

# Introduction to Software Engineering

CSE/ISE 308: Software Engineering

SUNY at Stony Brook

Fall 2009

# Course Information

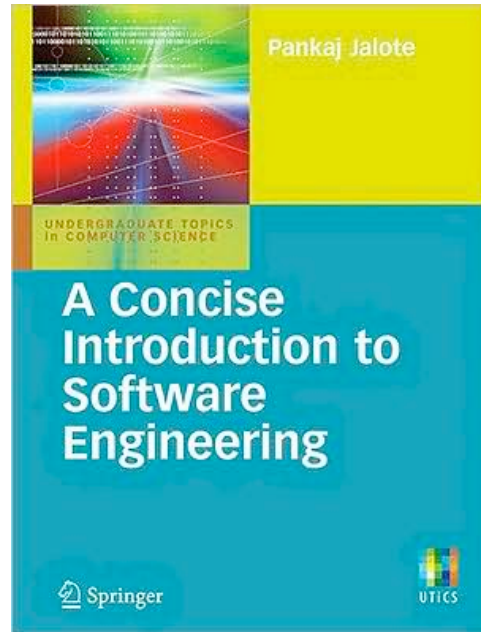
# General Information

- Prerequisite: CSE 219
- Meeting Information:
  - Section 02: Tu/Th, 3:50–5:10 PM, Melville W4535
- Course Web page:  
<http://www.cs.sunysb.edu/~tashbook/fall2009/cse308/>

# Instructor Information

- Michael Tashbook
- E-mail: [tashbook@cs.sunysb.edu](mailto:tashbook@cs.sunysb.edu)
- Office: Computer Science 1402
- Office Hours
  - In person: Tuesday/Thursday, 12:30–2:30 PM
  - Online via AIM (times TBA)

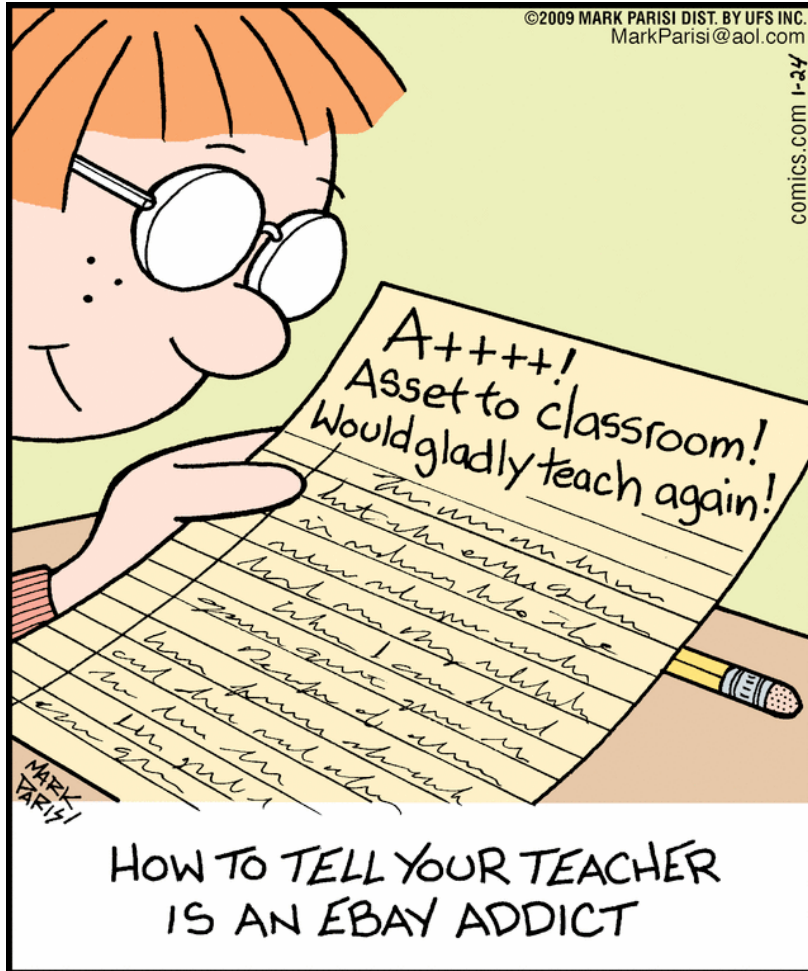
# Textbook



- A Concise Introduction to Software Engineering
- Pankaj Jalote
- Springer 2008
- We will generally follow the topic/chapter sequence in this book, with a few deviations

# Course Components

- Semester (group) programming project
  - This makes up the bulk of your final course grade (approximately 65%)
  - We'll talk more about this on Thursday
- Written Midterm Exam
  - Worth approximately 25% of your final grade
  - Based on your project and other supplied case studies
  - Tentative date: Thursday, October 22, in class
- Class participation (quizzes, etc.)
  - Worth approximately 10% of your final course grade



# Grading Methodology

## **THIS COURSE HAS NO CURVE!**

Each assignment has a point value (equal to its relative grade weight  $\times$  2)

Total for everything: 200 points

Final numerical grade is (total # of points)  $\div$  2

94+  $\rightarrow$  A

84+  $\rightarrow$  B , etc.

Except for highly unusual circumstances, each member of a group will receive the same grade for each project component

# Assignment Submission

- All assignments should be submitted electronically
- Deadlines are NON-NEGOTIABLE
  - Make-ups or extensions require a **VERY GOOD** excuse
- Late or improperly-submitted assignments WILL NOT be graded, or will receive a penalty for lateness



"Well, Jones has a photographic memory. Mine is more ... uh ... mimeographic, which is why it's best for me to copy off him."

# Academic Honesty

You are free to discuss assignments with other students

Anything you submit **MUST** be your own

You are required to cite any outside works that you use

You **MAY NOT** share code or other answers!

# House Rules

- Call me “Mike”
- Please be on time
- Please respect your fellow students
  - turn off cell phones, etc.
  - Questions are always welcome
- Learning (and class) should be fun; if it isn't, let me know ASAP!

# The State of Modern Software Engineering

# The Problem We Face

- Assertion: All software sucks
- Corollary: Not all software sucks equally
- This course is about designing and developing software that doesn't suck (or sucks less than other software)
  - This also applies to development processes

# Why Does Software Suck?

- ❑ It doesn't do what I need
- ❑ I can't figure out how to do what I need
- ❑ This is unnecessarily frustrating/complex
- ❑ This breaks all the time
- ❑ It's so ugly I want to vomit just so I have something prettier to look at
- ❑ It doesn't map to my understanding of the universe
- ❑ I'm thinking about the tool instead of my work

Source: <http://www.scottberkun.com/essays/46-why-software-sucks/>

# How Do You Make Something That Really Sucks?

- ❑ General incompetence
- ❑ Unclear purpose
- ❑ Unplanned design
- ❑ Poor engineering
- ❑ Make customers miserable

# Berkun's Laws

- Law #1: If you don't apply the right skills at the right time, you will make things that suck.
  - Successful project management calls for you to organize the right skill sets for the right parts of the task
  - Sometimes, it's better to start learning a new skill than it is to improve an existing one that you're already strong in
- Law #2: No matter what you do, someone, somewhere will think your software sucks.
  - Translation: Select a (limited) target audience and design for them
  - Whom do you care about? What are their expectations?

# How Do We Define Quality?

- ■ Functionality
- ■ Reliability
- ■ Usability
- ■ Efficiency
- ■ Maintainability
- ■ Portability

# What Is Software Engineering?

# Software Engineering, Defined

1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software
2. The study of approaches as in (1).

— IEEE Standard 610.12

# Characteristics of SE

- Systematic
  - Done according to a system (a set of mechanisms working together)
- Disciplined
  - Behaving in a controlled way
- Quantifiable
  - Able to be measured

# Another Definition of SE

- Software engineering refers to the problem of building and delivering complex software systems on time
- As with other engineering disciplines, this frequently involves ill-defined problems and partial solutions

# What's so hard about SE?

- Complexity

- Software systems perform many functions, comprise many components, involve many developers, are difficult to understand, etc.

- Change

- Software systems are subject to constant change (including correcting errors)

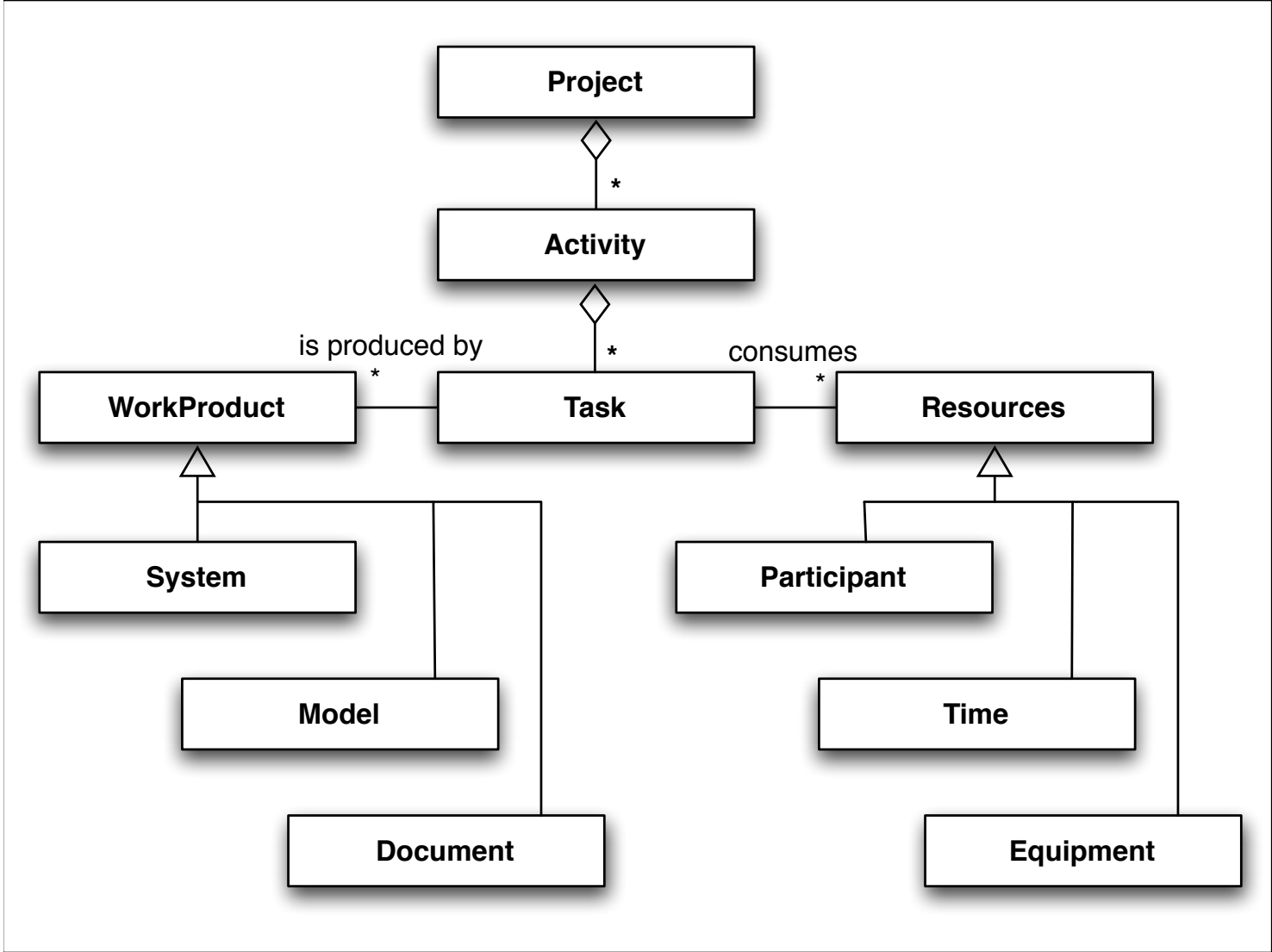
# The Chaos Report (1995)

- 31.1% of projects are canceled before completion
- 52.7% will cost over 189% of their original estimates
- Only 16.2% are completed on-time and on-budget
  - This success rate falls to 9% for large corporations
- Large companies' projects only complete 42% of features
  - 78.4% of small company projects achieve 74.2% feature coverage

# Software Engineering Concepts

# Basic Terminology

- Project: purpose is to develop a software system
  - composed of Activities
- Activities are composed of Tasks
- Tasks consume Resources and produce WorkProducts
- Resources are Participants, Time, or Equipment
- WorkProducts are Systems, Models, or Documents



# System Participants

- Software development requires many people to collaborate
  - Clients order and pay for the system
  - Developers construct the system
  - Project managers coordinate developers
  - End users are supported by the system

# Participant Roles

- Role: a set of responsibilities
- Roles are associated with a set of tasks and are assigned to participants
- A single participant can fill multiple roles
- Ex. Client, User, Manager, Technical Writer, Developer

# Systems and Models

- System: a collection of interconnected parts
- Model: any abstraction of a system
  - Ex. wiring schematics, object models, blueprints
- Development projects can also be modeled
  - L. Osterweil, “Software Processes Are Software Too” (ICSE 1987)

# Work Products

- Work product: an artifact produced during development
  - Ex. documents, pieces of software
- Work products can be internal or deliverables
  - Deliverables must be given to the client
  - Deliverables are usually specified in a contract

# Activities, Tasks, and Resources

- Activity: set of tasks performed for a specific purpose
  - Ex. requirements elicitation, delivery, specification
- Task: an atomic unit of work that can be managed
- Resource: asset used to accomplish work
  - Ex. time, equipment, labor

# Requirements

- Functional requirement: specifies a function that a system must support
  - Ex. an ATM must dispense cash
- Non-functional requirement: specifies a constraint on system operation that is not directly related to a system function
  - Ex. timing constraints, GUI appearance, platform

# Modeling

- Models are abstract representations of systems
- Models allow us to answer questions about systems
  - These systems may not exist physically, or may be too inconvenient to experience firsthand

# Domains of Modeling

- Application domains
  - the environment(s) in which the system has to operate
- Solution domains
  - alternative systems that could be built
  - used to investigate trade-offs of different approaches

# Object-Oriented Modeling

- O-O methods combine application & solution domains
- The application domain is modeled as a set of objects and relationships
- Solution domain concepts are also modeled as objects
- The solution domain model is a transformation of the application domain model

# Software Engineering Development Activities

# O-O SE Development Activities

- Requirements Elicitation
- Analysis
- System Design
- Object Design
- Implementation
- Testing

# Requirements Elicitation

- Purpose: define the purpose of the system
- Result: system description in terms of actors and use cases
  - Actor: external entity that interacts with the system
  - Use case: general sequence of events that describes all possible actions for a given piece of functionality

# Analysis

- Purpose: produce a model of the system that is correct, complete, consistent, and unambiguous
  - Developers transform use cases into an object model
- Result: system model annotated with attributes, operations, and associations (in UML, for example)

# System Design

- Purpose: define project design goals and decompose system into smaller subsystems
- Result: subsystem decomposition and deployment diagram
  - Deals with entities that are beyond the client's interest

# Object Design

- Purpose: define solution domain objects
  - Ex. describing interfaces, restructuring the object model, performance optimization
- Result: detailed object model annotated with constraints and precise descriptions

# Implementation

- Purpose: translate solution domain model into source code
- Result: working source code

# Testing

- Purpose: find differences between the system and its models
  - Unit testing: uses object design model
  - Integration testing: uses system design model
  - System testing: uses requirements model
- Goal is to discover faults before the system is delivered

# Managing Software Development

“Managing programmers is like herding cats”  
— Unknown

# Management Activities

- Management activities focus on planning the project, monitoring its status, tracking changes, and coordinating resources
- Management activities include communication, rationale management, software configuration management, project management, the software life cycle, and software maintenance

# Communication

- This is the most critical and time-consuming activity in SE!
  - Misunderstandings and omissions are difficult and expensive to correct
- Conventions are extremely useful for communication
  - Ex. UML, document templates, Hungarian notation

# Rationale Management

- “Why are we doing this (this way)?”
  - Rationale management justifies decisions
- Rationale is the most complex information to deal with
  - Difficult to update and maintain
  - Capture rationale during meetings and discussions

# Configuration Management

- Process that monitors and controls changes in work products
- Enables developers to:
  - Track changes
  - Control changes

# Project Management

- Oversight activities that:
  - ensure the delivery of
  - a high-quality system
  - on time
  - and within budget

# The Software Life Cycle

- Software life cycle: a general model of the software development process
- We'll discuss traditional and agile methodologies
  - Traditional: Waterfall, Spiral, Timeboxing
  - Agile: Extreme Programming, Scrum