# Course Overview

Kenneth M. Anderson
University of Colorado, Boulder

CSCI 5828 — Lecture 1 — 01/12/2009

## Goals

- Survey software engineering concepts, terminology and techniques
  - Emphasis on Agile Design Methods
    - Will supplement with traditional/historical material as needed
- Take an in-depth look at software engineering techniques for dealing with
  - concurrency (including model-based techniques)
  - review PL support for concurrency

## CAETE Announcements

- In-Class Students
  - CAETE has a busy studio schedule
    - Be sure to exit promptly so next class can begin
  - Food and drink are not technically allowed!
    - They are tolerated as long as you keep the studio clean!
- Distance Students
  - Textbooks can be ordered from the CU Bookstore
    - Call 303-492-6411 or 800-255-9168
      - Or order on-line at <http://cubooks.colorado.edu/>

## Due Dates

- In the past, due dates for CAETE students were one week behind the due dates for in-class students
  - However, now that lectures are being made available to you in a timely fashion, both in-class and distance students will have the same due dates
- CAETE students need to have a “test proctor” to administer the midterm for them
  - If you don’t know who your test proctor is, contact CAETE to find out (Do this during the first week of classes!)

## A bit about me...

- Associate Professor
  - At CU since July 1998
- Ph.D. at UC Irvine
- Research Interests
  - Software Engineering
  - Hypermedia and the Web
  - Web Engineering
  - REST-based Web Services

## A little bit more…

- 24th semester at CU (!!)
- 6th time teaching CSCI 5828
- Software Development Experience
  - Approximately 16 systems, 30K — 100K LOC each
  - Some industry experience with IBM & Unisys
  - Experience with academic / industry collaboration

## Class Participation

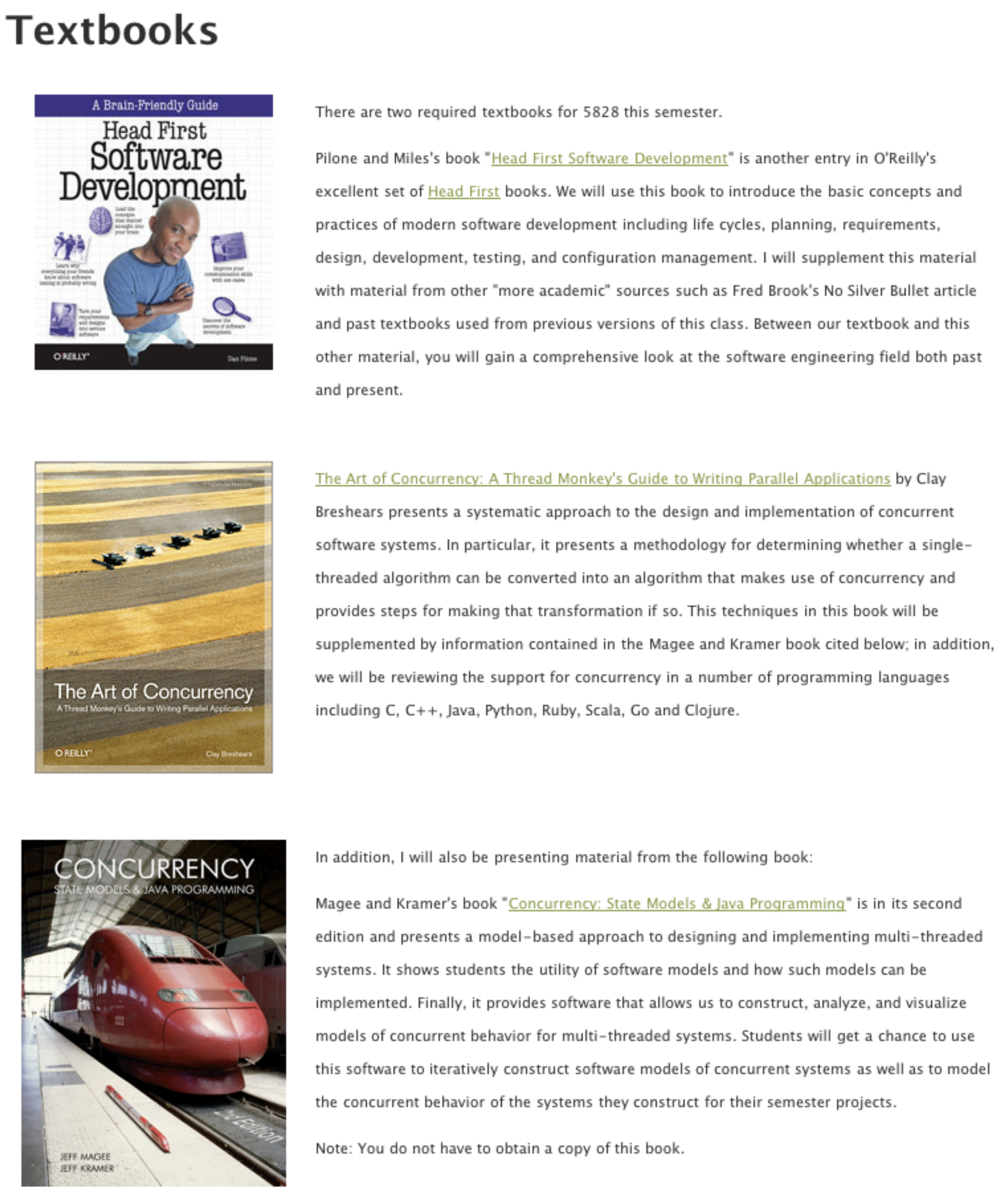
- I welcome contributions to the class by students
  - both in lecture and off-line
- Feel free to interrupt me during lecture to ask questions!
  - “Stupid questions” — No such thing!
  - Engage in discussion otherwise we get the “silent tomb” effect — Boring for you and me!
- If I’m speaking too fast, stop me and ask me to slow down

## Office Hours

- By appointment
  - Send me e-mail at <kena@cs.colorado.edu>
  - Will most likely meet with students in ECCS 127
    - or in the faculty lounge on the 7th floor of the ECOT

## Class Website

Has a What's New page with an RSS feed that you can use to stay current with all course announcements!


A Brain-Friendly Guide
Head First Software Development
The Art of Concurrency
A Thread Monkey's Guide to Writing Parallel Applications
CONCURRENCY
STATE MODELS & JAVA PROGRAMMING
JEFF MAGEE
JEFF KRAMER

### Textbooks

There are two required textbooks for 5828 this semester.

Pilone and Miles's book "Head First Software Development" is another entry in O'Reilly's excellent set of Head First books. We will use this book to introduce the basic concepts and practices of modern software development including life cycles, planning, requirements, design, development, testing, and configuration management. I will supplement this material with material from other "more academic" sources such as Fred Brook's No Silver Bullet article and past textbooks used from previous versions of this class. Between our textbook and this other material, you will gain a comprehensive look at the software engineering field both past and present.

The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications by Clay Breshears presents a systematic approach to the design and implementation of concurrent software systems. In particular, it presents a methodology for determining whether a single-threaded algorithm can be converted into an algorithm that makes use of concurrency and provides steps for making that transformation if so. This techniques in this book will be supplemented by information contained in the Magee and Kramer book cited below; in addition, we will be reviewing the support for concurrency in a number of programming languages including C, C++, Java, Python, Ruby, Scala, Go and Clojure.

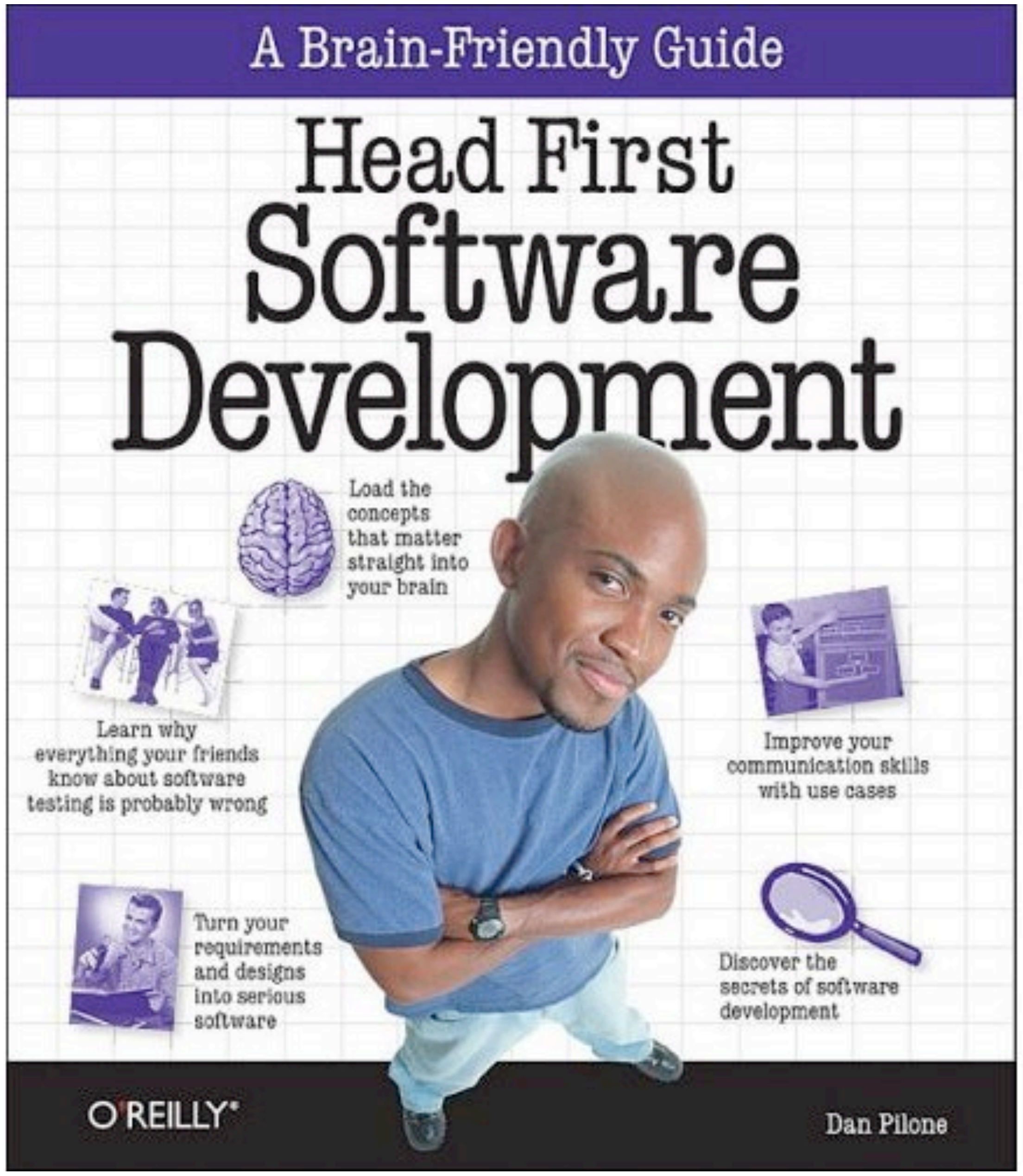
In addition, I will also be presenting material from the following book:

Magee and Kramer's book "Concurrency: State Models & Java Programming" is in its second edition and presents a model-based approach to designing and implementing multi-threaded systems. It shows students the utility of software models and how such models can be implemented. Finally, it provides software that allows us to construct, analyze, and visualize models of concurrent behavior for multi-threaded systems. Students will get a chance to use this software to iteratively construct software models of concurrent systems as well as to model the concurrent behavior of the systems they construct for their semester projects.

Note: You do not have to obtain a copy of this book.

<http://www.cs.colorado.edu/~kena/classes/5828/s10/>

# Textbooks


A Brain-Friendly Guide
Head First Software Development
Load the concepts that matter straight into your brain
Learn why everything your friends know about software testing is probably wrong
Improve your communication skills with use cases
Turn your requirements and designs into serious software
Discover the secrets of software development
O'REILLY
Dan Pilone

Head First Software Development

Dan Pilone & Russ Miles

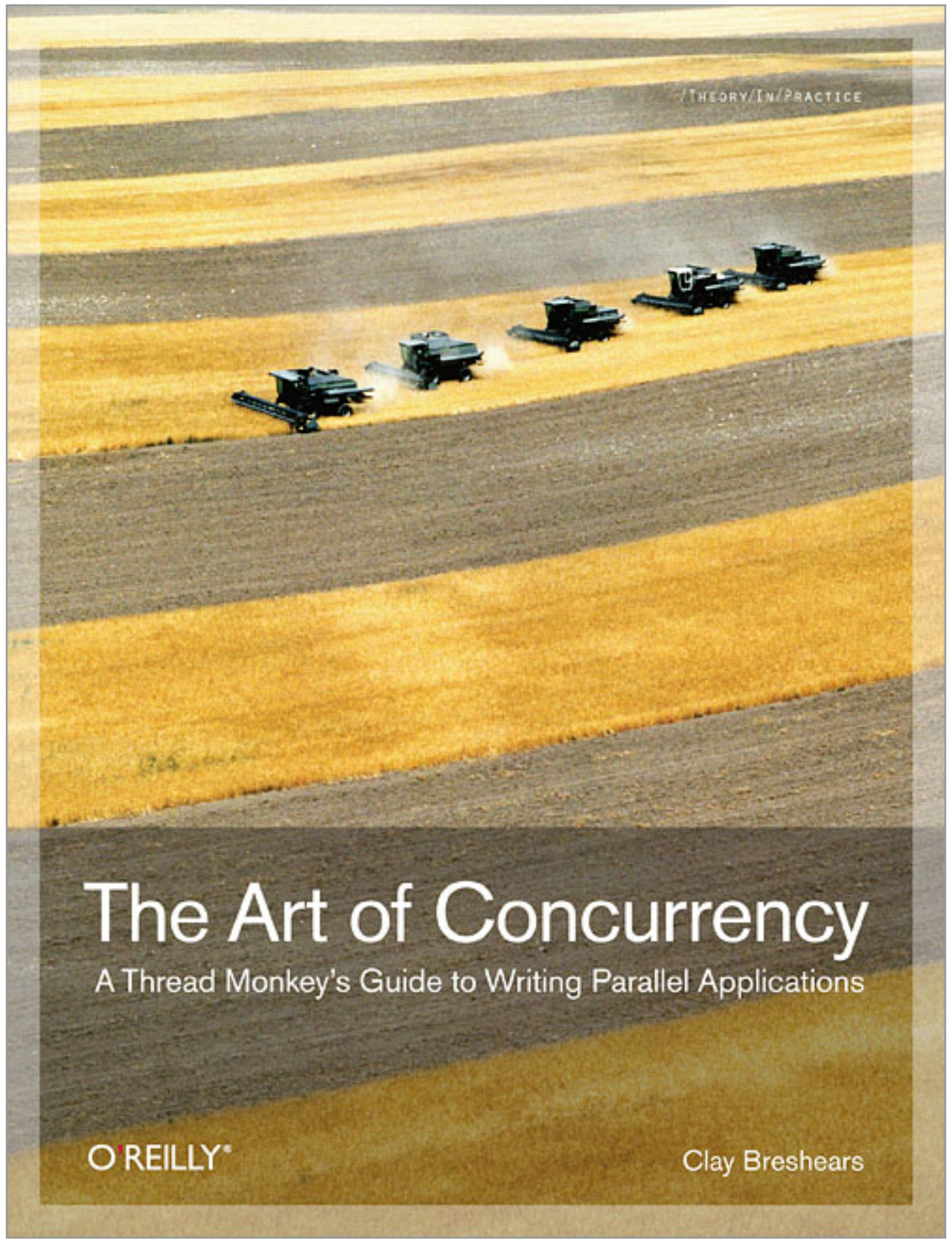
First Published: Jan 2008

Published by O'Reilly

Lots of great examples; decidedly non-academic feel but covers all the essentials

Will supplement with other material throughout semester

## Textbooks



The Art of Concurrency
Clay Breshears

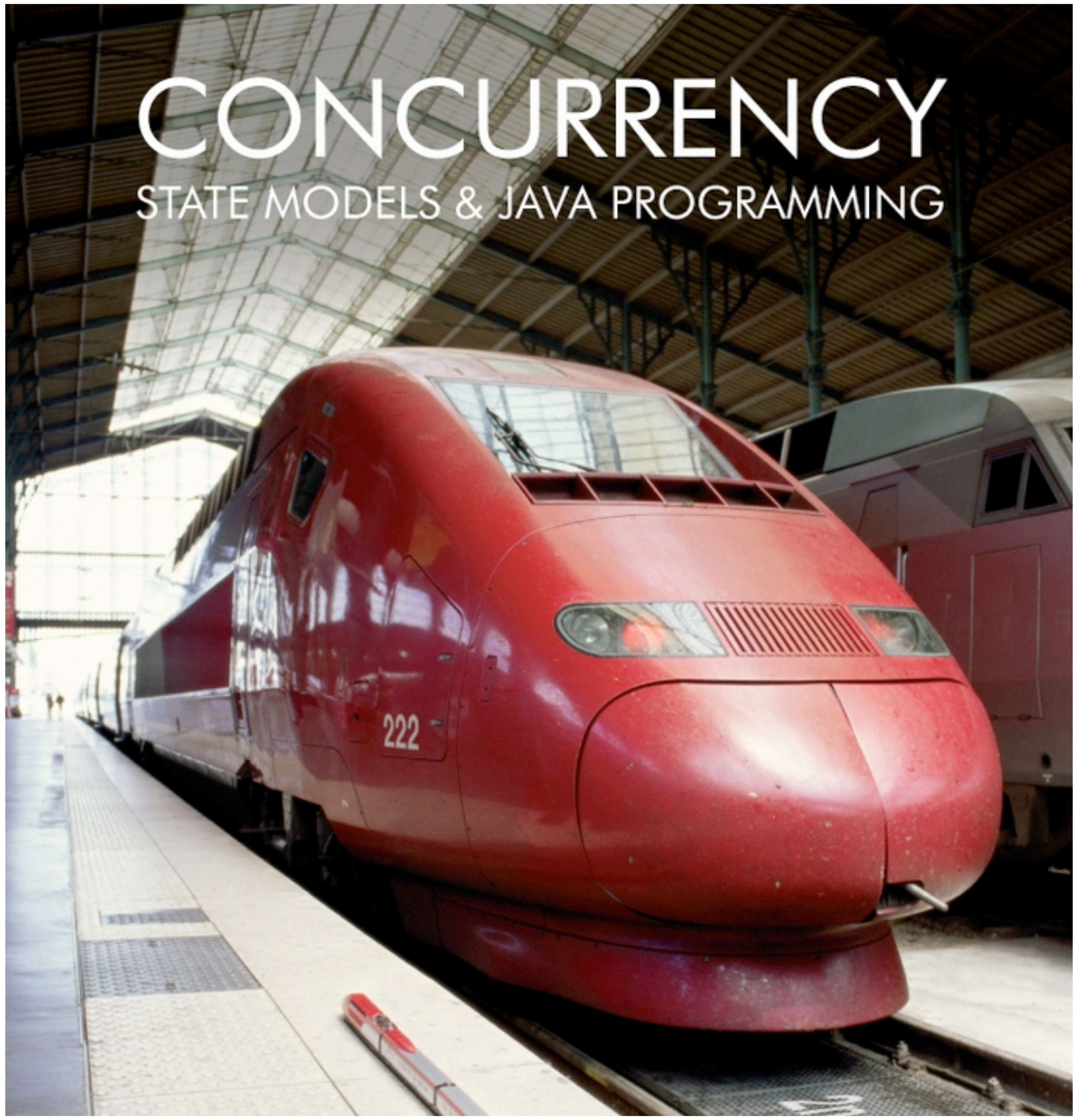
First Published: May 2009

Published by O'Reilly

Main source of material for understanding how to design concurrent software systems

Will supplement with material on concurrency in C, C++, Ruby, Java, Python, Go, Scala, Clojure, ...

## Textbooks



Concurrency by Magee and Kramer (aka “The Two Jeffs”)

Every developer needs to understand concurrency

This **optional** book does an excellent job covering this topic with a software engineering perspective

I will be drawing from this textbook to introduce model-based techniques.

## Structure of Semester

- The course will switch between the first two topics
  - providing an introduction to software engineering one day
  - and looking at issues of concurrency the next
- May also look at additional topics later in the semester

# Course Evaluation

- Your grade will be determined by your work on
  - Homeworks (20%)
  - Midterm (Tuesday, March 2nd; 20%)
  - SE Presentation (20%)
  - Class Project (Due at last class session; student teams; 40%)
- NO FINAL!

# Honor Code

- I encourage collaboration in this class via the homeworks, the SE presentation (2 person teams) and the class project (4 person teams)
  - You're on your own for the midterm!
- The Student Honor Code applies to classes in all CU schools and colleges. You can learn about the honor code at:
  - <http://www.colorado.edu/academics/honorcode/>.

## Late Policy

- Assignments submitted late incur a 20% penalty
  - Assignments can be handed in up to two weeks after the initial due date (except for the final assignment of the class project)
    - after that you are out of luck…

## What is Software Engineering?

- The computer science discipline concerned with developing large applications. Software engineering covers not only the technical aspects of building software systems, but also management issues, such as directing programming teams, scheduling, and budgeting

## What is Software Engineering?

- Software
  - Computer programs and their related artifacts
    - e.g. requirements documents, design documents, test cases, UI guidelines, usability tests, …
- Engineering
  - The application of scientific principles in the context of practical constraints
    - Consider: Chemist versus Chemical Engineer

## What is Software Engineering?

- What is Engineering?
  - Engineering is a sequence of well-defined, precisely-stated, sound steps, which follow a method or apply a technique based on some combination of
    - theoretical results derived from a formal model
    - empirical adjustments for unmodeled phenomenon
    - rules of thumb based on experience
- This definition is independent of purpose
  - i.e. engineering can be applied to many disciplines

## What is Software Engineering?

- Software engineering is that form of engineering that applies…
  - a systematic, disciplined, quantifiable approach,
  - the principles of computer science, design, engineering, management, mathematics, psychology, sociology, and other disciplines…
- to creating, developing, operating, and maintaining cost-effective, reliably correct, high-quality solutions to software problems. (Daniel M. Berry)

## What is Software Engineering?

- Issues of Scale
  - Software engineers care about developing techniques that enable the construction of large scale software systems
- Issues of Communication
  - Consider the set of tools provided by sites like Rally, Fogbugz, or Assembla.com
- Issues of Regulation
  - Other engineering disciplines require certification; should SE?
- Issue of Design
  - dealing with integration of software/hardware/process

## Types of Software Dev.

- Desktop Application Development
- Contract Software Development / Consulting
- Mobile Application Development
- Web Engineering (Development of Web Applications)
- Military Software Development
- Open Source Software Development
- Others??
  - These categories are not orthogonal!

## SE-related Jobs

- Software Developer
- Software Engineer
- SQA Engineer
- Usability Engineer
  - requires strong HCI/CSCW background
- Systems Analyst
  - professional reqs. gatherer/ professional designer
- DBA
- Sysadmin
- Software Architect
- Software Consultant
- Web Designer
- Build Manager / Configuration Management Engineer
- Systems Engineer

## The Big Three

**Specification**

Software Engineers specify everything
**code, requirements, design, process**
What makes a good specification?

**Translation** **Iteration**

## The Big Three

**Translation**

The work of software engineering is one of **translation**
from one **specification** to another
from one level of **abstraction** to another
from one set of **structures** to another
(aka decomposition)

**Iteration** **Specification**

## The Big Three

### Iteration

The work of software engineering is done **iteratively**
**step** by **step**
**task** by **task**
**sub-process** by **sub-process**
until we are **“done”**

### Specification Translation

## The Big Three

**Specification**

**Translation** **Iteration**

## The Big Three

**Specification**

**Translation** **Iteration**

**Are Supported by Many Things**

**Principles, Techniques & Tools**
**Planning, Estimates & Process**
**PEOPLE**

```
import random

class atts(object):

    def __init__(self, name):
        self.name = name
        self.atts = {}

    def __len__(self):
        return len(self.atts)

    def __contains__(self, key):
        return key in self.atts

    def __iter__(self):
        return iter(sorted(self.atts.keys()))

    def __getitem__(self, key):
        return self.atts[key]

    def __setitem__(self, key, value):
        self.atts[key] = value

    def __delitem__(self, key):
        del self.atts[key]

    def __str__(self):
        result = "Name: %s" % self.name
        for key in self.atts:
            result = "    %s : %s" % (key, self.atts[key])
        return result

    def get_name(self):
        return self.name

    def keys(self):
        return sorted(self.atts.keys())

    def return_random_key(self):
        position = random.randint(1, 20)
        return self.atts.keys()[position]
```

Code written by a programmer

Would a software engineer produce the same code?

Just by looking at it, can we see anything wrong with it?

**No documentation?**
**No tests?**
**Is this under version control?**
**What sort of code coverage has been achieved?**

# How to Fix?

- How would we fix the code on the previous side?
  - One step at a time!
    - Place under version control (or configuration management)
    - Add comments
    - Add tests
    - Check for coverage (for python, see: <coverage.py>
- A software engineer would want at least this level of engineering surrounding all components in their system
  - Why? Is this enough?

# SE is HARD

- No doubt about it: software engineering is hard
  - Projects are late, over budget, and deliver faulty systems
    - See 1995 Standish Report for one summary of the problem
- Why?
  - For insight, we will take a look at an article by Fred Brooks called No Silver Bullet
- Please read it by Thursday's lecture
  - Paper is available on IEEE Digital Library: No Silver Bullet
  - I will make PDF available via class website; you will need a password to download it, contact Prof. Anderson for details

http://www.virtualschool.edu/mon/SoftwareEngineering/BrooksNoSilverBullet.html

## Coming Up

- Lecture 2: No Silver Bullet
- Lecture 3: Chapter 1 of HFSD Textbook
- Lecture 4: Chapter 1 of Concurrency Textbook