Software is Discrete Mathematics

Rex Page
University of Oklahoma

Beseme Project

This material is based on work supported by the National Science Foundation under Grant No. 0082849. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

What's the Problem?

- Software is full of bugs
  - Typical: 25 - 50 defects per 1000 lines of code
    - ✓ Discovered in QA or by customers (over product life)
    - ✓ Measured over lifetime of product
  - "High" quality: under 5 defects per 1000 LOC
- Why?
  - Test and debug
    - ✓ ONLY defect prevention strategy used (almost)
  - 90/10 rule
    - ✓ Typical programmer day: 90% keyboarding, 10% thinking Software-centric thinking is 10 to 100 times more cost effective then behavior-centric thinking
    - ✓ 90% thinking, 10% keyboarding would be more productive

Cobb and Mills, IEEE Software, 1990
Humphrey, Addison Wesley, 1995

Think About What?

Software-centric approaches

- Design and code inspections
  - Short learning curve
  - Most students see at least a little of this
- Proofs of software properties
  - Applies real mathematics to software problems
  - Requires skills of real mathematicians
  - Mathematical logic is the primary tool
  - Practical application requires use of proof assistant – ACL2, Coq, HOL, Isabelle, PVS, …
  - Long learning curve
    - ✓ Six months of hard work for proof assistants
    - ✓ Difficult to arrange in industry setting
    - ✓ Would more experience with mathematical proofs help?

Learn Real Mathematics? Where?

Math Courses

Diff'l Calculus
Integral Calculus
Infinite Series
Multivariate Calc
Discrete Math‡
Diff'l Equations
Formal Lang/Automata‡
Statistics
Linear Algebra
Numerical Analysis
Algorithm Analysis‡

‡taught in CS Dept
all others taught in Math

theory and practice are decoupled

So, students don't see math/logic as part of software development

BS Requirements

CS at U Okla

Real Math
that is, proofs

Math Domains

$3\frac{1}{2}$ courses: discrete‡
$7\frac{1}{2}$ courses: continuum

Sfw Courses

Programming I
Programming II
Data Structures
Computer Org
Operating Sys
GUI
Prog Languages
Sfw Engr I
Sfw Engr II
CS Elective
CS Elective
CS Elective

Discrete Math
a missed opportunity

- Right topics, wrong examples (traditional math)
  - ✓ Induction: $\sum k$, $\sum r^k$, $Fibb_n = \sum C(n-k, k)$, …
  - ✓ Trees: unique path, edges are cuts, n-1 edges, …
- Textbooks
  - ✓ Rosen
  - ✓ Grimaldi
  - ✓ Scheinerman
  - ✓ Washburn *et al*
  - ✓ Many others …
- Arguments for traditional approach
  - ✓ Math is interesting and trains students to think
  - ✓ Students need to know math
  - ✓ Math relates to computer science
  - ✓ All true, but …
- Arguments against traditional approach
  - ✓ Eyes glaze over on day 1
  - ✓ Students fail to connect discrete math with software
  - ✓ Missed opportunity to practice use of math in programming

Software-Oriented Discrete Math

- Real math, with examples chosen from software
  - ✓ Induction: properties of software components
  - ✓ Trees: databases, grammars, games, …
- Textbooks
  - ✓ Hall and O'Donnell
  - ✓ Grassmann and Tremblay
  - ✓ Gries and Schneider
  - ✓ Hein (to a lesser extent)
- Arguments for software-oriented approach
  - ✓ Covers same topics as traditional course
    - Boolean algebra, propositional and predicate logic, induction, sets, functions, relations, trees, graphs, combinatorics
  - ✓ Students practice using logic to reason about software
  - ✓ Practice may improve programming effectiveness
- Arguments against software-oriented approach
  - ✓ Students find it demanding (lots of proofs)
  - ✓ Most instructors must revise notes

Course Content

- Propositional calculus 25%
  - ✓ Natural deduction (proof trees)
  - ✓ Equational reasoning
  - ✓ Boolean algebra

  with automated proof checkers
- Predicate calculus 10%
  - ✓ Software raises its head
- Mathematical Induction 35%
  - ✓ Induction $P(0) \wedge (\forall n.P(n) \rightarrow P(n+1)) \rightarrow \forall n.P(n)$
  - ✓ Strong induction $(\forall n.(\forall m<n.P(m)) \rightarrow P(n)) \rightarrow \forall n.P(n)$
  - ✓ Well-founded induction (on trees)
  - ✓ Loop induction (Floyd/Hoare)
  - ✓ Correctness + termination, resource analysis
- Other topics 20%
  - ✓ Sets, relations, functions, graphs, combinatorics
- Introductory and review lectures 10%

Example: Concatenation Conserves Length

Assume insertion (:), concatenation (++), and length satisfy

(x: xs) ++ ys = x: (xs ++ ys)	{equation 1 ++}
[ ] ++ ys = ys	{equation 0 ++}
length(x: xs) = 1 + length xs	{equation 1 length}
length[ ] = 0	{equation 0 length}

- Prove $\forall$xs. P(xs)

  where P(xs) $\equiv \forall$ys. length(xs ++ ys) = length xs + length ys

- Inductive case: P(xs) $\rightarrow$ P(x: xs)

length((x: xs) ++ ys)	
= length(x: (xs ++ ys))	{eq 1 ++}
= 1 + length(xs ++ ys)	{eq 1 length}
= 1 + (length xs + length ys)	{induction hypothesis, P(xs) }
= (1 + length xs) + length ys	{+ assoc}
= length(x: xs) + length ys	{eq 1 length}

length :: [a] $\rightarrow$ ~~Int~~ Integer

- Base case: P([ ]) — cites {eq 0 ++} and {eq 0 length}

Software Examples from Lectures

- sum
- and
- or
- length
- ++
- concat
- maximum
- vector addition
- perfect shuffle
- deal
- merge
- merge sort
- quick sort
- exponentiation
- binary tree search
- AVL tree insertion
- dot product

❑ **Significant properties verified**

✓ Lots of practice in reasoning about software

✓ Standard discrete math topics covered in software context

❑ **What students take away from the course**

✓Concern for software correctness

✓Adequate skills for proving software correctness? Probably not

✓Habit of thinking, not just typing? Yes

What Has the Beseme Project Produced?

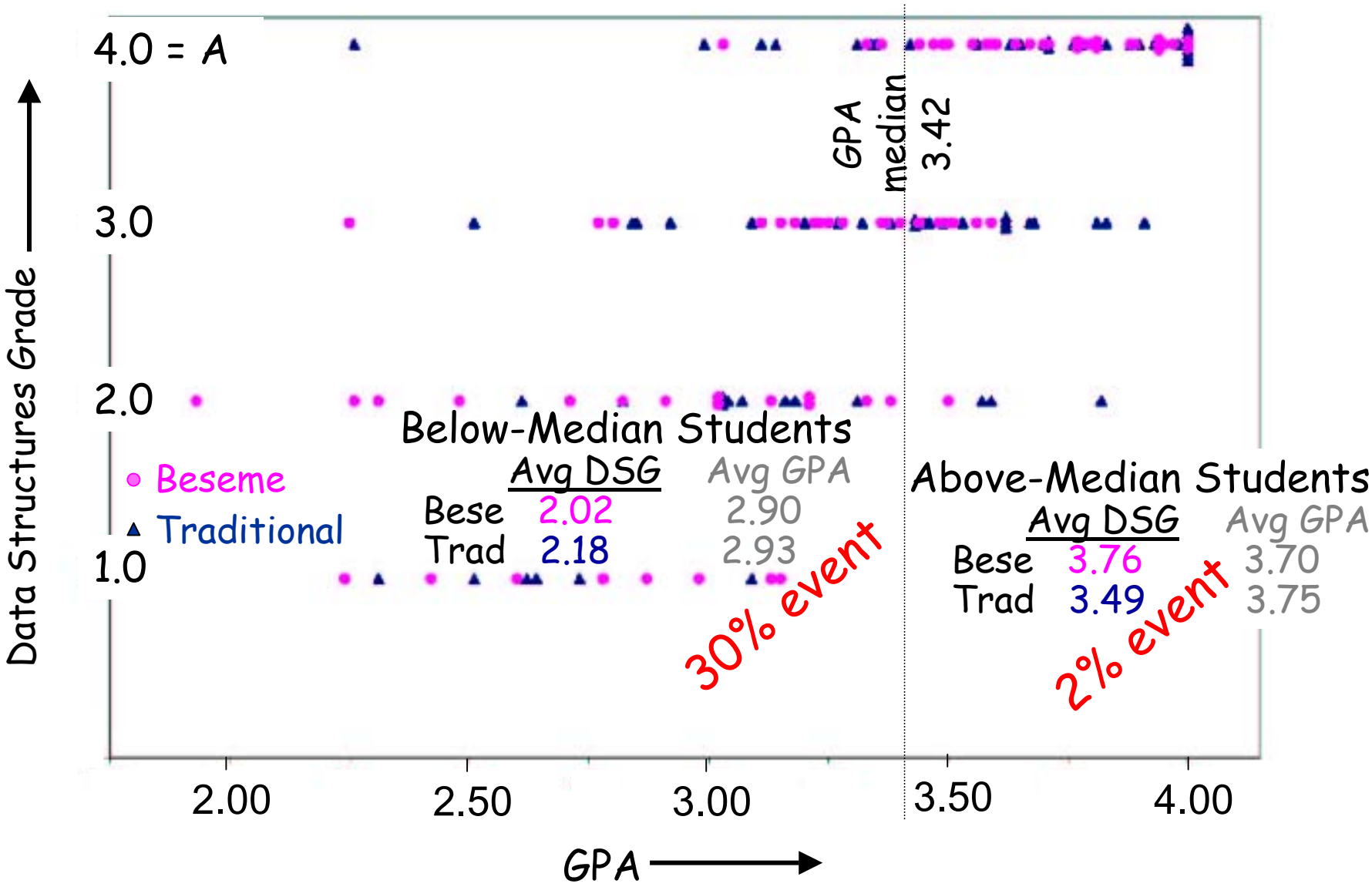
website: Google to "Beseme"

- Course materials accessible via web
  - About 350 slides in 29 lectures
    - ✓PowerPoint and PDF
  - 100 homework problems and solutions
  - 150 exam questions and solutions
  - Proof-checking tools (propositional calculus)
- Partial access open to public
- Full access limited to instructors
  - Because of exams, homework, solutions, etc.

What About Assessment?

- Estimate differences in programming skills
  - Three year project — Sep 2000 – Aug 2003
  - Data: GPAs, Grades in Discrete Math and Data Structures, …
  - Compare Traditional disc math (control grp) versus Beseme
    - ✓ Use Data Structures grade as estimate of programming skills
    - ✓ Note: Discrete Math is prerequisite for Data Structures
  - Detectable differences in grades in Data Structures?
    - ✓ Null hypothesis: both groups have same average grade in DS
- Population size
  - Discrete Math: 150 students per year
  - Data Structures: 120 students per year
  - Leakage: transfer students, advanced standing students, …
  - Expected database size (spring, 2004): 250 students
  - Current database: 150 students
- Statistical method
  - Estimate probability of observed difference in means
    - ✓ Assuming null hypothesis is true, and using Student's t statistic
  - If probability < 5% … Reject null hypothesis

Statistical Results



If Difference is Significant … What Causes It?

- Better students in Beseme sections?
  - Compare average GPAs
    - ✓Beseme students: 3.25
    - ✓Traditional students: 3.35
- Better instructor in Beseme sections?
  - Students' assessment of instructors (0.0 – 4.0 scale)
    - ✓Beseme instructor: 2.17 average
    - ✓Traditional instructors: 2.83 average

    Beseme instructor must be tough grader, eh?

2.79 Bese

2.84 Trad

Average DM grade awarded by instructor

- Course content?
  - More emphasis on logic helps?
  - Software-based examples?
  - More experience constructing proofs?

Where Is This Going?

Replacement Courses

Math Courses		Sfw Courses
Diff'l Calculus		Programming I [d]
Integral Calculus	Predicate Calculus + induction, sets, …	Programming II [d]
Infinite Series	Trees, Graphs, Grammars	Data Structures [d+i]
Multivariate Calc	Pi Calculus	Operating Sys [i]
~~Discrete Math~~		Computer Org
~~Diff'l Equations~~	Add ENGR Core	GUI
~~Formal Lang/Automata~~	Circuits	Prog Languages
Statistics	Signals/Systems	Sfw Engr I [i or d]
Linear Algebra	Statics/Dyn/Therm	Sfw Engr II [i or d]
Numerical Analysis	…	Tech Elective
Algorithm Analysis	FE Exam	Tech Elective
		Tech Elective

Core

Hard Core Software Engineering

a la McMaster Univ

BS program

[d]declarative
[i]imperative

Software Engineering

- Engineering (according to Merriam Webster, ABET, …)
  - Applying scientific and mathematical principles in the construction of useful artifacts
- Software Engineering
  - Webopedia: discipline concerned with developing large computer applications
  - Applying scientific and mathematical principles in the construction of software
  - Such as by using mathematical logic to construct and analyze software models

FAQ

- You don't really think proofs are feasible for real software do you?
  - Yes.
    - ✓ Long-term goal: Provide a basis in education for success using logic-based software/hardware verification
    - ✓ Short term: Shift just a little towards reasoning from the current overwhelming dominance of test-and-debug
- This is old hat … they were talking about it in the 1960s
  - They were talking about oop then, too … takes a while to catch on
  - Dijkstra, Hoare, Backus, McCarthy, Milner, Moore … can't be wrong
  - FP makes it more feasible … machines are powerful enough for FP now
  - Proof assistants that tie proofs directly to code are practical now
- Why functional programming instead of real programming?
  - Proofs tied to code aren't yet practical for imperative paradigm
  - Functional programming is practical: fast hdw, good compilers
- Why Haskell? Wouldn't Scheme or Java be an easier sell?
  - Probably
    - ✓ My usual excuse: Haskell looks more like standard math
    - ✓ Another excuse: forces functional code – if they can avoid it, they will
    - ✓ Logic and reasoning really count: programming language is secondary

The End