Features Editor: Dale Strok ■ dstrok@computer.org

Celebrating *Peopleware*'s 20th Anniversary

Ed Yourdon

At the 2007 International Conference on Software Engineering in Minneapolis, I had the honor of participating in a panel session celebrating the 20th anniversary of the publication of *Peopleware* with five of the software field's luminaries: Tom DeMarco and Tim Lister (the book's authors), and Barry Boehm, Fred Brooks, and Linda Rising. The panel was conceived, organized, and moderated by Steve Fraser, who deserves great credit for putting it all together and keeping the panel from degenerating into pandemonium.

The panel members

Barry Boehm is the "father" of software engineering economics. His 1981 opus, *Software Engineering Economics*, should be read along with his newer, vintage-2000 book, *Software Cost Estimation with* COCOMO*-II*. Barry originated the COCOMO software cost-estimating model, the spiral model of software development, and several other key ideas in software engineering. In addition to ongoing research in these areas, he teaches SE and computer science at the University of Southern California. During the panel discussion, he noted that undergraduate CS students are often taught that people are abstractions (for example, they're taught to create system models and diagrams that represent users with stick figures labeled U_1, U_2, …, U_n). They're also taught that project staffing is a "packaging" problem—if you need to accomplish X person-hours of work in Y calendar months, then you need Z people (often known as "resources," another abstraction) to get the work done. So, Barry said, it's a shock for such students to read *Peopleware* and to be told that concepts such as "jelling" and "teamicide" are realities.

Fred Brooks is known for his work on the first big mainframe operating system, IBM's OS/360. He's even better known for his landmark book *The Mythical Man-Month* and for numerous technical papers such as "No Silver Bullet," published in *Computer* (Apr. 1987). He told the audience that it's been 20 years since he has taught, or focused seriously, on SE; most of his work at the University of North Carolina is in virtual reality. But he still insists that all his students read *Peopleware*, and he predicts that the book will survive a long, long time. Why? For the same reason, he says, that the stories of Homer have survived for thousands of years: they're stories about people, and those stories are just as true today as they were a thousand years ago. Brooks emphasizes four key points from *Peopleware* to his students:

- The importance of team jelling and teamicide—a concept with which many CS students, having worked mostly on individual projects during their education, are entirely unfamiliar.
- The importance of "space"—that is, giving software engineers a decent working environment, rather than a cramped cubicle with Muzak blaring from the ceiling.
- The emphasis on "people quality"—some people can write programs that are 10 times

0740-7459/07/$25.00 © 2007 IEEE

faster and smaller, and they can do so 10 to 20 times faster. (For the first significant documented evidence of these differences, see "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," by Harold Sackman, Warren Erikson, and E. Eugene Grant, in the January 1968 *Communications of the ACM*.) So, companies should try to hire such people and should recruit, nurture, reward, and protect them.
- The (negative) impact of moving a large software project, en masse, from one geographical location to another. A very few such projects survive a move, Brooks said, but only by starting over.

Perhaps I was on the panel because of the rumor that Tom and Tim were initially going to title *Peopleware* "All the Things Ed Yourdon Screwed Up When He Was Our Manager." But I think my sins (at least in that area) have been either forgiven or forgotten, and I did my best to avoid causing too much trouble on the panel. I told the audience that I had begun working in the software field—and wrote my first few technical books—during a period of youthful naiveté when I thought that software development was a technical task, to be performed in a rational manner by mature adults. I gradually learned otherwise, although it was quite a shock to read Gerald Weinberg's *The Psychology of Computer Programming* in 1971 and learn that software was at least partly a touchy-feely activity carried out by "people." Sixteen years later, a new generation of software developers was equally shocked by the similar message in *Peopleware*, and I suggested to the audience that some of them listening today, in 2007, might be equally shocked by what they were hearing in the panel session.

Linda Rising is an expert in object-design metrics and has done a great deal of work in introducing software patterns and practices into organizations, including coauthoring *Fearless Change* with Mary Lynn Manns. She referred to Christopher Alexander's 1977 book *A Pattern Language* and his 1979 book *The Timeless Way of Building*, and asked how many in the audience had heard of him. Surprisingly (to me, at least), roughly 75 percent raised their hands. She suggested that many of us in the software field had borrowed Alexander's ideas about patterns without realizing it. She might or might not have been aware that my colleague Larry Constantine and I had borrowed an even earlier collection of Alexander's ideas from his 1964 book *Notes on the Synthesis of Form* as the basis for the structured-design concepts of coupling and cohesion. Rising said she was interested in people-related patterns, too. She said that one of the things she likes best from *Peopleware* is the story of Holger Danske, the legendary sleeping giant who will awaken if Denmark is in danger. DeMarco and Lister compare the staff of a large software company to the giant.

Tim Lister has been Tom's professional colleague and book-writing partner since 1976. He and Tom have written several books together, the most recent being *Waltzing with Bears: Managing Risks on Software Projects*. Tim suggested that 20 years might be a little too early to have a retrospective on *Peopleware*, but went on to tell us how he and Tom conceived of the book and collected the material for its contents. What was originally just a few slides for the just-before-lunch session of a seminar they were teaching unleashed a torrent of stories from real-world software managers about the good, the bad, and the ugly peopleware-related experiences in their projects. Looking forward, Tim encouraged the agile development community to continue exploring new ideas, and he encouraged all of us to read *Artful Making: What Managers Need to Know about How Artists Work*, by Rob Austin and Lee Devin.

Tom DeMarco initially gained fame by writing one of the first, and by far the most readable, textbooks on structured analysis: *Structured Systems Analysis and Specification*. He has since written several other books, including *Peopleware*. Tom emphasized that *Peopleware* had been written as a team effort with Tim. Emphasizing something Tim had said earlier, Tom told the audience that owning half of something wonderful was far better than owning all of something that was merely "okay." And while this half-ownership was something he obviously felt strongly about, in terms of his own experience, he suggested that it was really a metaphor for something all of us should strive for in our work. There's a "multiplier" effect that you can achieve from the work that you do as part of a team, especially with people you like and respect. But he said this phenomenon was unpredictable and referred obliquely to a coauthoring project in which the two authors never spoke to one another after they had finished the book.

Repeating some of Tim's themes, Tom said that *Peopleware* had let them become a "clear-

There's a "multiplier" effect that you can achieve from the work that you do as part of a team, especially with people you like and respect.

inghouse" for ideas about better ways of dealing with people in the IT profession. But he said that in some ways it was a failure—especially in persuading IT managers to provide better working conditions for their programmers and software engineers. He was pleased that phrases from the book such as "furniture police" have entered the common lexicon. However, all the book's rational, quantitative arguments (including results from a massive 600-person coding "war game") showing a positive correlation between decent office space and dramatically improved productivity and quality have had little or no effect on managers. They still try to squeeze the maximum number of people into the minimum number of cubic feet of office space.

The discussion

Michael, an audience member from Switzerland, told us that he had very much enjoyed *Peopleware*, but he wondered why agile development methodologies had taken so long to become known and accepted.

- Tom quipped, "It's all Barry's fault!" He suggested that we had all been brainwashed by Barry's argument, first published in his *Software Engineering Economics*, that the cost of repairing defects rises exponentially the later they're found in the software life cycle. (For a more recent exposition of this point, see "The Software Quality Lifecycle," by Yochi Slonim, in the 19 December 2005 *Dr. Dobb's*.) As a result, the commandment "Get the requirements right!" was drummed into the heads of a generation of software engineers. Tom turned toward Barry, smiled, wagged his finger, and said, "And I have never forgiven you!"
- Barry explained that, back in the 1970s, he had linked up with Win Royce at TRW, where the two of them found that the waterfall methodology worked pretty well. But he acknowledged that they were working in a time and an application domain (aerospace and military systems) in which the end user's requirements were fairly well defined. Consequently, it made a great deal of sense to capture those requirements early, rather than discover later on that a great deal of software had been built to implement the wrong requirements. Barry acknowledged that by the 1980s, things had begun to change drastically, and obviously this continues to be true.
- I answered Michael's question with a broader question of my own: Why has our field taken so long to assimilate and accept any of the SE ideas that we all agree are useful, important, and generally successful? I suggested that if we were to conduct an informal poll about not only agile methodologies but also code inspections, identification of "error prone" modules, and so on, we would probably find that only 10 percent of the audience was actually using them.
- Linda suggested that the software industry grew to its present (enormous) size before it was ready—so we've always been searching for a model to emulate, whether it's architecture or other engineering disciplines.

Larry from the Stevens Institute of Technology suggested that the main driver in our industry is fun. How can we organize our work—which consists of long hours of monotony, separated by moments of ecstasy—and ensure that our employers still make a profit?

Why has our field taken so long to assimilate and accept any of the SE ideas that we all agree are useful, important, and generally successful?

- Linda responded by telling us that she had recently given a talk on sex among primates and how it all related to agile software development. If we can presume that sex is fun (at least for primates), maybe she's got one answer to Larry's question.
- Tom suggested that "fun" equates to "play," and said he had been influenced by Alan Kay's distinction between "hard play" and "soft play." Soft play, he suggested, is like watching *American Idol*, while hard play is like learning to play the piano.
- Fred said that OS/360 was a once-in-a-lifetime experience, that he and his team felt they really could change the world—much like the comments we used to hear from the original Macintosh team at Apple back in the mid-1980s. Part of the fun, he said, is being on a winning team.
- Barry suggested that we tend to overemphasize the contractual nature of many software development projects—especially when the user-developer relationship in an in-house project gets transformed into a more formal vendor-customer relationship for an externally developed system. We need to emphasize helping clients (or users) to win, too, he said, and look for win-win situations.
- I reminded the audience of the phenomenon we see in the open source area: people often work at a "day job" that they hate, side-by-side with coworkers they despise, and taking orders from a manager they loathe. But then they leave their day job, march into their office at home (which is often equipped with more up-to-date computer facilities than what their employer gives them), and start having fun on an open source project they love, with coworkers (located all over the world) they respect. So the business of having fun doesn't have to be an all-or-nothing proposition; we all have to find a way to pay the rent and put food on the table, but it doesn't have to occupy us 16 hours a day.
- Fred suggested that only a small fraction of people on this planet have the luxury of working on some-

thing they consider fun. The fact that many of us in the software field can do so means that we're blessed.

Mary Poppendieck offered a comment rather than a question. She gave Fred Brooks a hard time by telling him that she had heard of his book in 1975 but didn't like the term "man-month."

- Fred said that he was sorry if the title offended her and that even liberal people in the mid-70s were using that phrase. And besides, he said, the title was alliterative: "Mythical Person-Month" doesn't roll off your tongue so easily.

An audience member named Earl asked how he and his colleagues could take the panel's experience and transfer it to his CS students.

- Linda remarked that our whole educational model is flawed. It reminds her of the Monty Python Theory of Education, which involves slicing open the presenter's head, scooping out knowledge, slicing open the students' heads, and distributing the knowledge around in some fashion. We should be moving toward an apprentice/mentor model, she said, much like we see in fields such as architecture. Students should see software "masterpieces" from which they can learn.
- Tim suggested that Earl was taking on too much responsibility for transferring the panel's experience to his students. Software, he said, is like paint: it's a medium that you use differently depending on whether you plan to paint a wall or a Rembrandt. He argued vociferously that the really big failure isn't in the universities but in IT organizations. Companies today invest zero in training, which is quite different from the situation he recalls when he first got into the field in the mid-1970s.
- Tom pointed out that we might be able to use middle schools as a guide: because of understaffing and overcrowded classrooms, industry people are working with teachers as partners. And the teachers are putting kids into teams so that they can help each other. The teachers will tinker with the teams to increase the chances of jelling, and will tell them that everyone in the team gets the same grade.
- Barry said that his university is working on this problem at the master's-degree level and is trying to figure out how to take it down to the freshman level in undergraduate curricula.
- Fred said, "People learn most concepts by induction from examples. Then, we're so pleased at having discovered the generalization that we all teach by deduction from the generalization (just as I'm doing with this very statement). Thus we create an impedance mismatch between the learner and the teacher." By contrast, what makes *Peopleware* popular is that it's readable. And the reason it's so readable is that it tells stories—vignettes such as the tale of the furniture police. We need to do more of this in universities, and pass on the wisdom of the panelists and *Peopleware* by telling more stories in our CS classes.

Steve Easterbrook from the University of Toronto asked, "Why is there so little research on peopleware-related topics in academic circles?" Someone in the audience immediately yelled "Tenure!" which drew some chuckles and laughter from everyone else.

Software is like paint: it's a medium that you use differently depending on whether you plan to paint a wall or a Rembrandt.

- Linda said that when she decided to go back to graduate school, she had trouble finding anyone on the faculty who was interested in such topics. The implication, of course, is that this phenomenon becomes self-perpetuating.
- Tom told us that he had submitted several papers to previous ICSE conferences on peopleware-related topics—which he described as "squishy." Unfortunately, most of those papers were rejected, except for a few that were finally accepted as "experience papers."

Kevin Sullivan from the University of Virginia asked why we're having such trouble attracting people in the CS field, considering that many studies indicate that software offers the best jobs in the best geographical locations.

- Tom responded that when software was first identified as an "industry," it had zero revenues; by 1985, as he recalled, its annual revenues were approximately $31 billion. Most of that money was spent on salaries, and many of the people who worked in the field were women, because it paid much better than most of the other jobs available to them. But about five years ago, he said, women started moving out of CS and SE to medicine, law, and other professions. Tom suggests that this is because the workplace is now so unfriendly and uncomfortable—which includes, he says, the all-too-common experience of having to sit through one boring meeting after another, rather than doing interesting work.
- Fred agreed with the part about meetings: back in the 1960s, he said, meetings were short.
- Barry pointed out that one reason for the difficulty of attracting people into the computer field in Europe is that most of the large hardware companies there have gone out of business.
- I suggested that another reason for the phenomenon is that high school graduates—at least in the US—have been hearing about offshore outsourcing and are concerned that all the high-

paying software jobs are moving to India. So, they're worried that if they major in CS or SE, they won't be able to get a job when they graduate. Whether or not this is actually true, it's the perception that influences a university student's choice of a major.

David Jansen from Cal Poly State University, San Luis Obispo, California, told us that his students enjoy reading *Peopleware* and that they use some of the material they've learned—stories about the furniture police—when going through interviews with prospective employers.

Rob Deline from Microsoft told us that his company has been researching peopleware-related issues. He noted that several people on the panel had written the textbooks for which they were best known while they were working in industry, after which they moved on to academia. Why, he asked, did this happen?

- Barry told us that his university does have an industry affiliate program … which implied that he interpreted Rob's question slightly differently than I had: do industry people maintain any kind of relationship with academia?
- Linda said that she gives talks at universities almost whenever asked to do so—and suggested that universities should be doing more of this (not just by inviting her more often, but by inviting all kinds of computer people from industry).

Tom from an English university suggested that we have a crisis related to getting more people interested in studying CS but that it starts at a much younger age—students as young as 11 to 14 are getting turned off to math, science, and computing. How can we change this?

- Tom suggested that we need to provide more educational materials to encourage hard play in the curriculum, in the sense that he described earlier. I think another good example of this kind of hard play is the Logo programming language that Seymour Papert and his colleagues developed years ago. Tom also noted that Roger Pressman, whose SE book is probably the most widely used text of its kind in universities, said that teachers need more support. I guess he was implying that it's hard to prevent kids from getting turned off if you're facing an overcrowded classroom and can't give them individual attention.
- Barry suggested that both universities and industry could help the situation by sponsoring more career days, where parents come into the classroom to explain what they do. This, of course, has been going for years in other fields, with parents telling the kids what it's like to be a soldier, a fireman, a policeman, or a doctor.
- Fred noted that his university also gives laboratory presentations on virtual reality to middle-school children. He suggested that we need to think more about using simulations as a teaching mechanism.
- I suggested that all of this might be moot because of offshore outsourcing. Several CEOs of high-tech American firms have been heard to say that their firms will continue to prosper for the foreseeable future, even if they never hire another American again—because there's an ample supply of well-educated, lower-paid, hard-working graduates from China, India, and other parts of the world.

Both universities and industry could help the situation by sponsoring more career days, where parents come into the classroom to explain what they do.

With our scheduled time drawing to a close, Steve Fraser asked the panelists to summarize their thoughts and positions:

- Barry told us that personnel is one of the top 10 risks in software projects, so we should keep it in mind. And companies should devote more attention to retaining the good people that they have.
- Tom repeated the primary theme from *Peopleware*: The major problems of our industry are sociological, not technological.
- Fred told us to remember one word: people. It's easy, he said, for us university people to forget that it's people, not papers, that count. (That was a take-off on GE's slogan that "Progress is our most important product.")
- Tim said that if we want to have fun, we should push decisions down in the hierarchy. And we should remember that the prime assets in our software organizations are people in the 25–30 age range; we should leave them alone and buffer them from corporate bureaucracy.
- Linda suggested that all of this seems so obvious; why do we need *Peopleware*? She said that her studies of cognitive psychology and primate sex taught her that, under stress, people fall apart and forget essential things that they would otherwise practice quite competently. Patterns help us crawl out of a bad situation, and the stories in *Peopleware* have become patterns.
- I suggested that both the panelists and audience had offered a lot of good ideas and that we should capture those ideas and distribute them more widely. Thus, you have this panel report.

With that, the panel session came to a close, and we all went our merry way.

Ed Yourdon is a software consultant in his own firm, Nodruoy, as well as a cofounder and senior consultant of the Cutter Consortium. Contact him at yourdon@mac.com; www.yourdon.com.