A Tale of Two Cultures

Paul De Palma

(Appeared in Software Engineering Notes, 28, 5, 12/2003)

Software's unreliability is the stuff of legend. *Software Engineering Notes* has been entertaining us with tales of failure for years. As long ago as 1968, responding to reports of unusable systems, cost overruns, and outright cancellations, the NATO Science Committee convened a meeting of scientists, industry leaders, and programmers. It was at this conference that the term "software engineering" was invented in the hope that, one day, systematic, quantifiable approaches to software construction would develop. In the mean time, one of every three large-scale systems is cancelled mid-project. Of those that do make it out the door, three-quarters are operating failures: some do not work as intended, others are just shelved. Among companies surveyed by IBM few years ago, eighty-eight percent built systems that required redesign [3]. Try to imagine the same kind of gloomy numbers for civil engineering: three-fourths of bridges carry loads below specification; almost nine of ten sewage treatment plants, once completed, must be redesigned; one-third of highway projects are cancelled because technical problems have grown beyond the capacity of engineers to solve them. Civil engineering failures are rare enough to make the news.

In my university, the computer science department shares a building with the engineering school. The culture of civil engineering stares us in the face each day of our working lives. Like Margaret Mead among Samoan adolescents, I have observed a curious practice among civil engineers and their students that highlights an even more curious practice of our own. Each fall, when the computer science department fields a

team to compete in the ACM-sponsored programming contest, the young civil engineers gather together to build a concrete canoe. The cultural assumptions surrounding these two events goes a long way toward explaining why civil engineering is engineering and computing, despite forty years of investigation, is neither engineering, nor science, but a black art, mysterious, thrilling, yet a promise not kept.

Please don't misunderstand. I am not for a moment claiming that the yearly ritual of a programming contest is the cause of the software industry's dismal record. I claim just what anthropologists have always claimed, namely that patterns of cultural practice provide insight into the culture's view of itself and the world. A latter-day Levi-Strauss, I wonder what kind of discipline produces a programming contest—or a contest to build a concrete canoe.

The American Society of Civil Engineers and a Cleveland construction firm have been sponsoring a concrete canoe contest among civil engineering students since 1988. Two summers ago, the team from the University of Alabama at Huntsville won with its 34 kilogram, 6.8 meter Survivor. The contest has several parts. There is a race, of course, but this accounts for only 30 percent of the score. The race is not to see who constructs the canoe most quickly but to determine whose design is best, rather like comparing the relative efficiency of algorithms. A full 70 percent of the score comes from items that would be familiar to any engineering team: the quality of a written design, the eloquence of an oral presentation, and the clarity of a three-dimensional model. It was the written design that was most striking to this visitor from another planet. Last year's winner came with hypotheses, drag vs. velocity plots, and meticulously gathered test data. Interestingly, the design team used the boat's resonant frequency, the same phenomenon

that caused the famous collapse of the Tacoma Narrows Bridge in 1940, to flex and store energy between strokes [6]. This is engineering. A novel hypothesis that is based on well-understood principles followed by test, refinement, and product presentation.

Let's compare the concrete canoe competition with something closer to my home, the ACM-sponsored programming contest. I can do no better than let contest sponsors speak for themselves. Under the heading "Battle of the Brains" we read: "The contest pits teams of three university students against eight or more complex, real-world problems, with a grueling five-hour deadline. Huddled around a single computer, competitors race against the clock in a battle of logic, strategy and mental endurance…. The team that solves the most problems in the fewest attempts in the least cumulative time is declared a winner" [1]. What exactly are these "real-world" problems? Try *The Archeologist's Dilemma:*

> An archeologist…stumbles upon a partially destroyed wall containing strange chains of numbers. The left-hand part of these lines of digits is always intact, but unfortunately the right-hand one is often lost by erosion of the stone. However, she notices that all the numbers with all its digits intact are powers of 2, so that the hypothesis that all of them are powers of 2 is obvious. … she selects a list of numbers on which it is apparent that the number of legible digits is strictly smaller than the number of lost ones, and asks you to find the smallest power of 2 (if any) whose first digits coincide with those of the list. …you must write a program such that given an integer, it determines (if it exists) the smallest exponent E such that the first digits of 2E coincide with the integer…. [4]

Now, I like this kind of problem as much as the next geek. But are those who can solve it quickly and under pressure "the best and brightest students from around the world," as the sponsors claim? And does its answer constitute a "software system" under any meaningful definition of the term? I might even quibble about whether the Archeologist's Dilemma comes to us from the "real-world." But that's not the important issue. No one is going to build a concrete canoe in that fabled place, the real world, either. One might argue that The Archeologist's Dilemma does for

young programmers what the canoe does for young engineers. It asks them to use the tools and materials of their trade to solve a clever problem. Presumably, if they can solve this problem, once consigned to the real world, they will be able to solve real "real-world" problems, though not in five hours, a caveat that seems lost on the contest's sponsors.

No, the cultural assumptions to be teased out are not to be found in the problem statement, but rather in the terms of the contest itself. Let me count the ways:

- **The Tyranny of the Examination**

We in computer science owe a great deal to mathematicians. Turing, Von Neuman, Shannon, Minsky, Knuth, McCarthy. They are our patron saints and elder statesmen. What we should shed, however, is the rather peculiar notion—embedded deeply in mathematics education—that the ability to solve trivial problems quickly, scales up to the ability to solve genuinely complex problems. Gather two or three mathematicians together, and you'll find a test somewhere among them. Maybe training by timed examination works fine for young mathematicians, though I have my doubts. It is not the way to train engineers, however, not if you want them to build structures that stand up under stress. "Grueling five-hour deadline," "race against the clock." What matters here is the program. Neither its design, nor its documentation, nor its reliability, nor its maintainability, not even its user interface, is mentioned. These are left to chance in the programming contest's "Battle of the Brains."

- **The Cult of the Genius**

The marble slab inside Santa Maria del Fiore in Florence commemorating the man who designed its dome, the largest since the church of Santa Sophia was built in Constantinople 900 years earlier, reads:

> Corpus Magni Ingenii Viri Philippi Brunelleschi Fiorentini
> ("Body of the great and clever man Filippo Brunelleschi of Florence")

The words *genius* and *ingenious* come from medieval Latin words used to describe the construction of machines. An *ingenium* was a machine. An *ingeniator* was a man who built one [5]. Since the Italian word for *engineer* is *ingenere,* the etymology is clear. An engineer was once an *ingeniator,* one who used his genius to build clever structures. We know mostly about the medieval buildings that still stand, however. Sadly, the products of genius, then, as now, collapsed regularly. Lucky for us, an engineer is no longer a genius, but rather someone who uses well-understood materials to build structures with known tolerances and predictable behaviors. What is lost in romance is gained in reliability. In calling the ACM's programming contest "The Battle of the Brains," its sponsors implicitly acknowledge that we in computer science are still casting about, like the city fathers in 15th century Florence, for a few clever men (and women, of course). The acknowledgement indicates just how far we have to go to become a true profession.

- **The Revenge of the Nerds**

More than a quarter century ago, Joseph Weizenbaum described young men obsessed with computers like this: "…bright young men of disheveled appearance, often with sunken glowing eyes, can be seen sitting at computer consoles, their arms tensed and waiting to fire their fingers, already poised to strike, at the buttons and keys on which their attention seems

to be as riveted as a gambler's on the rolling dice" [8]. As the French say, *la plus ca change*…. Surely I am not the only one who thinks that this kind obsessive behavior is the very thing rewarded in the ACM programming contest: "competitors race against the clock in a battle of … mental endurance." This is sport, not engineering, a useful device to siphon off excess testosterone, but hardly a way "to assist in the development of top students…." In fact, anyone who has ever spent time taming the youthful, often boundlessly arrogant, energy of young programmers will begin to understand just why our field attracts so few young women.

- **The Final Irony**

The ACM Advisory Panel on Professional Licensing has advised against licensing in our field because no one has designed an examination that will ensure public safety [2]. Presumably, this includes The ACM International Collegiate Programming Contest as well. Since we don't know how to make matters better through examination, at the very least, let's not make them worse. Let's not encourage the very tendencies—speed at the cost of design, an aversion to documentation, maintenance left to chance—that we ought to be wringing out of our young charges at every opportunity.

In the early seventies, Gerald Weinberg, wrote an important and insightful book on the practice of programming. There he counseled managers to fire immediately any programmer who had become indispensable to their operations [7]. I offer preemptive advice to the same managers. Beware of job applicants excessively proud of their participation in a programming contest. Confusing cleverness with care, they might one day become indispensable programmers. To faculty who have been asked to lead a

programming team, I offer the words of the ACM Advisory Panel on Professional Licensing: “Not Now, Not Like This.”

References

1. ACM International Collegiate Programming Contest. Retrieved June 2, 2002 from: http://icpc.baylor.edu/past/default.htm.

2. Allen, Fran, et. al. Not Now, Not Like This. *CACM* 43, 2 (1998), 29-30.

3. Gibbs, W. Wayt. Software's Chronic Crisis. *Scientific American.* (September, 1994), 166-174.

4. ICPC Valladolid Problem Archive. Retrieved June 2, 2002 from: http://acm.uva.es/problemset.

5. King, Ross. *Brunelleschi's Dome*. NY, Penguin Books, 2000.

6. Team UAH - ASCE Concrete Canoe Team at the University of Alabama, Huntsville. Retrieved June 2, 2002 from: *http://www.uah.edu/student_life/organizations/ASCE/Competition/2001.htm*

7. Weinberg, Gerald. *The Psychology of Computer Programming*. NY, Dorset, 1998.

8. Weizenbaum, Joseph. *Computer Power and Human Reason*. San Francisco, W.H. Freeman, 1978.