Certitude and Rectitude

Peter G. Neumann
Principal Scientist, Computer Science Lab
SRI International, Menlo Park CA 94025-3493
Neumann@CSL.sri.com, http://www.csl.sri.com
1-650-859-2375

Cer.ti.tude: the state of being or feeling certain;
Rec.ti.tude: correctness of judgment or procedure [1]

There is a fundamental difference between certification (which is intended to give you the *feeling* that someone or something is doing the right thing) and correctness (for which you hopefully have some *well-founded reason to believe* that someone or something is doing the right thing – with respect to appropriate definitions of what is right). Certification is typically nowhere near enough; correctness is somewhat closer to what is needed, although often unattainable in the large – that is, with respect to the entire system. However, formal demonstrations that something is correct are potentially much more valuable than loosely based certification. So, a challenge confronting us here is to endow certification – of people and of systems – with a greater sense of rigor and credibility.

I have long worked on systems with critical requirements for security, reliability, and survivability in the face of all sorts of adversities. Numerous system failures [2] demonstrate the vital importance of people. Many cases are clearly attributable to human shortsightedness, incompetence, ignorance, carelessness, or other foibles. Ironically, accidents resulting from badly designed human interfaces are often blamed on operators (e.g., pilots) rather than developers. Unfortunately, *software engineering* as practiced in much of the world is merely a buzzword rather than an engineering profession [3,4]. This is particularly painful with respect to systems with life-critical, mission-critical, or otherwise stringent requirements. Consequently, various alternatives deserve extensive exploration:

- Good development practice, enforceable requirements, and sensible system architectures are all valuable. Their principled use should earn insurance discounts, contractual bonuses, and perhaps even legal relief from certain aspects of liability. Bad development practice (including low bidders taking unwise shortcuts and risks) must not be condoned.
- Critical systems should be developed by persons and companies with adequate training and education.

The recent Y2K fiasco [5] should remind us of how difficult it is to develop dependable systems (even with competent people), and how much better it is to have systems well designed and well engineered from the outset. Much better education and training must become an intrinsic part of the mainstream [6], especially addressing the foundations for developing dependable systems. Overall, we urgently need to explore alternatives within the context of the entire process of development, maintenance, and continued evolution. Although lowest-common-denominator certification of conventional programmers and simplistic metrics for judging organizational competence are likely to be palliatives at best, sensible procedures for certifying requirements engineers, system engineers, software engineers, debuggers, etc., could be just one of many potentially useful steps toward instilling greater discipline into the development process – particularly for critical systems.

1. Abstracted from Webster's International Dictionary.
2. Peter G. Neumann, Computer-Related Risks, Addison-Wesley, 1995, with considerable subsequent material on-line. http://www.csl.sri.com/neumann
3. David L. Parnas, Software Engineering: An Unconsummated Marriage, *CACM 40*, 9, September 1997.
4. Peter J. Denning, Computer Science and Software Engineering: Filing for Divorce?, *CACM 41*, 8, August 1998.
5. Peter G. Neumann, A Tale of Two Thousands, *CACM 43*, 3, March 2000.
6. PGN, notes for a course on developing highly survivable systems and networks, University of Maryland, Fall 1999. http://www.csl.sri.com/neumann/umd.html