SOFTWARE PIRACY
As a long-term software professional, I enjoyed David Alan Grier’s column on software piracy in Computer’s March 2008 issue (The Known World: “Long John Software and the Digital Jolly Roger,” pp. 7-9). I think, though, that drawing an analogy to piracy on the high seas is misleading. Piracy of physical goods deprives others of the opportunity to use those goods, while software piracy is more like other white-collar crimes—tax evasion, political corruption, and so on—in that no legitimate users suffer a loss because someone uses a pirated copy.

I believe that piracy actually and paradoxically serves the interests of many software companies whose products are being pirated. As Grier points out, many users of pirated software do not have the resources to purchase a legitimate copy. Without the pirated software, these users would constitute a large market for any software vendor who would undercut the market leader in price.

When these users buy the pirated copy, they boost the market leader’s share and actually help maintain the market leader’s legitimate business. This is particularly true for software that uses proprietary file formats, as do most leading word processing, graphics, and computational software programs. These packages have become de facto standards because everyone needs them to access those formats, and the pirated copies allow many users access. Requiring data exchange to take place in open standard formats would reduce the lock the market leaders have on users.

Warren Montgomery
wamontgomery@att.net

SOFTWARE SOLUTIONS
I teach a “capstone” course titled “Strategy and Policy for Information Technology” that is aimed squarely at equipping government and military leaders to make a difference in the “software century” as Barry Boehm called it (Mar. 2008, “Making a Difference in the Software Century,” pp. 32-38).

Boehm’s concerns seem valid and warrant our attention. I would like to suggest, however, that he hasn’t embraced the truly disruptive nature of the problem or the required solution.

The problem we must address is that the nature of the “value” that any prospective software product can deliver is changing at a rapid rate. Because available technology, user contexts, and social networks exhibit what Kurzweil called “accelerating returns,” the software developer is trying to close on a target that is moving faster and faster away from what stakeholders think they will need at any fixed time in the future.

Closing on targets that are increasing their velocity and acceleration requires at least that we continually shorten the development and delivery cycle. So one characteristic needed in our solution approach is continually accelerated iterations. While shorter cycles will reduce opportunity costs and total failures, they can’t possibly deliver software that meets all objectives.

A second element is required to enhance our success: We must utilize automation to generate software products faster. Many techniques are relevant and have promise, and we should increasingly shift resources to put them to work. Once we can generate candidate software products rapidly, as a third innovation we will want to test and winnow them rapidly, because we can often recognize value more easily than we can specify it or design it.

At this point, the three elements of the required solution will constitute a recognizable pattern to meet the challenges of assembling viable products that live in and contribute to ecosystems: We will have implemented adaptive, evolutionary solutions using natural selection.

While that solution poses implementation challenges for us, it at least represents a plausible approach to fundamental characteristics of the problem we face. Without such an approach, slow processes and severely limited populations of system candidates will inevitably lead us to fall further behind our aspirations and potential.

Rick Hayes-Roth
fahayesr@nps.navy.mil

Barry Boehm responds:
I think Rick Hayes-Roth is right on in his identification of the need for new approaches to deal with increasingly rapid change and in his suggestions for addressing this challenge via shorter iterations, rapid software generation, and rapid testing and winnowing of alternate solutions.

One trend that is likely to help us in the future is multicore chip technology. For example, while one set of CPU cores is executing the current program, others can be monitoring change trends, exploring alternative methods for capitalizing on the trends, and testing them on alternative change scenarios to help users determine when to change to a new approach.

This is another example of the exciting challenges and solution media that future software engineers will be able to use in determining how best to make a difference.

boehm@csse.usc.edu

NETWORK TECHNOLOGY
In “Thinking Locally, Acting Globally” (The Known World, Apr. 2008, pp. 7-10), David Alan Grier wrote, “The benefits of community networks might be obvious, but so are the problems of building a robust wireless network for a modern, urban environment.” Apart from the commercial reasons that deter their development, some innovative technologies for implementing community networks clearly are necessary.

The tremendous development of Internet infrastructures and communication technologies has led to a dramatic increase in network management complexity which thereby compounds many reliability problems.
Developers have designed, implemented, and centrally coordinated traditional networks. They use a homogeneous population of components with common technical standards and management goals to architect these networks. However, the next-generation networks are expected to grow more chaotically with no centrally mandated goals. Autonomic management inspired by nature and biological systems is one way for IT professionals to gain the initiative required to manage complexity, especially for large networks like community nets.

Autonomic communication defines a self-organizing network concept and technology that can be situated in multiple dynamic contexts. Therefore, autonomic communication needs to use artificial intelligence algorithms to build and maintain models of what the network is supposed to do.

To avoid the high cost of system failures, it is imperative to investigate ways to dynamically validate these systems. Self-managing features in autonomic systems dynamically invoke changes to the structure and behavior of components that will be operating in unpredictable circumstances.

So far there is still a lack of development in the testing of these types of systems at runtime. If code gets corrupted or faces explicit instruction deletion by an attacker, how will the system react? How can developers make the self-diagnosis mechanism itself robust to instruction loss? How can they trigger a self-healing mechanism to rebuild lost rules from original nonrobust code or by writing new rules?

Currently, examples of autonomic and adaptive systems include routing protocols in ad hoc networks and service discovery protocols. However, future computing will be driven by the convergence of biological and digital computing systems and be characterized by self-awareness, self-configuration, self-healing, self-organization, and self-optimization.

Although “the social benefits of ubiquitous, community broadband are becoming obvious,” the removal of central control over networks has the potential to release an enormous burst of creativity for new economic activity, driving economic growth and social change. 

Hong-Lok Li
libl@ams.ubc.ca

ERROR DETECTION
Regarding “The $100,000 Keying Error” by Kai A. Olsen (The Profession, Apr. 2008, pp. 108, 106-107), I agree that the banking software/system’s error-detection performance should be improved. However, good user interface design should also be employed to reduce the number of user errors that the system is faced with.

I’m not a user interface professional, but one idea that occurred to me is to break up the account number so that it’s harder to enter the wrong number of digits. The number could be printed and displayed as 715-815-550-22, and fill-in boxes on the screen could be structured the same way so the user would be much less likely to try to fill in the wrong number of digits. The software also might “ring a bell” or in some other way provide notification if the user tries to enter too few or too many digits.

Merlin Dorfman
dorfman@computer.org

I have another real-life data point that parallels Kai Olsen’s observations.

About a year ago, I was paying a utility bill using my bank’s online payment system. I accidentally used a comma instead of a period to separate the cents from the dollars. (In many countries a comma is the usual cents separator.)

My bank’s software silently strips all comma characters from input. Thus, it interpreted my entry as a value 100 times larger than intended. Usually there would not have been enough money in my account, but by bad luck I was getting ready for a large transaction. By further bad luck, I was called to come to dinner just as the confirmation screen came up, and I clicked on OK without checking as carefully as usual.

I was eventually able to get the large overpayment reversed, but it took weeks and held up my planned transaction.

I sent an e-mail message to my bank strongly suggesting specific improvements in their user interface. I got a prompt form reply saying that they would take my comments under consideration. So far their software has not changed, and I rather doubt that it will unless they are forced to take action.

Alan Quirt
aquirt@ieee.org

The author responds:
The bank gave in. At the last minute, before the case went to court, it offered to cover all losses the consumer had in this case. Fossbakk gets the money back, including interest on the amount. I am very glad on behalf of the consumer that her case was solved. However, this case should have been tried. It would have been nice to have a court decision showing that the user-interface designers have a responsibility in these cases.

Anyway, the Fossbakk case seems to set some sort of precedent. It has been extensively debated in the media and also in professional journals. Let’s hope that this forces everybody to take a second look at their user interfaces and perform the necessary usability studies.

As users, we make mistakes. It’s the responsibility of the systems to capture these mistakes before they have any serious consequences.

Kai A. Olsen
Kai.A.Olsen@hiMolde.no

We welcome your letters. Send them to computer@computer.org.