CSE403 Software Engineering
Autumn 2001

Gary Kimura

Lecture #1

October 1, 2001

~ You are a marvel ~

Each second we live is a new and unique moment of
the universe, a moment that will never be again…
And what do we teach our children?

We teach them that two and two make four, and that Paris is
the capital of France. When will we also teach them what they are?
We should say to each of them:

Do you know what you are? You are a marvel.
You are unique. In all the years that have passed, there has never been
another child like you. Your legs, your arms,
your clever fingers, the way you move.

You may become a Shakespeare, a Michelangelo, a Beethoven.
You have the capacity for anything.
Yes, you are a marvel. And when you grow up, can you then
harm another who is, like you, a marvel?

You must work – we must all work – to make the world
worthy of its children.

Pablo Casals (1876-1973)

Today

- Who I am
- CSE403 educational objectives (and why)
- Overview of the course
- Quick overview of software engineering and why we study it

Who I am

- Gary Kimura (GaryKi@cs.washington.edu)
- Sieg 325E (206) 685-3133
- Office Hours: MWF 10:30 – 11:30 or by appointment
- Email is the best way to contact me
- Background
  - 4 years at DEC
  - 11 years at Microsoft
  - Learned Software Engineering by doing it with one of the best
- Please ask questions during class the lectures should be interactive

CSE 403 rumors and myths

- What is the worst thing that you've heard about CSE403?
- What is the best thing that you've heard about CSE403?
- Why take CSE 403?
  - Is it an easy 4 credits?
  - Learn some of what goes into building a complex software project
- Software Engineering is not rocket science and not something to learn from a textbook
- Software Engineering is the use of common sense and discipline

Educational Objective of CSE 403

- Learn that building large software systems is not a mere matter of programming
- My goal is to have you more than just intellectually know it but really feel it and believe it

How to teach Software Engineering

- There is not a single right way to teach software engineering
- My approach is to teach what I've learned from experience and not simple textbook solutions
- All engineering, including software engineering, is concerned with building useful artifacts under constraints (some people even define engineering to be "design under constraints")
- If it weren't for the constraints, there might be a single right way to engineer software systems

How we will meet our educational objectives

- There is a big multi-person project
- Lectures and sections
  - Monday, Wednesday, and Friday will be more formal lectures with occasional group project days and oral presentations.
  - Thursday section for quizzes and TA discussion
  - Attend lectures is key to get the most out of the class because a lot of the material is not covered in the readings
- Instructor - Been there, done that
- Teaching Assistants - Student advocate, discussion leader, and interpreter <u>of what I say</u>

More on the class

- Project (approximately 20 students, we'll form groups on Thursday)
  - We give you a high-level description, you refine it
  - Strict milestones (some flexibility later on)
  - Weekly reports, due on Fridays; list top 3 risks, plus other material (email is fine)
  - Not primarily graded on whether your program "works"
- Special topics (often guest lectures), last weeks of quarter
- Schedule is on the web page

Grading the course

- Project accounts for 80% of final grade
  - Heavily objective
  - Group members will not necessarily receive identical grades
- Quizzes account for 20% of final grade
  - 4 short quizzes based on readings and lectures

In groups of 2 or 3 people

- Take one minute and write down some possible issues besides simply writing code that might affect how you would engineer software; some examples include
  - Ship your computer game by November 15th or miss the Christmas market — and your company will go out of business
  - The US market for your product is nearing saturation, and you need to start marketing internationally
  - The US government restricts export of a key piece of technology needed by your company to remain competitive
  - Headcount on this project is ten people
  - You have to program in C++, because that's the only language people currently in this shop use, but new hires only know Java.

What is Software Engineering?

- The practical application of scientific knowledge to the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them [Boehm].
- The systematic approach to the development, operation, maintenance, and retirement of software [IEEE].
- The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines [Bauer].
- Multi-person construction of multi-version software [Parnas].
- “Software from womb to tomb.”

Why study Software Engineering?

- Building software without discipline is crazy
- Software is critical to society
- Building a large complete software project is hard
- There is a perceived crisis in our ability to build software
- It’s fun!
- $$$$

Software is critical to society

- Economically important
- Essential for running most enterprises
- Key part of most complex systems
- Essential for designing many engineering products
- In many if not in most cases, the software is embedded in the system you are using — you don’t see the software

Software is big

Code sizes due to Jon Jacky (mostly); KLOC = 1000 lines of code; MLOC = 1,000,000 lines of code

Bar code scanners	10-50KLOC
4-speed transmissions	20KLOC
ATC ground system	130KLOC
Automatic teller machine	600KLOC
Call router	2.1MLOC
B-2 Stealth Bomber	3.5MLOC
Seawolf submarine combat	3.6MLOC
NT 4.0	10MLOC
NT 5.0 (NTFS alone)	60+LMLOC (250K source lines or 100KLOC)

Software it big, so what?

- Delivered source lines per person
- Common estimates are that a person can deliver about 1000 source lines per year (including documentation, scaffolding, etc.)
- Therefore, most complex systems require many people to build
- This would be true even with an order of magnitude increase in productivity
- Hence the critical need for coordination
- Brooks reminds us that people ≠ time

“The Software Crisis”

- We’ve been in the midst of a crisis ever since the 1968 NATO meeting that christened software engineering
- “We are unable to produce or maintain high-quality software at reasonable price and on schedule.”
- “Software systems are like cathedrals; first we build them and they we pray. [Redwine]"

To some degree this is accurate

- We'll look at some classic software failures later on in the quarter
- But it's not fully accurate
- Given the context, we do pretty well
- We surely can, should and must improve
- Some so-called software "failures" are not
  - They are often management errors (the choice not to do something that would have helped)
- In some areas, in particular safety-critical real-time embedded systems, we may indeed have a looming crisis

Some “crisis” issues

- Relative cost of hardware/software
- Low productivity
- “Wrong” products
- Poor quality
- Constant maintenance
- Technology transfer is slow

Why is it hard?

- There is no single reason software engineering is hard—it's a "wicked problem"
- Lack of well-understood representations of software makes customer and engineer interactions hard [Brooks]
- Still a relatively young field
- Software intangibility is deceptive
  - Norman Augustine [Wulf]: "Software is like entropy. It is difficult to grasp, weighs nothing, and obeys the second law of thermodynamics; i.e., it always increases." [Law XXIII]

As the size of the software system grows, the dominant discipline changes (due to Stu Feldman)

Code Size (LOC)	Dominant Discipline
10^3	Mathematics
10^4	Science
10^5	Engineering
10^6	Social Science
10^7	Politics
10^8	Legal?

Your CSE classes to date

- In CSE so far you have mostly implemented carefully defined specifications
- In most cases, the central objective was to build an implementation that satisfied specific speed and space requirements

From a CSE326 assignment

– “For this assignment, you are required to write a program **mzsolve** that finds routes through mazes and solves the 2-coloring problem. Your program will take a maze as input (in a file format described below) and output a solution path for the maze and a strategy for 2-coloring the maze. To accomplish this, you will implement a Maze ADT to represent the maze as a graph. You will implement Maze ADT operations to construct a graph from a maze file, search the graph for solution paths using DFS and BFS, and determine a two color painting scheme for the maze.”

Continued

- **mzsolve.C -** this is the main driver code for the program that we've partially implemented. Mostly, this will just make calls to the Maze ADT and output their results.
- **Maze.h,Maze.C -** these contain a skeleton of the specification and implementation of the Maze ADT as a C++ class. You will implement the internal representation of the the maze. You are free to use any implementation you prefer. Be aware that some representations are better than others, and we will evaluate your choice. The data structure should be able to handle arbitrary mazes (thus, arbitrary graphs). At the minimum, you will need to implement these methods:

continued

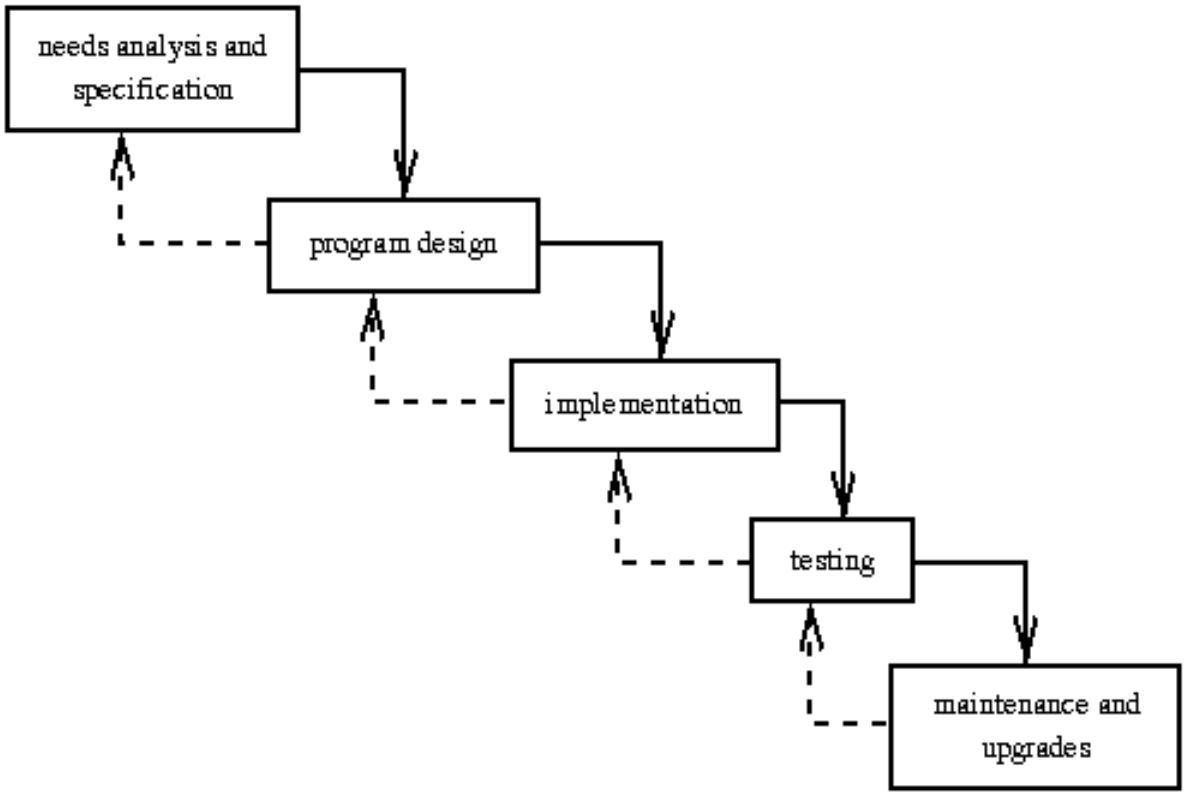
```
Maze(istream &inFile); // construct a maze from input
   stream
   DFSolve(); // solve using DFS
   BFSolve(); // solve using BFS
   TwoColor(); // determine whether a maze is 2-
   colorable
```

- **Makefile - make mzsolve** will use this to build your program.
- You were given...
  (1) …the specification (what to do)
  (2) …the design (roughly, what ADTs to build)
  (3) …hints about the implementation
  (4) …some partial code
  (5) …a pointer to some libraries to use (omitted from the slides)
- Somebody had to figure all this stuff out for you; now that's software engineering, and now it's your turn

Software lifecycle

- A software engineering lifecycle model describes how one might put structure on the software engineering activities
- The classic lifecycle model is the waterfall model (roughly due to Royce in 1956), which is structured by a set of activities and is inherently document-driven

Software lifecycle



Software lifecycle

- Limited feedback increases risk
- “You start at the top and it’s all downhill from there.” [uncertain origin]
- The cost of fixing errors at later lifecycle phases is much higher than at earlier stages
  - Boehm’s 1976 data still hold — the differences are often an order of magnitude
  - Must increase feedback in the lifecycle to reduce these costs (and other risks)

Next time

- Wednesday
  - Models
- Friday
  - Project overview