SOLVING THE SKILLS SHORTAGE

I read The Profession column by Neville Holmes titled “Digital Technology and the Skills Shortage” (Mar. 2007, pp. 100, 98-99) with great interest. However, Holmes did not explicitly proffer a practical solution for how to solve the present skills shortage plaguing various countries. Even the digital technology he suggests requires systems architects, software developers, expert systems specialists, psychologists, educators, and so on to come up with efficient solutions.

How can we apply his model to solve the skills shortage in Africa? Is the theory applicable to the skills shortage in Canada? Nurudeen Aderinto
nurudeen@aderinto.net

Neville Holmes responds:

There are perhaps two important points I did not make clear enough.

The more general point is that for any individual, the acquisition of skills is cumulative. Only people with relatively basic skills can develop advanced skills. This is why my essay focused on the development of skills in the very young, who are best able to learn skills, and who are then best placed to become highly skilled. Of course, the technique is also useful for older individuals.

The more particular point is that the digital technology with which to implement the drill and practice needed to induce a high level of skills already exists, and it is routinely applied in videogame development in a form much more elaborate than is needed for drill and practice. The barrier is not the lack of technology or technologists but the lack of any financial incentive to apply them to basic educational problems.

As far as South African development is concerned, cheap drill and practice technology would have two major benefits. First, using it in South Africa would support teachers and parents in developing skills in their pupils and children. Second, used in countries like Australia, Canada, and the UK, it would enable such countries to develop skills in their own citizens so that eventually they would not need to import skilled people from countries like South Africa, where the need is actually greater than theirs.

A specific further advantage in Canada would be that drill and practice delivered by machine would make bilingual and trilingual education practical throughout the country so that it would come much closer to achieving its cultural aspirations, which would arguably greatly improve the country’s internal politics.

Neville Holmes
holmes@utas.edu.au

ANTISPAM TACTICS

Will the problem of spam get worse before it gets better? Without question, spam is on the rise, having taken a toll on many organizations around the world. These victims are recognizing their vulnerability to the risk of spam and looking for an effective solution.

“Vendors Fight Spam’s Sudden Rise” (Technology News, N. Leavitt, Mar. 2007, pp. 16-19) made a good point: Industry observers doubt that governmental regulation will succeed. After all, spam has no borders and no world government exists to police the matter. I believe that a well-defined organizational policy toward spam is more constructive.

Several antispam techniques that the article mentioned have a significant weakness: They can’t hold spam at bay. “Identifying e-mail senders” relies on the use of blacklists. Spammers can easily get around such lists by routing the spam through legitimate free e-mail accounts like Hotmail. Spam can also be routed through a chain of servers and addresses by removing the point of origin and faking their original addresses.

“Analyzing the e-mail content” counts on the recognition of specified keywords. This is problematic for accuracy. If a filter blocks an e-mail with certain words, it could also block other e-mails with more innocent words that are part of the blocked words, causing many false positives.

Using several security layers for message inspection is a better approach for dealing with creative spammers. Some vendors have deployed techniques like reverse DNS lookup (the message is blocked if the sender domain and DNS domain don’t match); threshold scanning (using a mathematical algorithm to scan predefined content dictionaries); or a policy-based plug-in management approach for whitelist/blacklist, antivirus, blank recipient checking, and image and attachment inspection.

In addition, gateway-based antispam solutions provide good network resource benefits for blocking spam before it enters the corporate network. This reduces the need for corporate mail servers and hardware for routing and archiving e-mail traffic.

Bringing spam to its knees will depend not only on antispam software but also on the way we implement the solutions.

Hong-Lok Li
libi@ams.ubc.ca

FIGHTING BUGS

In consideration of repair by rejuvenation, as described in “Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate” (Software Technologies, M. Grottke and K.S. Trivedi, Feb. 2007, pp. 107-109), note that software developers are in fact widely using less extreme forms. For instance, an application that is invoked, runs for a relatively short time, then exits, can omit much of its memory cleanup, since exiting releases all the memory at once. This also can exhibit the side benefit of better performance because memory management is unnecessary.

Other well-known techniques involve resetting numeric registers due to cumulative error; copying and compacting data in memory to improve paging performance and prevent memory fragmentation, subsequent memory exhaustion, and failure; and reinitializing programs upon the change in some static parameter, thus avoiding difficult and error-prone incremental-update design for something whose occurrence is rare.
But these are all deliberate techniques. What I find disturbing about rejuvenation as described in this article is that it is being used to compensate for lack of thoroughness in design. Indeed, it is possible to build reliable systems. We need only look at the operating system sphere. I use some operating systems that stay up for months—that I only reboot if the power or hardware fails. And I use some that I need to reboot every couple of days, as “stuff accumulates.” I won’t mention any names.

While it is statistically valid that rebooting a system prior to the appearance of symptoms might forestall age-related problems, it’s not an engineering approach to a solution. It assumes preexistence, taking the system as an immutable natural artifact to be tamed and domesticated. It also falls into a clever trap: The software industry has managed to convince consumers that it’s okay for their computers to be unreliable and that, to avoid failure, they need to clean their systems periodically, much as they need to change the oil in their cars. But these techniques are simply encouraging software developers to continue their hasty, haphazard approach to design and implementation.

The public should be demanding long-term, reliable operation from the systems they purchase, not better methods to change the oil. Lawrence A. Stabile l.stabile@comcast.net

The authors respond:

We agree that software faults should be avoided whenever possible. Any advances in software engineering that enable developers to release better software are to be applauded. However, the state of the art fails to achieve (let alone guarantee) the desirable goal of fault-free software.

Like recovery-oriented computing, software rejuvenation therefore accepts the existence of (some) residual faults and tries to offer additional lines of defense against their consequences. To use Lawrence Stabile’s analogy: As long as engines require motor oil, we might want to change it from time to time, rather than claim that lubrication should not be necessary—and eventually lament a piston seizure.

Lawrence Stabile correctly points out that in addition to system reboots and program restarts, there are less intrusive rejuvenation techniques, like process restarts. Sometimes, software developers employ these techniques without deliberately addressing the software-aging phenomenon. For example, in our analysis of data collected on an Apache Web server, we discovered that the default weekly log rotation under Linux triggers Apache to kill all of its child processes and thus incidentally rejuvenates the system.

Of course, any technique can be abused or could set wrong incentives. Software rejuvenation might take the pressure from developers to deliver software with a low fault content. In the same vein, we might argue that the improvement of testing strategies undermines the motivation of programmers to avoid implementation faults in the first place. However, this clearly need not be the case. In the end, software developers will try to achieve an optimal mix of various strategies to avoiding or dealing with bugs, based on their respective marginal costs and benefits.

Software rejuvenation is no silver bullet. It is one of many weapons in the software developer’s armory for fighting bugs.

Michael Grottke Michael.Grottke@duke.edu
Kishor S. Trivedi kst@ee.duke.edu

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