CSC 509 Lecture Notes Week 9

The Dream of Formal Verification

or is it Delusion?

I. Administrative Matters

A. *Recap of Remaining work:*

1. Assignment 6 -- last set of readings.
2. Assignment 7 -- in-class presentations.
3. Assignment 8 -- final project/paper.

Administrative Matters, cont'd

B. ***Presentation scheduling:***

1. See wiki.
2. Note no diff between one and two-person teams.

Administrative Matters, cont'd

C. ***Final Project Deliverables:***

1. See wiki.

2. Tune up brief summary if necessary.

-- Now onto the Dream --

II. Mathematical Foundations of Computing

A. Computer hardware and software are both based on mathematical logic.

B. Founders of computer science were logicians.

C. So were inventors of programming language concepts still in use today.

III. How does program verification fit into the scheme of things?

A. Began with modern programming languages.

B. John McCarthy's 1962 paper
"Towards a mathematical theory of computation"

C. Robert Floyd's 1967
"Assigning Meaning to Programs"

How verification fits in, cont'd

D. Hoare's 1969
"*An Axiomatic Basis for Computer Programming*"

E. Hoare and Wirth's 1973
"*An Axiomatic Definition of the Programming Language PASCAL*"

IV. How Does Verification Relate to Testing?

A. *Testing:* show that a program is correct for*n* ***some finite set of inputs.***

B. *Verification: prove that a program is correct for* ***all possible inputs.***

Now onto the "Reliable Without Proof" and "Verified Software Initiative" papers

First the 1996 "Without Proof"

V. Noteworthy citations from the "Reliable" paper:

A. Hmm, he didn’t give any.

B. Can he get away with that?

VI. **Who is Hoare guy, anyway?**

A. "Sir Tony" to his friends.

B. Among other things:

1. Inventor of Quicksort.
2. Co-Inventor of program verification.
3. Inventor of CSP (mentioned in Week 4 notes).
4. 1980 ACM Turing award winner.

VII. Some Starter Questions

A. Does Hoare like testing OK, or does he still pine for formal verification?

B. Back in 1996, how prescient was Hoare about the TDD thing?

Some Starter Questions, cont'd

C. Overall, how do Hoare's analyses say square with Agile-style testing?

D. Does Hoare say that we can do without formal methods altogether, or just the proof stuff?

E. We'll consider these and other questions further ...

VIII. Section 1 -- Introduction

A. *"... in spite of appearances, modern software engineering practice owes a great deal to the theoretical concepts"*

Section 1 -- Introduction, cont'd

B. The plain fact -- *"the current state of industrial practice lags behind the research we did in the past. Twenty years perhaps?"*

C. Twenty years ***at least!***

IX. Section 2 -- Management

A. *"Above all, the strictest management is needed to prevent premature commitment to start programming as soon as possible."*

B. *"This can only lead to a volume of code of unknown and untestable utility,"*

C. Do these statement contradict agile testing??

Management, cont'd

D. *"... inspections, walk throughs, reviews and gates are required to define important transitions between all subsequent phases in the project ..."*

E. This is precisely SCRUM, though Hoare may have envisioned it happening monthly rather than daily.

Management, cont'd

F. And there's this -- *"... there is now increasing experience of the benefits of introducing abstract mathematical concepts and reasoning methods into the process, right from the beginning."*

G. See, e.g., 2009 IEEE Computer article:
"Formal Versus Agile: Survival of the Fittest?"

Management, cont'd

H. The gist of the paper:

"Success in the use of mathematics for specification, design and code reviews does not require strict formalization of all the proofs."

Management, cont'd

I. Some appropriate humility:

"It [formal reasoning] is not immune from failure; for example simple misprints can be surprisingly hard to detect by eye. Fortunately, these are exactly the kind of error that can be removed by early tests."

Management, cont'd

J. Still a bit of a dream

"Informal reasoning among those who are fluent in the idioms of mathematics is extremely efficient, and remarkably reliable."

Management, cont’d

K. And finally

"More formal calculation can be reserved for the most crucial issues, ..., where bugs would be most dangerous, expensive, and most difficult to diagnose by tests."

X. Section 3 -- Testing

A. "Thorough testing is touchstone of reliability"

B. "Tests are applied as early as possible"

C. "They are designed rigorously"

D. "If a test fails, it is not enough to mend the faulty product."

XI. So, how do does testing work so well?

A. Skeptics may still say it doesn’t

B. Per E.W. Dijkstra *"program testing can reveal only the presence of bugs, never their absence"*

C. And QA folks say "you cannot test quality into a product". How then can testing contribute to reliability of programs, theories and products?

D. Is the confidence it gives illusory?

XII. **But it evidently does work.**

A. *"The resolution of the paradox is well known in the theory of quality control."*

B. *"It is to ensure that a test made on a product is not a test of the product itself, but rather of the methods that have been used to produce it"*

But it does work., cont'd

C. *"The first lesson is that the test strategy must be laid out in advance."*

D. *"The real value of tests is not that they detect bugs in the code, but that they detect inadequacy in the methods, ... and skills of those who design and produce the code"*

But it does work., cont'd

E. He goes on to point out the specific benefits of black box and regression testing (remember this is 1996).

XIII. Section 4 -- Debugging

A. He presents a cute Mosquito analogy.

B. Vicious biting mosquitos are killed quickly.

C. Gentle non-biting ones lurk on.

D. Cf. the known "gentle" bugs in, e.g., `gcc`

XIV. **Coverage**

A. Hoare preaches the virtue of coverage.

B. Did he read an early version of the 1997 Zhu paper?

Coverage, cont'd

C. A bit early to have read a draft of the 2009 MS research paper, but he boldly states their conclusions anyway:

"Equally unfortunately, total coverage is found to be necessary: more errors continue to be discovered right up to the last line tested."

Coverage, cont'd

D. He also gets in his bit about OPs --

"... most general-purpose programs are only used in highly stereotyped ways ..."

Coverage, cont'd

E. And some more from 2009 --

"Suppose a hundred new errors of this kind are detected each year. Crude extrapolation suggests that there must be about half a million such errors in the code. [Undetected errors] play the same role as the swarms of the gentle kind of mosquito that hardly ever bites."

Coverage, cont'd

F. Ah, but

"The less fortunate corollary is that if all the errors that are detected are immediately corrected, it would take a thousand years to reduce the error rate by twenty percent."

Coverage, cont'd

G. And how little we seemed to have progressed in the last 20ish years:

"Unfortunately, before that stage is reached, it often happens that a new version of the system is delivered, and the error rate shoots up again."

XV. Section 5 -- Over-engineering

A. He mentions *defensive programming*

B. Also *software audits*, which are nowadays called "code reviews".

C. In general, he considers over-engineering to be a sub-optimal tactic for reliability.

XVI. Section 6 -- Programming Methodology

A. Things are bit dated here.

B. He talks about *structured programming*

C. Also the benefits of strong typing (Python folks, are you listening?)

’Programming, cont’d

D. And don’t forget *information hiding*

E. He (tries) to argue that more languages have evolved from formally specified progenitor (e.g., Algol 60) as opposed to less formal ones (e.g., COBOL). Alas, he’s left C out of the mix."

XVII. Section 7 -- Conclusions

A. He notes again the 20 year gap (at least) between theory and industrial practice.

B. He laments the fragmentation of researchers into competing tribes.

C. He describes, in so many concluding words what will be his 2009 manifesto

Now onto "Verified Software Initiative"

XVIII. The 1996 paper conclusions are the direct lead-in to the 2009 manifesto.

A. What's needed to make verification happen?

What's needed, cont'd

1. Researcher collaboration.
2. Tools.
3. Industrial partners.
4. More edgimicashun.