What might Alan Turing say about the Inevitable Fallibility of Software?

by W. Kahan, Prof. Emeritus,
Math. Dept,. and
Elect. Eng. & Computer Sci. Dept.,
University of California @ Berkeley

prepared for the ACM's Celebration in
San Francisco, 15 - 16 June 2012, of the
Centennial of Alan Turing's Birth

Now posted at www.eecs.berkeley.edu/~wkahan/15June12.pdf
More details: see www.eecs.berkeley.edu/~wkahan/Boulder.pdf

What might Alan Turing say about the Inevitable Fallibility of Software?

Abstract:

Large-scale floating-point computations in science and engineering have become nearly impossible to debug or to validate for lack of software tools and practices that could exist and would, if promulgated widely, reduce the costs of debugging and validation by orders of magnitude.

Most of roundoff's effects could be mitigated by modest retrofits to programming languages. Exceptions and unanticipated events deemed errors pose difficulties more severe; they become errors only if handled badly. Example: 2009's crash of Air France #447 in mid-Atlantic.

At times, Alan Turing seemed preoccupied with the defence of his belief that …

When computers get fast enough and have enough memory, one will be programmed to fool practically everybody into believing it is a human at the other end of the wire.

cf. IBM's *Watson*, Apple's *Siri*, … .

Aren't we humans all computers endowed with sensors and other appendages, and with firmware and software of which some is more or less buggy?

If Rodin's Thinker were a computer programmed Turing's way, what would he be thinking about?

Thoughts censured in Turing’s time are now at worst merely controversial.

“You can fool all the people some of the time, and some of the people all of the time, but …”

Will deceitful simulation of human thinking ever be deemed worth its cost?

Many of the 1950s’ most contentious questions seem much less so nowadays.

Since the 1950s and 1960s, when so many of our practices were established,

how has computing changed most?

• Speed and Storage?	Up by ~ 10^6
• Ubiquity?	Up by > 10^6
• Price?	Down by < $1/10^4$
• Diversity of Usage?	

Engineering, Science, Business, Administration, …

⇒ Embed, Entertain, Communicate, “Befriend”

Times have changed. I think the bigger change by far is the

Widening Gap

in Space-Time and Mentality

between a program’s Users and its Programmers.

That Gap ⇒ Users’ Expectations exceed Programmers’ Accomplishments

⇒ Sometimes Unreliability ⇒ Sometimes Tragedy, as we’ll see.

Comparing a computer's to a human's thinking (@ London Math. Soc. ***1947***),
Turing said …

"I would say that fair play must be given the machine.

. . .

In other words then,

if a machine is expected to be infallible,
it cannot also be intelligent."

pp. 104-5 of the vol. on *Mechanical Intelligence* in Turing's *Collected Works* (North-Holland)

Can that be right? … "expected" by whom? What about the Pope in Peter's Chair?

Were Turing to think again along such lines ***now***
he would probably reach a logically complementary inference:

To expect ambitious software to be debugged completely
is imprudent.

To expect ambitious software to be debugged completely is imprudent.

Whence come persistent bugs?

Some persist because debugging tools are inadequate. For instance, software tools are needed desperately to help users debug big floating-point computations in science and engineering. See www.eecs.berkeley.edu/~wkahan/Boulder.pdf.

In the time afforded today, I wish to consider another source of bugs:

Customary Practices that made sense in the 1960s,
but Times have Changed.

We are now so inured to the consequences of these customs that we no longer see them as customs but take them for granted instead. Here are two examples:

- FORTRANnish evaluation of floating-point expressions with operands of different precisions or types.
- Can you identify a software custom's pernicious rôle in the following story?

The Crash of *Air France*'s Flight #447 on 1 June 2009

Nowadays airliners achieve fuel economy by flying "on the razor's edge" at a high altitude, an optimized speed, and a critical angle of attack that maximizes Lift/Drag. A slightly different angle can incur an abrupt stall, so pilots must control the angle closely.

Only a computerized autopilot has enough stamina to maintain that optimal regime.

• 35000 ft. over the Atlantic about 1000 mi. NE of Rio de Janeiro, AF#447 (Airbus 330) flew through a mild thunderstorm into another so violent that its super-cooled moisture froze in and blocked all three *Pitot Probes*. They could no longer sense airspeed.

• Bereft of consistent airspeed data, the autopilot relinquished command of throttles and control surfaces to the pilots with a message of "Invalid Data" that ***did not explain why***.

• The three pilots struggled for perhaps ten seconds too long to understand why the computers had disengaged, so the aircraft stalled at too steep an angle of attack before they could institute a standard recovery procedure. (2/3 throttle, stay level, regain speed)

• Three minutes later, AF#447 pancaked into the ocean killing all 228 aboard.

Why had the autopilot's computer abandoned AF#447 so completely?

Sources about AF#447 :

<www.bea.aero/fr/enquetes/vol.a.point.enquete.af447.27mai2011.en.pdf>

NOVA6207 from PBS, rebroadcast over KQED (San Francisco) Wed. 13 June 2012.

<www.aviationweek.com/aw/jsp_includes/articlePrint.jsp?headLine=High-Altitude%20Upset%20Recovery&storyID=news/bca0711p2.xml>

Jeff Wise's article "What Really Happened Aboard Air France 447" in *Popular Mechanics*: <www.popularmechanics.com/technology/aviation/crashes/what-really-happened-aboard-air-france-447-6611877>.

Additional Details for the Story of AF#447

- AF#447 still carried too much fuel to be able to climb above the storm.
- Airbus has retrofitted stronger heaters to its Pitot probes in the hope of forestalling a reoccurrence.
- "Invalid Data" was airspeed too low to sustain flight at 35000 ft., but the pilots were not told this, so they could not know which instrument(s) to distrust.
- After AF#447 had fallen below 20000 ft., the ice melted in the Pitot probes, restoring airspeed indications, but the pilots were not told this, so they still could not know which instrument(s) to distrust. Outside, all was pitch black, depriving them of any external reference.
- Idiosyncratic Airbus controls hid from the pilots that their control inputs were at cross-purposes; still, the crash is likely to be blamed posthumously upon *Pilot Error* of the youngest pilot.

Answers to a few Anticipated Questions

Q: (p. 3) How dare you say that I am a soulless computer?

A: Our firmware is the site for our soul, our *Divine Spark*, or our *Original Sin*. Our software is influenced by our experience after birth.

Q: (p. 4) Do you assert that a computer can be programmed today to fool all of us into believing that it is a human at the other end of the telephone wire?

A: I am sure that there are now computer systems big and fast enough to deceive all of us all the time. I am not so sure that someone is willing to pay programmers enough merely to accomplish that deception. Other tasks are more urgent.

Q: (p. 6) What has the Pope to do with "… infallible … cannot also be intelligent" ?

A: Many Catholics expect the Pope's pronouncements on doctrine or morals to be infallible. The Pope is certainly intelligent despite being a machine, or part of one, expected to be infallible. He provides a counter-example to Turing's quote.

Q: (p. 7) What is wrong with "FORTRANnish evaluation of floating-point expressions" as practiced almost universally after the mid-1960s ?

A: It was chosen by writers of one-pass compilers that had to operate within small memories and produce object-code that executed as fast as possible regardless of the opinions of numerical-error-analysts. The better way to obtain computed results at least about as accurate as is deserved by uncertain data and desired results is to carry extravagant precision, at least over twice as wide as the data and desired results, in all intermediate arithmetic operations regardless of the narrower precisions of operands. Ample experience supports this better way:

- Electro-mechanical desk-top calculators did so for decades before the 1970s.
- Serendipitously, Kernighan-Ritchie ***C*** on early DEC PDP-11s before ~1983 evaluated all floating-point expressions in Double regardless of Float operands. Reverting later to FORTRANnish evaluation allowed by ANSI X3.J11 spoiled geometrical calculations in ***C***, but only a few error-analysts realized that, and they did too late.
- In the 1980s, Intel's x87 and Motorola's 68xxx numerical architectures carried 64 sig. bits when performing arithmetic on operands with 53, 24 or fewer sig. bits. They provided superior results, but too few compilers supported them properly. JAVA prohibits them.

  See my web page's …VtetLang.pdf, …Mindless.pdf, …/MathH110/Cross.pdf, …etc.

To protect us from clever programmers who use floating-point occasionally without ever having endured a competent Numerical Analysis course, programming languages should be changed to use IEEE 754's quadruple precision by default for all scratch variables.

Q: (p. 8) What software custom contributed to the crash of AF#447?

A: Recently Jeff Wise's article "What Really Happened Aboard Air France 447" appeared in *Popular Mechanics*: see <www.popularmechanics.com/technology /aviation/crashes/what-really-happened-aboard-air-france-447-6611877>. It is based upon extracts from the now recovered flight recorder. This posting on the internet is followed by a long list of commentators' nasty accusations about Air France's pilot training procedures, Airbus, and especially the younger copilot, who appears likely to have to bear all the blame posthumously for the crash. But nobody objected to an implicit (accepted without debate or explanation) convention among programming languages that obliges no programmer to consider the effect his error-message (if any) would have upon users of his program after it aborts, nor to consider the states in which the program's data structures will be left after abortion caused by an unanticipated event deemed an error. (Is it the user's error, or the programmer's?) This convention made sense in the 1960s when batch computing was universal. Now it amounts to a licence for irresponsibility among programmers who would rather not think about what happens after their program aborts.

Computing professionals should at least deprecate that convention or, better, amend compilers and programming languages to prohibit it. This will not be easy. For a very early (1966) precursor see pp. 17-25 of my web page's posting …/7094II.pdf .