6.894 INTRODUCTION

Daniel Jackson · February 2, 2005

today's class

course organization & admin (5 mins)
history of lightweight formal methods (30 mins)
introduction to software abstractions (5 mins)
an example (40 mins)

course organization

web site
› http://sdg.csail.mit.edu/6.894
› readings at 6.894/papers
›
your tasks
› attend class, weekly readings and exercises
› small final project
› collaboration on project and some exercises
› optional quasi-recitation, office hours

grades
› 80% exercises, 20% project
› discretionary: participation, readings, book comments

administrative

signup sheet -- hand in before you leave today!

registration
› no 6.097
› units: 3 (class) - 6 (lab) - 3 (prep)

why 6.894?

timing
- lightweight formal methods is a hot topic nowadays
- gap in MIT curriculum
- I'm completing a book on 'software abstractions'

why take this course?
- become a better software designer
- become a better problem solver
- (get ideas for research)

a brief history of the software crisis

origins, 1960-70

› yikes, this is hard!

> The major cause [of the software crisis] is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.
>
> -- Dijkstra, Turing Award Lecture, 1972

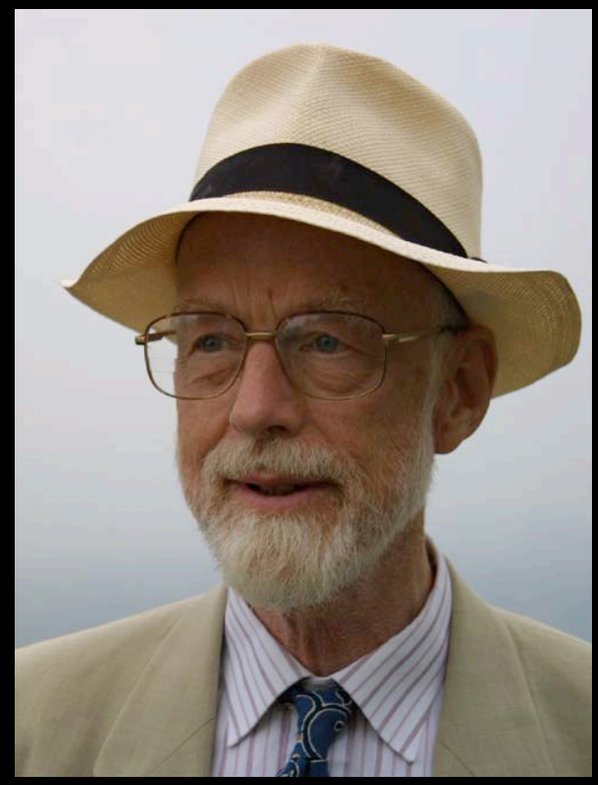
proposed solutions, 1970-1990

programming languages
› strong types, automatic memory, applicative

formal methods
› program verification
› formal specifications

development methods
› SA/SD, JSP/JSD, OMT, etc

process
› waterfall and spiral models
› certification: CMM, ISO9000, etc



reaction from US academia

I think that it's extraordinarily important that we in computer science keep fun in computing. When it started out, it was an awful lot of fun. Of course, the paying customer got shafted every now and then, and after a while we began to take their complaints seriously. We began to feel as if we really were responsible for the successful, error-free perfect use of these machines. I don't think we are.

-- Alan Perlis, quoted in SICP

crisis returns, bigger: 1990's

The demand for software has grown far faster than our ability to produce it. Furthermore, the Nation needs software that is far more usable, reliable, and powerful than what is being produced today. We have become dangerously dependent on large software systems whose behavior is not well understood and which often fail in unpredicted ways.

Information Technology Research: Investing in Our Future
President's Information Technology Advisory Committee (PITAC)
Report to the President, February 24, 1999
http://www.ccic.gov/ac/report/

failed developments

IBM survey (1994)
› 55% of systems cost more than expected
› 68% overran schedules
› 88% had to be substantially redesigned
Advanced Automation System (FAA, 1982-1994)
› industry average $100/line, expected to pay $500/line
› ended up paying $700-900/line
› $6B worth of work discarded
Bureau of Labor Statistics (1997)
› for every 6 new systems put into operation, 2 cancelled
› P(cancellation) ~ 50% for biggest systems
› average project overshoots schedule by 50%

suffering users

I have always wished that my computer would be as easy to use as my telephone. My wish has come true. I no longer know how to use my telephone.
--Bjarne Stroustrup

things to lie awake at night about

devices
› Therac-25, 1985-87: killed 7
› Panama City, 2000-01: killed 17
› new machines are more powerful
› (eg, NPTC at MGH: proton beam, 4 rooms)

infrastructure
› Ted Williams tunnel: huge sensor network

toxic industry
› chlorine tank rupture could kill 100,000+

what's changed?

'going solid' -- no margin
› surgery nixed after anesthesia because no ICU bed

reliance on software
› hospital database issues bad prescriptions (Cook, 04)

tighter coupling
› USS Yorktown dies from divide-by-zero (2002)

bigger software systems
› but no fundamental improvement in methods, yet

classical formal methods: origins

mathematical roots
› 1910's: Whitehead & Russell's Principia Mathematica
› 1920's: Hilbert's program to formalize mathematics

program verification
› Hoare's axiomatic method (1969)
› Dijkstra's weakest preconditions (1975)

formal specification languages (1980's)
› VDM, based on Scott's denotational semantics
› Z, based on set theory
› algebraic approaches: Larch, OBJ, etc

classical formal methods: origins, more

interactive theorem provers

› Isabelle: tactics based on small core of rules
› ACL2 (Boyer Moore): functional LISP, heuristics
› PVS: combines tactics with decision procedures

classical formal methods: outcome

too costly for most developments
› proofs need mathematical gurus
› full specifications too hard to write

verification applied only to limited domains
› small programs over arrays of integers
› no handling of objects, references, encapsulation

origins of lightweight FM's

model checking (1989)
› focus on finding bugs, not proof
› fully automatic, huge state spaces

practical static analyses
› PreFix (Pincus, 1996), ESC (Leino, Nelson et al, 1998)

automatic checking technologies (1990's)
› BDDs, SAT, Simplify

absolutism fades
› partial models, incomplete and unsound analyses

examples of lightweight FMs

Praxis UK
› formal specs + static analysis + tools
› lowest defect rate in industry?

Microsoft's SLAM
› model checking of Windows device drivers
› now used in product development group

Alloy
› developed here at MIT
› used for design-level modelling and analysis

what else is there?

extreme programming (XP)
› test first, refactor, incrementality
› no ‘big design upfront’
› a popular movement of programmers

model-driven architecture (MDA)
› generate code from UML models
› promoted by OMG, IBM and consortium

process management
› document and enforce process, measure defects
› promoted by SEI, industry groups

course topics

software abstractions
› designing & analyzing abstractions with Alloy

model checking
› analyzing concurrent processes

code verification
› writing bug-free code with new checking tools