ACM Fellow Profile

L. Peter Deutsch

Please elaborate on the work leading up to your achieving the distinction of ACM Fellow:
The greatest part of my career has been concerned with improving the ease of using computers and the ease of developing quality software. This desire led me into a variety of activities related to programming languages, interpreters, compilers, and tools.

What is the best reference to your work?
There is no single good reference, since most of my work took the form of working systems rather than publications. I was a substantial contributor to Interlisp and to ParcPlace (now ObjectShare) Smalltalk (now VisualWorks). My two best published papers were my Ph.D. thesis, published by Xerox PARC in 1971, and the paper I co-authored in the 1984 ACM POPL.

What are your current research interests?
I have been doing engineering in industry since 1991 and wouldn't say I have research interests per se at the moment. However, I continue to be deeply interested in tools and languages that support the productive creation of high-quality software, and specifically in the area of machine-checkable, formally stated properties of programs.

What are your current outside interests?
Assuming you mean outside software engineering, I enjoy traveling, singing, square dancing, writing music, and cheap ethnic restaurants.

What was the greatest influence on you?
Lisp, which made me aware that software could be close to executable mathematics; Smalltalk, which made me aware of the synergy between language, tools, and libraries in software productivity; and the PDP-1 time-sharing project at MIT in the early 1960s, which introduced me to interactive computing. In terms of people, I would say Danny Bobrow, for being a mentor and collaborator when I was in high school and college; Robert Floyd, for the 1971 IFIP paper that motivated my Ph.D. work; and Edsger Dijkstra, for insisting that software is mathematics.

What was your greatest influence?
I think I have had two significant influences on the world of software. The 1984 POPL paper is the most widely cited reference for what is now called JIT compilation. The other influence has been through my Ghostscript software, which both has been of tremendous practical benefit and also has an innovative licensing approach. Beyond that, I have no idea to what extent the various tools and implementation techniques I contributed to the Interlisp and Smalltalk systems have affected the industry.

Who do you think has made the greatest impact on software engineering?
I think the greatest impact on software engineering has come from the development of better languages. I would hesitate to choose between John Backus, for FORTRAN, and Thompson and Ritchie, for C.

Which computer-related areas are most in need of investment by government, business or education?
The government desperately needs to be better educated so that it can start to undo the deleterious effects of software patents (by reversing the error of interpretation by the courts that allowed them in the first place) and of Microsoft's desktop hegemony (by understanding that seamlessly interoperable software does not require single-sourcing the entire collection, and that interoperability without a requirement of timely, free, and open documentation of interfaces makes creative competition impossible). Beyond that, see my advice below.

What advice do you have for computer science/software engineering students?

1. Good software requires the ability to think formally (mathematically), even more than other kinds of engineering. Make sure you have some exposure to assertions, proofs, and analysis of algorithms, and that you use them enough to understand what they are good for.
2. On the other hand, so-called "formal methods" of software development processes are vastly overrated: don't hesitate to challenge them if they get in your way.
3. Documentation and readability are as important to software quality in the long run as speed of creation, correct functioning, and performance are in the short run: make sure you have the time to do a good job on them.

What is the most often-overlooked risk in software engineering?
That as a system grows over time, it will become too complex or disjointed to understand or make work reliably.

What is the most-repeated mistake in software engineering?
Underestimating the time or effort required to complete a project.

Do you have any other comments on software engineering?
"Software engineering" is something of an oxymoron. It's very difficult to have real engineering before you have physics, and there isn't anything even close to a physics for software – an account of elementary objects in terms that can be used to usefully describe and understand the composites that are built out of them, including the emergent properties of those composites. If the individual statements of programming languages are the elementary particles, standard libraries are the building materials: we have the standard C libraries (which are at about the conceptual level of things like clay, rock, and wood), a rich but proprietary (single-source, hidden-construction) array of single-source libraries from Microsoft, and the emerging Java standard libraries, none of which have the formal specifications that are needed to make them good building materials.

Thank you!

– profiled by Ron Finkbine, Ph.D. [finkbine@hanover.edu]