

6170 · laboratory in software engineering

lecture 1 · february 2, 1999 · introduction

Daniel Jackson
Dept of Electrical Eng & Computer Science
Massachusetts Institute of Technology

introductions

lecturers

- Prof John Chapin (jchapin@lcs)
- Prof Daniel Jackson (dnj@lcs)

teaching assistants

- intro tomorrow

lab assistants

- Ruben Brown
- Charlie Cano
- Richard Li
- Jeremy Lueck
- Sara Pickett
- David Wang

today's handouts

- 1 student info sheet
return today at end of class
used for setting up accounts and assigning to recitations
- 2 general info
- 3 schedule
- 4 problem set 0
- 5 notes coupon (for class text, go to cashier's office then 38)
- 6 software coupon (for software, just take to NE43-529)

dnj -2/13/99 · 3

course obligations

what we expect from you

- attend lectures and recitations
most in first half of term
- problem sets
4 regular assignments
1 individual design exercise
1 team project
- two evening quizzes
- reading

grading

- 75% on individual work
45% problem sets, 25% quizzes, 5% recitation participation
- 25% on final project

workload

- expect median of 15 hours/week
a lot of work, but no way to learn design without doing it!

dnj -2/13/99 · 4

reading

texts

- assigned readings from new Java version of “Abstraction & Specification in Program Development”, Liskov & Guttag
- recommended Java text, no assigned readings “The Java Programming Language”, Arnold & Gosling
- lots of Java examples in “Java in a Nutshell”, David Flanagan

warning

- lecture material *not* covered in readings

drj -2/13/99 · 5

policies

prerequisite

- programming experience in a higher-level language: 6.001
- 200 pre-registered, can take 10–20 more
- will drop: those who don't turn in info sheet or PS0 on time

collaboration

- feel free to discuss problem & approaches
- but solutions must be your own (design *and* code)
- team project: collaborate freely

teams

- write down your prefs; if mutual, staple info sheets together
- team members must be in same recitation
- no recitation switching (but can reorganize teams)

extensions

- none, except under extraordinary circumstances
- see “policies & procedures” for penalties

drj -2/13/99 · 6

what makes good software?

a good software system is...

- delivered on time and on budget
- works reliably
 - easy to use
 - functions mostly as expected
 - performs well enough
 - fails gracefully
 - secure enough
- can be maintained
 - small changes in users' needs can be accommodated
 - by small changes in the code

drj -2/13/99 · 7

what makes a good software engineer?

listens to client

- wants to understand the problem first
- willing to accept criticism

invests wisely

- figures out where the most serious risks are
- doesn't avoid the hard problems by doing the easy things
- writes terse but careful documentation, and values communication
- not afraid of math, but doesn't use formalism for its own sake

works consistently

- knows limitations (how many bugs per kloc?)
- doesn't jump to hasty conclusions

drj -2/13/99 · 8

on simplicity

I gave desperate warnings against the obscurity, the complexity, and over-ambition of the new design, but my warnings went unheeded. **I conclude that there are two ways of constructing a software design: One way is to make it so simple there are *obviously* no deficiencies and the other way is to make it so complicated that there are no *obvious* deficiencies.**

Tony Hoare
Turing Award Lecture, 1980

dnj -2/13/99 - 9

how to keep it simple

avoid skating where the ice is thin

- be wary of clever hacks and tricky algorithms
- don't optimize code until proven necessary
- don't aim to use the most obscure language features

be skeptical of complexity

- be stubborn: it can always be simpler

don't be over-ambitious

- get the simple & important things working first
- avoid creeping featurism

and remember...

- it's easy to make something complicated
- but hard to make something truly simple

dnj -2/13/99 - 10

... but i'd rather be a wizard hacker

some people say

- don't need software engineering if you're smart enough
- more fun just to hack the code, and that's what really counts

why this is misguided

- the ones who say they're smart enough usually aren't
 - sausage and the demise of the Mac
 - actually, none of us are smart enough!
- coding isn't what really counts (more later)
- wizard hackers can't delegate
 - no way to convey design ideas or expertise
 - can't ever build really big things
 - why don't hackers build air-traffic control systems?
- skyscrapers built with dog kennel methods
 - end up being huge dog kennels
- if you wanted just to be a hacker
 - you wouldn't be at MIT spending \$25k/yr!

dnj -2/13/99 - 11

your country needs you!

IBM survey (1994)

- 55% systems cost more than expected
- 68% overran schedules
- 88% had to be substantially redesigned

FAA's Advanced Automation System (1982-1994)

- industry average \$100 ; expected to pay \$500/line
- paid \$700-900
- \$6B of work discarded

Bureau of Labor statistics

- for every 6 new systems put into operation, 2 cancelled
- prob (cancellation) ~ 50% for biggest systems
- average project overshoots by 50%
- 3/4 systems regarded as 'operating failures'

dnj -2/13/99 - 12

effects of bad software

costly

- Smith Barney credits half a million customers with \$19M each (1997)
- Ariane 5 explosion (1996)

tragic

- London Ambulance (1992)
loss of calls, double dispatches from duplicates calls
- Aegis Missile System (1988)
shoots down Iranian Airbus, confusing with F-14

funny

- BMW navigation system directs couple into river (1998)

for more see Risks Forum

- <http://catless.ncl.ac.uk/Risks>

dnj -2/13/99 - 13

everyday bugs

Quicken 1999 bug

"James S. Vera" <vera@anna.stanford.edu>
Tue, 12 Jan 1999 16:44:39 -0800

Another 1999 bug, Intuit's Quicken'99 fails with a "divide by zero" message when a transaction dated in January 1999 is recorded in the Auto category and its "Home and Car Center" is opened. See
<http://www.intuit.com/support/quicken/faqs/win2/1913.html>

James S. Vera | Internet | Voice: +1.415.725.0256
Stanford University | vera@anna.stanford.edu | FAX: +1.415.725.7398

dnj -2/13/99 - 14

everyday bugs

UAL clock wraparound

John Rushby <RUSHBY@csl.sri.com>
Mon 4 Jan 99 23:25:21-PST

I don't know about Y2K, but United airlines has a problem with H24.

This is what www.ual.com provided for the flight status of today's UA63
(scheduled to depart San Francisco at 7:15p and arrive Honolulu at 10:46p)

Flight: UA 0063

Date: 01/04/99

Airport

Time

Status

San Francisco Intl Arpt 9:10pm Mon Delayed 1 hr 39 min

Honolulu Intl 12:01am Tues Early 22 hr 35 min

dnj -2/13/99 - 15

course themes

design, the focus of the course

- only about 30% of software development cost is coding
- design affects all other activities & determines software quality
- programming language matters a lot but
 - can't compensate for bad design
 - can still build good software in a primitive language, eg C
- engineering skills will endure
 - programming language proficiency won't

abstraction, the key idea

- simplify a problem by suppressing irrelevant detail
- we'll use it in two ways
 - component specs: describe behaviour at interface
 - system models: describe system structure & behaviour

dnj -2/13/99 - 16

what's new in 6170 this term

scale

- more focus on large-scale issues
- clean coding still crucial

patterns

- reusable 'micro-architectures'
- expertise = knowing patterns & where to use them

modelling

- good design relies on good representations
- must be more precise than sketches, but still lightweight
- object modelling notations & how to use them

state-of-the-art tools

- CodeWarrior Professional 4, generously given by Metrowerks
- Visio Professional 5.0, generously given by Visio

dnj -2/13/99 - 17

closing comments

remember to

- hand in your info sheet, stapled with mates
- pick up your software & textbook
- read email tonight for recitation assignment

this is going to be fun!

dnj -2/13/99 - 18