CSE 70: Software Engineering
Welcome, Overview: Past, Present, Future

Ingolf Krueger
ikrueger"AT"ucsd.edu, http://sosa.ucsd.edu

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA

California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA
What are we doing today?

- Get to know each other
- Why become a Software Engineer?
- What is Software Engineering?
- Course Overview
- Administrative Details
- Software Engineering Past, Present and Future
- Why is High-Quality Software Difficult to Build?
Why Become a Software Engineer?

Software Engineering is great fun!
Why Become a Software Engineer?

Why Become a Software Engineer?

Who are we?

Filippo Seracini
TA

Ingolf Krueger

http://sosa.ucsd.edu

Arturo Flores
TA
Who are we: Tutors!

Willy Koswara

David Watson
Project Example: OOI - Complex system of systems

Figure courtesy of Matthew Arrott/ORION CI Executive Summary
What will you learn in this class?

• Know what Software Engineering and its fundamental activities, processes and principles are

• Be able to identify the influences of SE on other areas of CS

• Be able to compare characteristics of development processes for projects of varying scales and complexities

• Be able to execute an agile development process with pair programming

• Be able to identify and operate the major components of a software development and production environment
What will you learn in this class?

- Understand the notion of requirements, and their importance in software development

- Be able to identify and articulate requirements using textual and graphical notations

- Understand the importance of quality assurance and the relationships among various QA techniques

- Be able to write tests, and to execute test-first programming
What will you learn in this class?

• Understand the notion of Software Architecture and its relationship to other activities of SE

• Understand the basic principles of software composition

• Know and apply basic design and architecture patterns

• Know and apply key refactoring techniques

• Know and apply basic UI Design Principles

• Know and exercise SE Ethics

• Know basic challenges and solutions of software deployment, maintenance & update
Administrative Information

Class web page: http://cseweb.ucsd.edu/classes/wi10/cse70/
Administrative Information

• Grading:
  – 25%: Project
  – 25%: Two In-Class Quizzes (at 12.5% each)
  – 20%: Midterm
  – 30%: Final

• Readings:
  – Class homepage (reading announcements there), slides, handouts
  – Course Reader (Soft Reserves)
  – Background material on Java, RMI, UIs: highly recommended, will be posted on the class homepage
Labs

• Run by TAs, **start in week 2**
  – Wednesdays, 1-2:50pm: Lab

• Deepening the understanding of material covered in class

• Lab problems designed to increase proficiency in applying concepts w/tools, programming environments

• Labs also prepare the ground for success in the project

• Labs are not graded, but material will be covered in quizzes, midterm and final
Project

- 4-8 students per team, project start beginning of week 3
- Team assignment by me
- Goal
  - Develop a basic chat system in Java
  - Text-based UI/Graphical UI
  - Running system throughout the project with increasing functionality
- Tools/Languages you will use/learn about:
  - Java, JUnit
  - Eclipse
  - SVN
  - Wiki (Confluence)
  - Issue Tracker (Jira)
  - Continuous Build/Integration (Bamboo)
- Grade will be based on: functionality, tests, overall project performance in team
In-Class Quizzes

- Two quizzes, ~50 minutes each
- Schedule on class home page
- Cover materials from class, readings, lab and project
What is Software Engineering?

NATO Software Engineering Conference
Garmisch, Germany, 7-11 Oct 1968

The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. (Fritz Bauer)

image credit: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/
What is Software Engineering?

NATO Software Engineering Conference
Garmisch, Germany, 7-11 Oct 1968

Dijkstra
McIlroy
McClure

image credit: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/
SE Pioneers: Edsger W Dijkstra

Known for:
• Structured Programming
• THE Operating System
• Semaphores
• Program Semantics
• Shortest Paths Algorithm
• ...

Awards:
• ACM Turing Award

May 11, 1930 – August 6, 2002

Today: programming languages, software architecture, distributed systems, ...

image credit: http://www.cs.utexas.edu/users/EWD/
SE Pioneers: David L. Parnas

Known for:
• Information Hiding
• Modular Programming
• Precise Documentation
• ...

Awards:
• ICSE Most Influential Papers
• Fellow of ACM
• Fellow of IEEE

Today: OO, software architecture, ...

image credit: en.wikipedia.org/wiki/David_Parnas
SE Pioneers: C.A.R. Hoare

Known for:
• Quicksort
• Monitors
• Provable Correctness
• CSP Programming Model
• ...

Awards:
• ACM Turing Award

Today: programming languages, verification, business processes, ...

image credit: www.computerhistory.org
SE Pioneers: Barry Boehm

Known for:
• Cost Estimation (COCOMO)
• Spiral Development Process
• ...

Awards:
• Fellow of the ACM
• Fellow of IEEE
• Fellow of AIAA
• Fellow of INCOSE

Today: RUP, XP, Scrum, project management, ...

image credit: sunset.usc.edu
Quote of the Day

The software field is not a simple one and, if anything, it is getting more complex at a faster rate than we can put in order.

B.W. Boehm*

SE Pioneers: Frederick P. Brooks

Known for:
- Computer Architecture
- OS/360 Operating System
- Mythical Man-Month
- No Silver Bullet

Awards:
- ACM Turing Award
- National Medal of Engineering

Today: Linux, MacOS, Windows, Grid-/Cloudcomputing...

image credit: http://www.cs.unc.edu/~brooks/
SE Pioneers: Tom DeMarco

Known for:
• Structured Analysis
• Peopleware
• Project- / Risk Management

Awards:
• J.D. Warnier Prize
• Stevens Prize
• Fellow of IEEE

Today: Model-Driven Architecture (MDA), project management, ...

image credit: http://www.systemsguild.com/GuildSite/TDM/TDMBio.html
Many others

- Niklaus Wirth (Stepwise program refinement)
- Peter Chen (Entity relationship model)
- Alexander Wolf, Dwayne Perry (Software Architecture)
- Michael Jackson (Structured programming and design)
- Erich Gamma et al. (Design Patterns)
- Kent Beck et al. (eXtreme Programming)
- ...

Huge impact on how we develop software today!
The application of a **systematic**, disciplined, **quantifiable** approach to the **development**, **operation**, and **maintenance** of **software**; that is, the application of engineering to software.

IEEE 610.12
Software Engineering Today

• Is a highly creative activity under project constraints

• Requires strong analytical and “people” skills

• Requires understanding of requirements (what does the customer/user want/need?)

• Requires capability to design, implement, validate and verify a software solution relative to a set of requirements, on time, within budget

• **Software Quality!**
June 4, 1996: “Ariane-5” Catastrophe

- Superfluous calibration program for inertia sensors runs in-flight; the software was “reused” from Ariane-4
- Measured values in Ariane-5 exceed those assumed for the Ariane-4 software.
- The corresponding (Ada-) exception is handled by stopping the control computer and switching over to a redundant system.
- The second system contains the same error and deals with the exception the same way...

| • Costs of the Ariane-5-program until 1996: |
| ~ $ 8 Billion |
| • Value of destroyed satellite: |
| ~ $ 500 Million |
Software Catastrophes Abound

• Financial Catastrophes:
  – 1993: Stopped CA project aiming at integration of systems for driver licenses and vehicle registrations: $44 Mio
  – 1997: Stopped CA project State Automated Child Support System “SACSS”: $300 Mio

• Deadlines:
  – 1994: Denver International Airport delayed 18 Months due to software problems of the luggage transportation system
  – 2003: Toll Collect: start date for German road toll system delayed to 1/2005, claimed losses > 4 Billion Euro

• Technical Catastrophes:
  – 3/1999: Launch failure of Titan/Centaur-Rocket due to wrong software version
  – 9/1999: Loss of “Mars Climate Orbiter” due to wrong unit conversion

• USA: Losses during year 2000 due to software defects: $100 Billion.
Why is High-Quality Software Difficult to Build?

• Software-Quality is difficult to measure

• Software is easily changeable at any stage during the development process

• Implementation platforms and technologies change rapidly

• Complexity is rapidly increasing
  
  – For every 25% increase in problem complexity there is a 100% increase in complexity of the software solution (R. Glass)

• Adequate methods and tools do not exist or are only partially adopted

• Short time-to-market

• Requirements change
Why is High-Quality Software Difficult to Build?

• 0.1%-defect rate means:
  – per year:
    • 20,000 errors in medication
    • 300 failing heart pacers
  – per week:
    • 500 errors in medical surgeries
  – per day:
    • 16,000 lost letters in the postal system
    • 18 airplanes crashed
  – per hour:
    • 22,000 checks posted incorrectly

• Therefore:
  – Massive QA efforts required also in the future.
Software Engineering Today

• Systematic development made possible by advances in
  – Methods & methodologies
  – Languages
  – Tools
  – Processes

• Programming:
  – One person projects:
    Manager, SW Architect, Programmer, Tester, Maintainer,
    Deployer, Customer, User
  – One-off systems

• Software Engineering:
  – Vast number of, often distributed, stakeholders playing different
    roles
  – Complex, long-lived systems
Software Engineering Topics

- Project Management
- Process Modeling
- Software Development Methods
  - Requirements Engineering
  - Software Architecture and Design
  - Software Maintenance
  - Re-Engineering
- Quality Management (incl. Testing)
- Notations and Languages (UML, Java, ...)
- Tool Support (incl. CASE, SVN, make, Ant, Maven)
What have you learned today?

• Software Engineering is fun!
• Software Engineering
  – The establishment and use of sound engineering principles
    in order to economically obtain software that fulfills its
    requirements and runs efficiently on real machines.
• Software Engineering ≠ Programming
• We are standing on the shoulders of giants, literally!
• System complexity is rising
• Productivity & Quality have not yet caught up
• Key challenge: changing requirements!
• Fun lectures, labs and projects await!
Please participate in the survey! You’ll earn two points counting toward the first quiz.

It will help us tailor the class to your needs!

The link is on the class homepage.

Password: available from TAs and me!

Thanks very much!