Engineering Software Correctness

Rex Page
University of Oklahoma

supported by
National Science Foundation
Grant No. EIA 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.

FDPE Sep 2005
Engineering Software Correctness

a report on teaching a required, two-course software engineering sequence
for 4th year students

Observations (from 2+ years classroom experience)

- Undergraduate students can use a mechanical logic to prove properties of their programs
  - Well ... 30% just go through the motions
  - 40% acquire basic skills
  - Top 30% get good enough to use it on the job
  - Top 10% become accomplished theorem provers

Opinions

- Theorem provers are ready for prime time
- It won’t happen if we don’t teach it
Why ACL2?

* A Computational Logic for Applicative Common Lisp
  (purely functional subset of Common Lisp)

- Students can succeed early
- Integrated programming language and logic
  - Same syntax
  - Theorems side-by-side with code
  - Dual use of functions
    - Stating theorems
    - Specifying computations
- Fast - completes proof in a few seconds or fails
- Good documentation
  - Online tutorials and user’s guides
  - Well-written textbook (Kaufmann/Moore/Manolios)
  - Good email helpline
Engineering Software Correctness

presentation outline

- Course content
- How did this course evolve?
- Software projects assigned
- Potential improvements
- Student reactions
Software Engineering Courses at OU

- Two 3-credit courses in successive semesters
  - Required for CS baccalaureate
- Calls for both individual work and team work
  - 1st semester: 2/3 individual, 1/3 team
  - 2nd semester: 1/3 individual, 2/3 team
- Three themes
  - Processes - Humphrey PSP
  - Design - Component architecture
  - Quality - Testing + Mechanized logic (ACL2)
    - Low defect rate
- Collateral effect (in recent offerings of the course)
  - Significant experience in functional programming
ACL2 Coverage

- 8 of 22 lectures devoted to ACL2
  - Verification as a part of software development
    - Stating/proving correctness, standard part of process
    - ACL2 has two roles: programming language & mechanical logic
  - Logic examples focus on correctness properties

(defun my-take (n xs)
  (if (or (zp n) (atom xs))
      nil
    (cons (car xs) (my-take (- n 1) (cdr xs)))))

(include-book "sources/books/arithmetic-2/meta/top")
(defthm take-append-identity
  (implies (true-listp xs)
    (equal (my-take (length xs) (append xs ys)) xs)))

(defthm drop-append-identity
  (implies (true-listp xs)
    (equal (nthcdr (length xs) (append xs ys)) ys)))

Most common mistake: stating false theorems
(as in programming, what you first believe is true often isn't)
Relevant Properties Depend on Usage

- List of nodes from tree
  
  ```lisp
  (defun flatten (tr)
    (if (atom tr)
        (cons tr nil)
        (append (flatten (car tr)) (flatten (cdr tr)))))
  ```

- An important property: conservation of nodes
  
  ```lisp
  (defun occurs-in (x tr)
    (or (and (atom x) (atom tr) (equal x tr))
        (and (atom x)
            (not (atom tr))
            (or (occurs-in x (car tr))
                (occurs-in x (cdr tr))))))
  ```

  ```lisp
  (defthm flatten-conserves-atoms
    (iff (occurs-in x tr)
      (and (atom x) (member x (flatten tr))))
  ```

- Properties related to order
  - Maybe important, maybe not ... depends on application
  - Additional theorems needed if order properties are important
Correctness Is Relative

statements of relationships among functions

(defun packets (d xs)
  (if (atom xs)
      '(nil)
      (let* ((split (break-at d xs))
          (first-packet (car split))
          (rest (cadr split)))
        (cons first-packet
          (if (atom rest)
              nil
              (packets d (cdr rest)))))))

(defun packet-n (n d xs)
  (take-to d (drop-past-n-delimiters n d xs)))

(defthm packets-thm
  (implies
   (and (true-listp xs) (integerp n) (>= n 0))
   (equal (packet-n n d xs)
     (nth n (packets d xs))))

proof in lecture goes
through about a dozen lemmas
**A More Extensive Example**

**AVL trees - insertion, deletion, search**

(defun avl-insert (avl-tree key) ... )
(defun avl-delete (avl-tree key) ... )
(defun avl-search (avl-tree key) ... )

(defun treep (maybe-tree) ... )
(defun balanced-treep (tree) ... )
(defun ordered-treep (tree) ... )
(defun avl-treep (maybe-tree)
  (and (treep maybe-tree)
       (balanced-treep maybe-tree)
       (ordered-treep maybe-tree)))

(defthm insertion-preserves-AVLness
  (implies (avl-treep tr) (avl-treep (insert tr key))))

(defthm insertion-conserves-keys
  (implies (avl-treep tr)
    (iff (or (occurs-in k tr) (= k new-k))
         (occurs-in k (insert tr new-k))))

operators

"AVL-ness" properties

correctness properties

preservation law
(similar law for delete)

conservation law
(similar for delete)
Engineering Software Correctness
presentation outline

- Course content
- How did this course evolve?
- Software projects assigned
- Potential improvements
- Student reactions
Route Taken to Arrive at ACL2

- 1995 - 2003 (taught SE-I/II six times)
  - Software projects in conventional language
    - C, C++, Java, Tcl/Tk, MS Word macros
    - SE-II team project: 6000 to 12000 LoC
- 2003-2004 SE-I/II
  - Scheme + ACL2 (SE-II project: 3000 - 6000 LoC)
    - ACL2: purely functional + mechanized logic (Boyer-Moore)
    - A Computational Logic for Applicative Common Lisp
  - Uses of ACL2
    - Confirm certain properties of functions
    - Functions defined in Scheme, converted to ACL2
  - Many lapses into conventional programming
    - Not satisfactory as an experience in functional programming
  - ACL2 only
  - Advantages
    - Excellent experience in functional programming
    - Some experience in software verification
  - Disadvantages
    - No interactive programs — file I/O only
    - No higher-order functions
Engineering Software Correctness
presentation outline

- Course content
- How did this course evolve?
- Software projects assigned
- Potential improvements
- Student reactions
Overview of Projects

**SE-I**

- **Individual projects**
  - Six to ten small software projects (100 to 500 LoC)
  - Deliverables for each software project
    - Design
    - PSP report (time estimate, time log, defect log)
    - Code
  - One or two other written reports
- **Team projects (30% of class periods allocated for work in teams)**
  - Cooperative effort on 800 to 1200 LoC code
  - Design and code reviews
  - Comparative summary of individual PSP reports

**SE-II**

- **Team project: software product (3000 - 6000 LoC)**
  - About a dozen separate deliverables for the team project
    - Designs, estimates, reviews, standards, presentations, testing, assessment, etc
  - All but a few class periods allocated for work in teams
- **Individual project: a development journal**
  - Weekly progress reports (also given orally in weekly mtgs)
  - PSP reports and theorems on software components
Topics of Projects in SE-I

2004

0. List ops, towers of Hanoi, Newton's meth / no thms
1. Mean, variance, histogram
   Thm: histogram invariant with permutation of input
2. Fibonacci three ways, essay
   Thm: nested=tail, incr seq
3. Concordance – indiv & team
   Thm: sorting fn correct
4. Word frequency
   Thm: \( \sum \text{frequencies} = 1 \)
5. Token count as is vs inlined (using AVL) / Thm: open
   Team Proj: stock market data and analysis query (huge S&P file) / Thm: open

2005

0. List ops, arctan, Machin, Newton's meth / no thms
1. Text encryption
   Thm: decode inverts encode
2. Linear regression
   Thms: \( s v_i = s v_i \)
   running average = \( \sum \div n \)
3. Text in image – indiv & team
   Thm: decode inverts encode
4. ??
5. ??
   Team Proj: ??

Project Success Rates

- Code – 90% of students succeed
- Thms – 50% - 90% (70% typical)
Team Project in SE-II (2005)

- **Image Calculator**
  - **Basic image operations**
    - Remove background (-)
    - Superimpose (+)
    - Scale (÷)
    - Filter (convolution)
    - I/O
  - **Formula specifies image transform**
    - Lambda-like syntax
    - Intrinsic and/or defined operations

- **Deliverables**
  1. Project time estimate
     - Based on high-level design
  2. Engineering standard
  3. Design review
  4. Detailed design, revised est.
  5. Design presentation
  6. Code review
  7. Implementation
  8. Usage description and installation instructions
  9. Test suite
  10. Product presentation
  11. Acceptance test strategy
     - For another team’s software
  12. Acceptance test presentation
  13. Meeting log

First six weeks - 5 deliverables
Next six weeks - 1 deliverable
Last four weeks - 7 deliverables
Designing ACL2 Projects for SE

- Educational goals
  - SE Processes (PSP)
  - Design (component architecture, ACL2 implementation)
  - Quality (mechanically verified properties)

- Limitations imposed by ACL2
  - File I/O only
  - No floating point (exact rational, instead)
  - Mechanically provable termination

- Strategy for designing practicable projects
  - First two years - projects tested in summer REUs

- Rule of thumb for project design
  - File I/O only
  - Specify required theorems
    - So, certain functions must be required
  - Write critical functions and prove critical theorems
    - Before giving assignment
  - Supply basic i/o operations
Engineering Software Correctness
presentation outline

- Course content
- How did this course evolve?
- Software projects assigned
- Potential improvements
- Student reactions
Things I Might Do Differently

- **SE-I Projects – Software Properties Theorems**
  - Guidance on useful properties needed
    - At least in the beginning
    - SE-I has only 7 software projects
      - So, it’s all “in the beginning”
  - All project write-ups must require specific theorems

- **SE-II Projects – Software Properties Theme**
  - Specify some theorems
  - Describe an overall correctness theme
    - Students can develop some theorems
      - But need some kind of objective

- **Expanded mechanical logic coverage**
  - Proof hints, inductive measures, guards
  - Rule classes – rewrite, type prescription, elimination
Engineering Software Correctness
presentation outline

- Course content
- How did this course evolve?
- Software projects assigned
- Potential improvements
- Student reactions
Question Posed to Students

anonymous responses (instructor evaluation form)

Suppose ACL2 had the following features
(it doesn’t, but try to imagine that it did)

- As fast as C
- Able to build executables (.exe, .dll, ...)
- A good (in your opinion) interactive development environment, a la VisualStudio (or DrScheme)
- You had as much experience in ACL2 as in any other programming language
- Higher-order functions
- An efficient “float” facility for using approximate numbers

Under the above conditions

- Would you choose ACL2 for a major project?
- Why or why not?
Would you choose ACL2?

- Yes - 14 responses
  - Really like FP - 7
  - Mechanical logic benefit - 7
- No - 14 responses
  - Like OOP better - 4
  - Limited application domain - 4
  - Mechanical logic is useless - 2
  - Too few FP programmers - 2
  - Won’t get me a job - 2
- Couldn’t tell - 10
  - No response - 7
  - Response too vague to interpret yes or no - 3

Success rate is more-or-less typical, in my experience
- Over a third of the students acquire solid understanding
- Over half at least get the point
  - ACL2 application domain is limited by lack of GUI support
  - “Too few FP programmers” is a thoughtful answer
    (uncommon programming environment complicates staffing)
  - “Won’t get me a job” probably true (shallow, but not false)
- About a third just muddle through, somehow
The End