Testing Philosophy
Personal perspective

Yuri Gurevich, Microsoft Research

Apr. 18, 2015. Redmond to London

THE JOURNEY

Russia, the math period

- First there was math, the one area where you can actually prove something to somebody.
- Some applications came along as I had to support myself.
- Discovering logic, solving old logic problems, applying logic to math. and collaborating with industry

Israel, the modern-logic period

- Failure in forging industrial collaboration
- Wanted to turn to CS but discovered the wonderful logic group in Jerusalem.
- Studying and working on (modern) logic, model theory and set theory
  - Also having a brush with other subjects, e.g. game theory, operation research
- Eventually I decided to restart my CS journey.

Michigan, the CS period

- Working in Vol. A and Vol. B computer science. But it is the second that is more relevant here.
- What is CS about?
  - Algorithms. But what's an algorithm?
- Teaching Pascal.
- Declarative vs. executable
- What's an algorithm?
  Can one marry abstraction with executability?

What's a state of an algorithm?

- Informally: the info that, together with the program, determines the execution from now on.
- Formally: structures in the sense of mathematical logic.
  - Re VDL, graphs would not do.
- This paves a way to defining the abstraction level of the algorithm and to abstract state machines.

ASM thesis

For every algorithm there is a behaviorally equivalent ASM.

- Eventually proven for large strata of algorithms
- The thesis calls for applications.

Academia vs. industry

- Math and natural sciences are done primarily in academia. But CS is neither.
- The CS/SE split is historical, not intrinsic. CSE is one discipline.
- My attempts to connect with industry.
  - Ironically it was my Vol. A research that brought me to Microsoft.

The group on Foundations of Software Engineering

- Hiring (of me and then) of FSE folks
- The one focus (which is unusual in MSR): ASM tools.
- ASML, the ASM language
  - Purpose: to specify software on any abstraction level
  - The Haskell story
  - Intrinsically parallel and randomized

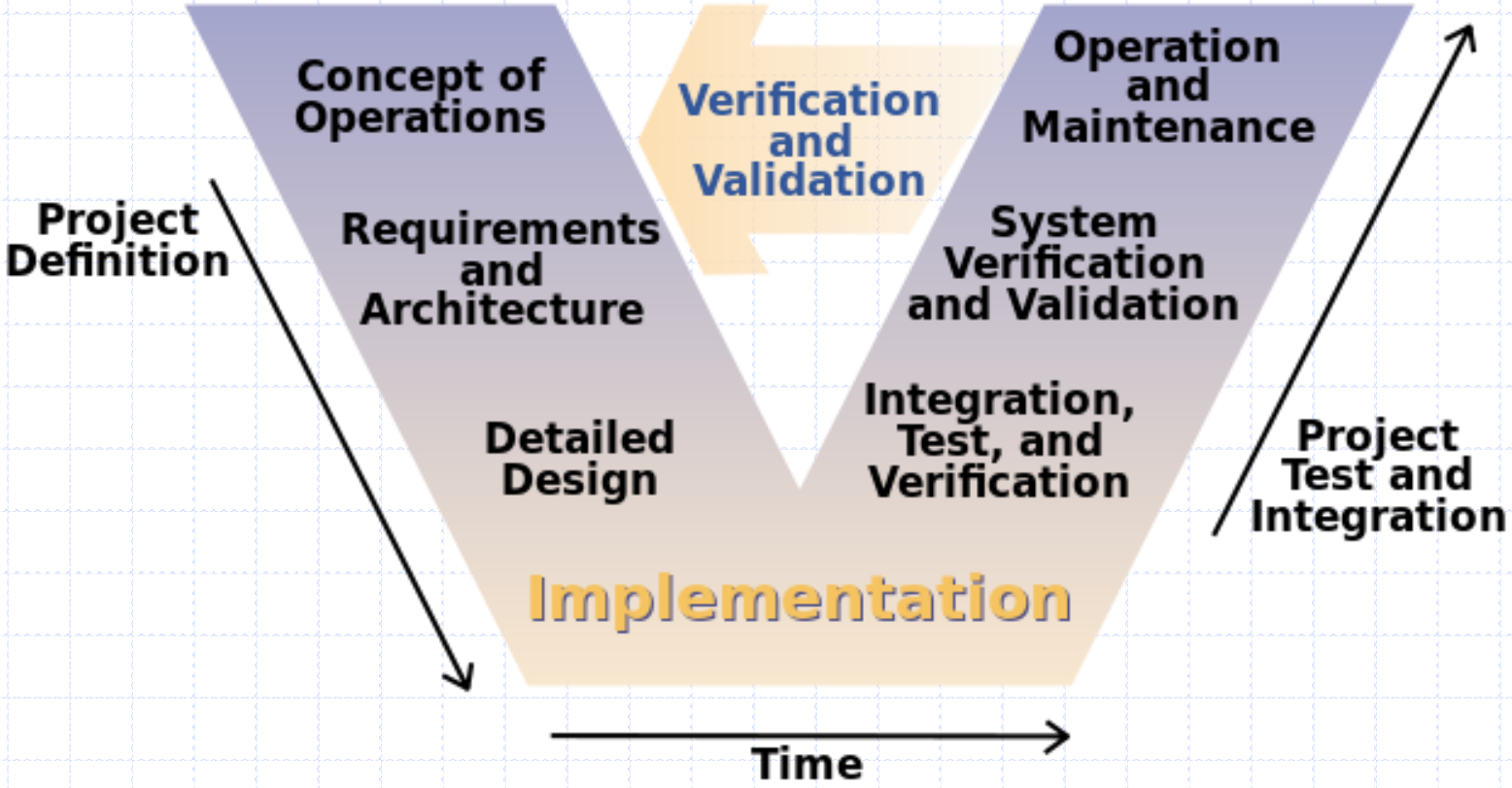
Original problem: poor specs

- A poorly specified design is hard to debug and to code
- The resulting code is hard to test for functionality
- Poorly specified components are hard to assemble
- The stuff is just hard to understand

Enter testing

- Customer problems tend to become your problems.
- What's testing?
  - It's hopeless. Undecidable, and NP-hard even if you deal with Booleans.
  - A little engineering goes a long way.

V-model (from Wikipedia)



A lower part
(from Wikipedia)

Requirement analysis	Aceptance testing
System design	System testing
Architecture design	Integration testing
Module design	Unit testing

Source code

“Love triangle”

1. Architects and devs
2. FSE
3. Testers

Spec Explorer

- The hard problem of tech transfer
- EU to rescue
- Model based testing

Slowing down on testing

- Enough management, writing up stuff
- Other engagements
- Eventually quanta

DIVERSITY OF TESTING

A partial taxonomy

- Black box, white (or glass) box, grey box
- Unit (or module), feature; integration; system
- Alpha (in house), beta (by end users)
- Poking around, manual, automated script, model based
- Acceptance, usability
- Functional; compatibility; regression; performance: load, stress
- Comparison; install, uninstall; recovery
- Combinatorial
- Incremental
- Sanity, smoke
- Random
- Security, privacy, compliance
- Conformance (of implementation to the spec or the other way round)

Exhaustive testing

- Exhaustive testing in the black-box case.
- What is exhaustive testing in the white-box case?
  - Program coverage?
  - Anything else?

Orthogonal ways of testing

- on the example of ASM to FSM

Math is useful

- Pairwise testing
- A particular testing game

TESTING AS A GAME

The players

- A two-player game.
  1. Programming team
  2. Testing team
- Could one entity have two hats and play both roles?

The goals

- Programmer's goal is to make his program work as desired.
  - To do what it is supposed to do.
  - Not to do what it is not supposed to do.
- Tester's goal is to find bugs, not to prove program correct.
  - Successful tests discover errors.
    Cf. positive diagnostic in medicine.
- It is much preferable that the two players are distinct.

But what if there are no bug?

- Maybe all the bugs have been found.

Some other questions

- How is the winner determined?
  - Not all games are antagonistic.
- What is a move?
- When is the game over?

TESTS AND PROOFS

Testing is inferior to verification

- Dijkstra:

  "Program *testing* can be used to show the presence of bugs, but never to show their absence!"

- That sits well with mathematically inclined folks.
  - The Montreal story

Don’t underestimate testing

Donald Knuth:

> "Beware of bugs in the above code;
> I have only proved it correct,
> not tried it."

Testing is superior to verification

- This is true in a kind of trivial way: verification is just a kind of testing.
- In fact, in industry the term "verification" is rarely used in the sense of a mathematical proof of correctness.
- Same elsewhere: "Trust but verify."

MYERS'S PARADOX

Glenford Myers in The art of software testing

"The number of uncovered bugs in a program section is proportional to the number of discovered bugs in the section."

Is the paradox true or false?

- Here is a counterexample with two one-line sections:

  sum(1,1) := ~~3~~ 2
  sum(1,2) := 2

- So the paradox is not literally true. Is it false?

One analogy

- "A bird in the hand is worth two in the bush."
- Is that literally true? The value of any bird in any hand is worth (= or $\geq$?) the combined value of any two birds in any bush. Ridiculous! But is the saying false?
- Maybe the true/false scale is inappropriate.
  One other scale is from "hogwash" to "brilliant."

One stronger meaning of the paradox

- Sriram Biyani and P. Santhanam
  "Exploring Defect Data
  from Development and Customer Usage
  on Software Modules over Multiple Releases"
  ISSRE 1998, International Symposium
  on Software Reliability Engineering
- "We have shown that modules with more defects in development are likely to have more defects in the field."

Deriving Myers's paradox

- The paradox, in its literal form, can actually be derived from assumptions that are strong but not completely ridiculous. We give two illustrations.

Example 1: Equally effective testing

Assume that the two program sections have been tested with the same efficiency; the fraction of bugs caught by the testing is the same, say 1/k, for both sections. Then, if b bugs are found in a section, there are kb bugs there altogether, of which

$$kb - b = b(k-1)$$

bugs remain undiscovered.

Example 2: Uniform distribution of bugs

- Modules A, B have 10,000 lines of code each. A has 1000 bugs; B has 500. In both cases the bugs are uniformly distributed.
- A perfect inspection of 1000 lines of code reveals about 100 bugs in A and about 50 bugs in B. Twice as many bugs have been found in A.
- A has twice as many remaining bugs: 900 vs. 450.

PHILOSOPHY UNDER TEST

Back to tests and proofs

- One can, in principle, verify a math proof. What about a physical theory?

Inductive reasoning

- Math employs only deductive reasoning. Natural sciences employ inductive reasoning as well.
- Examples
  - We observed sun rising on the East so many times. Hence it always rises on the East.
  - The water boils at 100 C.

Hume's problem of induction in natural sciences

- We can observe only a finite number among the infinitely many instances. How can you justify the jump to conclusion?
- Is it even useful to bother to observe more instances?
- There have been numerous attempts to justify induction e.g. probabilistically.

Popper's solution

- The problem of induction justification is so hard because such justification is impossible.
- Any scientific theory is just that, a theory.
  - Example: from Newton to Einstein
- On the other hand, scientific theories can be falsified. There is no symmetry between verification and falsification.
- Additional observations may have no additional weight.
  - The water boiling example.
- You have to challenge the theory

Falsification principle

- Falsification is the demarcation line between science
  - like physics, chemistry, etc.

  and pseudo-science
  - like astrology, Marxism, etc.
- The principle caught the imagination of scientists.

Falsification of falsification: self-application

- Self-application. Is the falsification principle scientific?

Falsification of falsification: formal logic

- Existential claims $\exists x\, \varphi(x)$ can in principle be verified provided that phi is observable.
- Universal claims $\forall x\, \varphi(x)$ can in principle be falsified provided that φ is observable.
- What is the claim is of the form $\forall x\, \exists\, y\, \varphi(x,y)$ or of the more complicated form?
- Typically scientific claims are universal
  e.g. in Newtonian mechaniics, any two objects attract each other ...
  Maybe this explains Popper's popularity with the scientists.

Pragmatics

- What is the form of a scientific claim?
- There are many explanations of a failed experiment.
  - The predicted discovery of Pluto was a triumph of Newtonian mechanics. What would happen is Pluto was not found.
- Theories are remarkably resistant to falsification especially if there is no good alternative theory.

Falsification principle and software

- Is falsifiability a demarcation line between bad and good software?
- Can you falsify software?
  - If yes, how many bugs does it take to do the trick?
- Buggy software is fine until we have a better one. Like with theories.