

Inculcating Invariants in Introductory Courses

David Evans
and Michael Peck

University of Virginia
ICSE 2006 Education Track
Shanghai, 24 May 2006

www.cs.virginia.edu/evans



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. I think we're responsible for stretching them, setting them off in new directions, and keeping fun in the house...

Alan Perlis (forward to Abelson & Sussman, *Structure and Interpretation of Computer Programs*)

How does software engineering education fit into a computer science education?

Inculcating Invariants - David Evans, University of Virginia

2



First Computing Course

- Fun programs
- It works on my input once
- One programmer
- One friendly user
- A few hours work
- No consequences of failure

"Customer Complaints"

- Working programs
- It always works on our inputs
- Many programmers
- Dumb, evil users
- Over many years
- Failure means lives and fortunes lost

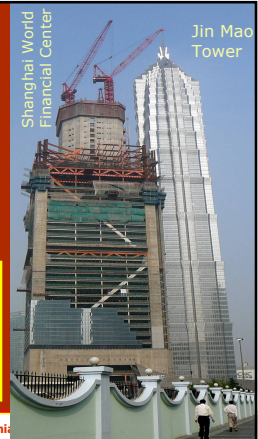
Inculcating Invariants - David Evans, University of Virginia

3



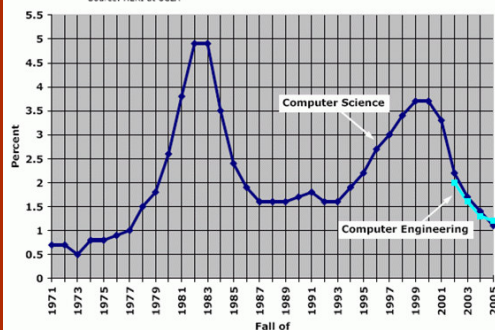
Strategies

1. Incorporate "customer complaints" into first course: Static typing, extensive testing, exceptions, formal specification, etc. ("Brick Laying")
2. Keep first course "fun": Big, exciting conceptual ideas in first course, try to fix bad habits later (Jin Mao Tower)



Inculcating Invariants - David Evans, University of Virginia

CS and CE listed as probable major among incoming freshmen
Source: HERI at UCLA



<http://www.cra.org/wp/index.php?p=75>

Inculcating Invariants - David Evans, University of Virginia

5



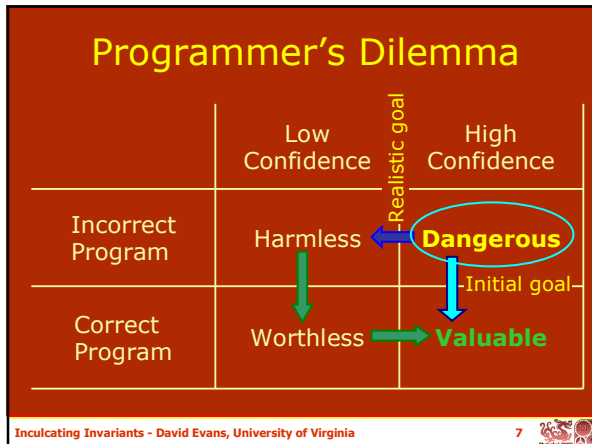
Shock therapy to break bad habits



Inculcating Invariants - David Evans, University of Virginia

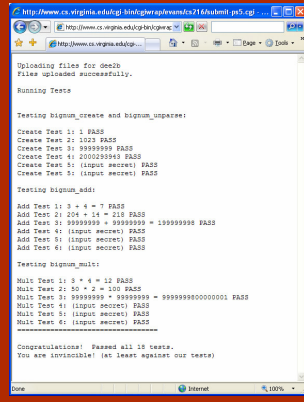
6





"Unfriendly" Testing


- Public test cases
- Secret test cases
- Interactive secret test cases
 - See when they fail, but not the tests



Inculcating Invariants - David Evans, University of Virginia 8

Gambling

- Capture the costs of defects, and value of confidence
- Bet up to 20 points on correctness of code – lose 2x bet if incorrect
- What is "correct"?



Inculcating Invariants - David Evans, University of Virginia 9

Philosophy

"This generation of students got into [college] by doing exactly and precisely what teacher wants. If teacher is vague about what he [sic] wants, they work a lot harder to figure out what they want and whether or not it is good. The vaguer the directions, the more likely the opportunity for serendipity to happen. It drives them nuts!"

Harvard Professor John Stilgoe
(on *60 Minutes*, 4 January 2004)

Inculcating Invariants - David Evans, University of Virginia 10

Correctness

- Code matches the specified behavior
- When the specification is vague or ambiguous, it matches what a rational (but unfriendly) "customer" expects
 - Unless student clarifies the specification
- Results:
 - 8 bet 0, 2 of them correct
 - 14 bet 2-10, 10 of them correct
 - 3 bet 20, 1 of them correct

Inculcating Invariants - David Evans, University of Virginia 11

Program Analysis Tools

- Motivation and feedback for documenting invariants
- Becoming widely used in industry
 - Microsoft requires all Windows developers to annotate their code
- Detect problems that are hard to find in testing

Inculcating Invariants - David Evans, University of Virginia 12

ESC/Java

- Extended static checking tool for Java
- DEC/Compaq/HP SRC [Leino 2001]
 - ESC/Java 2 [David Cok and Joe Kiniry]
- Assumptions documents using syntactic comments
- Produces warnings for code that could produce run-time errors



Documenting Assumptions

- Functions
 - Pre-conditions: `//@requires index < numEntries`
 - Permitted modifications:
`//@modifies numEntries`
 - Post-conditions:
`//@ensures numEntries == \old(numEntries) + 1;`
- Objects
 - Invariants
`//@invariant els.containsNull == false`



ESC/Java Warnings

```
AverageLength.java:7: Warning:
Array index possibly too large (IndexTooBig)
String filename = args[0];
                    ^

AverageLength.java:18: Warning:
Precondition possibly not established (Pre)
String name = names.getNthLowest (index);
                        ^

Associated declaration is "./StringTable.spec",
line 47, col 10:
//@requires index < numEntries;
```



Real Challenge

- How can you make documenting assumptions useful enough (for small programs) so students do it while they are developing instead of after?
 - Interactive secret tests can help some, but most students still put off writing annotations until their code appears to work



Annotating Programs

- Major difficulty is getting formal syntax right
 - Students understand invariant and can express it informally, but can't find right annotation

Can dynamic inference tools help?



Dynamic Inference Tools

- Guess likely invariants by examining test executions
 - Some invariants produced are wrong
 - Some needed invariants will be missed
- Daikon [Ernst+ TSE 2001]
 - Can produce ESC/Java annotations
- Perracotta [see Jinlin Yang's talk tomorrow]
 - Infers simple ordering properties



Experiment

- Based on experiment by Nimmer and Ernst [FSE 2002]
- Provide students with programs with Daikon-produced annotations
 - Two programs, two versions of each
 - Some correct annotations, some incorrect, some missing
- Students given 30 minutes per program to correct annotations
- Collected traces of their ESC/Java executions



Experiment Results

- Students rarely removed correct annotations (only 1 removed in entire experiment)
- Most students removed incorrect annotations that produced ESC/Java warnings
- Some added correct annotations, but most had trouble with complex ones



Conclusions

- Tool interfaces, clear feedback really matter
 - Eclipse front end to ESC/Java helped
- Writing formal specifications (even if you call them “annotations”) is still hard
 - Good tools can make the payoff immediate enough
- Dynamic inference tools might help
 - Side benefit: reveal weak test suites



Questions

- Should we teach software engineering in first CS courses?
- If not, how can we recover from the bad habits students learn in early courses?

Send me your ideas for an ISCE 2007 panel.

