got quality?

A number of readers have asked for more information on the Code Quality Summit I attended in January that I mentioned in the Editor's Forum for the March 2003 issue. Now that it's over, I can spill more beans about it.

Its inception consisted of conversations between Scott Meyers and Bruce Eckel last summer. Shortly thereafter Bill Venners and I were asked to help plan the event. The other attendees were Joshua Bloch, Alistair Cockburn, Matt Gerrans, Kevlin Henney, Andrew Hunt, Angelika Langer, Pete McBreen, Dawn McGee, Chris Sells, Randy Stafford, and Dave Thomas. We spent three days in Portland discussing the state of software quality and what, if anything, can be done about it. I was more than a little dismayed to hear Kevlin Henney share a quote estimating the annual cost of software bugs in the United States at $60 billion[1]. How did things get so bad? I don't claim to know the complete answer, but I have an idea or two.

While developers have certainly committed their share of errors, I have long observed that a hefty portion of that $60 billion is due to a lack of faith on the part of management. The efficacy of refactoring in reducing software entropy is now well established, for example, but few managers allow room for it in project schedules. Timely training and mentoring are invaluable, but developers lacking up-to-date skills seem to be the norm (at least by my informal polling). It seems that managers who care to do their part to raise the competence level of their employees are the exception.

A surprising culprit is technology itself. In our race to automate development, our tools have deceptively made programming seem easier than it really is (or in fact can be). This is not a new phenomenon, of course. COBOL was supposed to wrench the "black art" of software development from mathematicians and engineers by making it appear as easy as writing an essay. It succeeded in automating many bookkeeping tasks, all right, but there is more to our discipline than storing business records and writing reports. IDEs such as Visual Basic (the COBOL of the 90s and beyond) have engaged multitudes that perceive software engineering as visual Mouse Engineering ("I click, therefore I program")[2], and their numbers are an order of magnitude greater than the mentors that can help them master the timeless programming arts of abstraction and good design that have always been and will remain difficult (and well they should)[3]. At the summit Alistair Cockburn quoted research showing that we're losing this "battle of the exponents." A good

---

[1] CACM, Jan. 2003, p. 136. The 309-page report can be viewed at http://www.nist.gov/director/prog-ofc/report02-3.pdf.

[2] For an excellent wake-up call on the gotchas of visual, event-driven programming, see Miro Samek's article, "Who Moved My State" in last month's CUJ.

[3] Two timeless quotes from the late Dijkstra's *The Humble Programmer* are in order:

- "The effective exploitation of his powers of abstraction must be regarded as one of the most vital activities of a competent programmer."
- "Programming will remain very difficult, because once we have freed ourselves from the circumstantial cumbersomeness, we will find ourselves free to tackle the problems that are now well beyond our programming capacity."

programmer is at once an abstractionist, a designer, an engineer, an artist, and a craftsperson. In the right hands visual tools are a godsend, but maybe we've made it too easy to take on the appearance of programming without the substance thereof.

We are also raising the bar here at CUJ. If you peruse the author guidelines on our website, you will notice we insist that you to adhere to language standards and accepted industry "best practices" when crafting well-behaved code to accompany your articles. We also expect all articles to be reviewed by at least two skilled, technical peers before you even submit them. For your reference, we've updated our Reading List to include those titles that teach quality principles.

During my twenty-five years in this business, I have repeatedly seen efforts to attain excellence pay off. I have witnessed the flexibility and ease of maintenance brought on by incremental, modular design coupled with continuous refactoring. I have seen skill levels raised through judicious training, mentoring, and code reviews. I have observed risk mitigated and morale boosted. It has demanded trust, teamwork, and careful thought. Quality is worth the effort.