:

- Give them OCSF framework and have them build new systems using it

Topics With Focus Reduced to 2-3 Hours

Why downplay them?

- Students can't relate to extensive detail
- Students can only absorb certain key concepts in a first course
  - —They need more motivating experience first to be able to relate better to the material

Requirements

Testing

Project management and process issues

Topic With Reduced Focus: Requirements

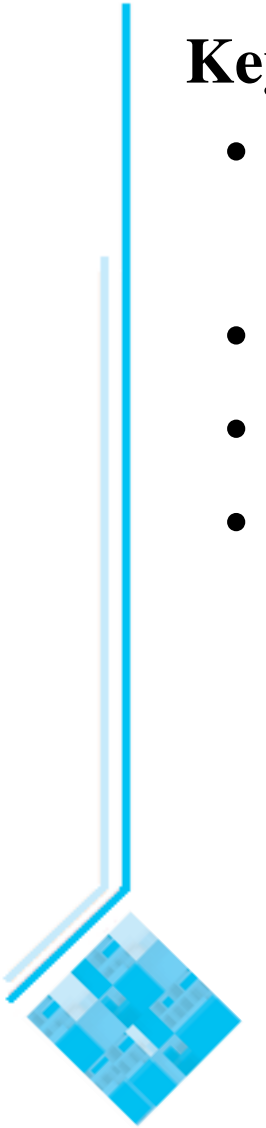
Examples given early, but how to do it now covered only half way through course

- Now taught after modeling

Key concepts emphasized

- Use cases
- Alternatives considered and rationale
- Criteria for reviewing

Topic With Reduced Focus: Testing



Key concepts that remain

- Test driven development
  - —The excitement of getting something working
- Equivalence classes and boundaries
- The challenge of trying to break the system
- Wide spectrum of surprising types of test
  - —E.g. Testing under heavy load, documentation tests

Topic With Reduced Focus: Project Management and Process

Key concepts that remain:

- The difference between agile and waterfall
- Surprisingly long list of tasks that the project manager has to do
- Ad-hoc doesn't work: Disasters that result
- What do key planning tools look like?
  - —Gantt and Pert charts

Topics Currently Covered at the 'Minimal Awareness' Level

Formal methods

- A few examples of OCL
  - —Motivate why discrete math is important
  - —Point out that this helps ensure programs are correct
  - —But deeper knowledge left to later courses

Metrics

- Only basic performance measurements in the lab

UI Design

- It's my favourite topic, but it deserves its own course

Conclusions

Introductory SE can be made interesting and relevant

Keys to good teaching include:

- Use a variety of teaching tactics including live problem solving and live tool use
- Teach a limited number of topics well; don't try to "cover it all"