COMPUTER PROGRAMMING AND SOCIETY

I found the article by Neville Holmes on the problem of the ingrown nature of present-day computer programming very interesting (The Profession, “The Craft of Programming,” May 2008, pp. 92, 90-91). Holmes is certainly right in stating that the plugboard method of programming (connecting entities that perform certain tasks) is both natural and efficient, but then the question is one of timing.

However, this article does not emphasize one very important aspect of early computers. After the CPC, most computer programming input was done with typewriter-like devices: key punches, teletypewriters, and, later, keyboards. It is my strong feeling that executives and managers were led by this fact to equate programming with secretarial work and give it a very low status in the hierarchy of positions, in turn leading to the situation that Holmes describes so well. If an audio input device had been perfected in the 1950s, the development of computer programming might have been quite different.

Richard Clow
dikclow@yahoo.com

E-MAIL POLLUTION

I would like to pass on my comments regarding the July The Profession column (A-F. Rutkowski and M. van Genuchten, “No More Reply-to-All,” pp. 96, 94-95).

At AMERICAN SYSTEMS we, like other companies, seem to be in a situation of dealing with ever-increasing e-mail pollution. Between the amount of spam—we are currently blocking around 2 million spam messages a week—the proliferation of e-mail newsletters, unnecessary carbon copies, and blind carbon copies, it seems that many of our employees are having trouble finding the time to sort the wheat from the chaff as it were.

This well-written article offers actionable advice to help get the monster back in the cage.

Thank you to all concerned.

Don Young
donald.young@americanystems.com

SECURITY IN VIRTUALIZED ENVIRONMENTS

In “Virtualization Sparks Security Concerns” (Technology News, Aug. 2008, pp. 13-15), Steven J. Vaughan-Nichols wrote, “Virtualization is rapidly becoming a standard technology for businesses.” Today’s technological requirements include ease of access, security, reliability, and a balance between performance and cost.

Virtualization is still at an early stage, and technologies and applications in this segment are continuously evolving. Nevertheless, security issues should not thwart the business agility of IT professionals. The question is how to make the security infrastructure flexible enough to coexist with and contribute to a virtualized environment.

It’s easy to set up virtual machines, and they can run a variety of operating systems and applications on the same host by isolating different workloads. Virtualization also greatly helps reduce time and costs for disaster recovery operations. However, security is still a critical issue for implementing virtualization systems.

In a virtualized environment, IP addresses change as virtual machines are created or moved from one physical server to another. IP addresses do not identify servers because servers can be redeployed on the fly to a different subnet. Multiple virtual machines sharing one physical system often use a sequential range of IP addresses. In some cases, these virtual machines have identical local administrator passwords. If one is cracked, all servers with similar characteristics will be victims.

In addition, the possibility that one compromised virtual machine could infect all VMs on a server cannot be overlooked. Therefore, protecting virtual servers presents difficulties for firewalls and intrusion-prevention systems.

The critical point of security leverage in a virtualized environment is within the hypervisor itself, not behind it. Dynamic security solutions must be plugged into this crucial layer to protect against attacks on known vulnerabilities percolating behind the hypervisor layer as new virtual servers are created and moved.

Hong-Lok Li
libl@ams.ubc.ca

LANGUAGE AMBIGUITY AND IDIOSYNCRACIES

In his comments in “The European Union and the Semantic Web” (The Profession, Aug. 2008, pp. 108, 106-107), Neville Holmes fails to appreciate that languages, both natural and artificial, have ambiguity and idiosyncrasies because they are used by people—the same people who would be using his e-speranto.

There isn’t any known mechanism by which a language can bind its users to particular semantic understandings.

Consider mathematics as an example of an “artificial” language. One of the known problems for searching the mathematical literature is that the semantics of the terms used vary from one mathematical community to another.

The more generalized the use of a “unified standard vocabulary,” the more it will exhibit the ambiguity and idiosyncracies it was supposed to cure.

Instead of treating the ambiguity and idiosyncracies of languages (both natural and artificial) as a problem, why not seek better techniques for addressing those issues?

Patrick Durusau
patrick@durusau.net

Neville Holmes responds:

I’m sorry that I didn’t emphasize enough that my proposals, both of E-speranto for the European Union...
and of a related grammar for vocabularies in the Semantic Web, were not intended for direct communication between people, only indirect.

This is not a new concept. The highly multilingual Persian Empire of more than two millennia ago was held together for two centuries by the use of written Armenian as an intermediary language, although Armenian speakers were relatively few.

For E-speranto to serve as an intermediary language within the European Union’s central bureaucracy, rules to expunge ambiguity and idiosyncrasy from it could be quite easily enforced by the relatively few translators and interpreters who would be using it, especially if their use of it were supported by digital technology. This unnatural sterilization would be important to prevent more general use of E-speranto from displacing minority languages and their cultures, which would be a social tragedy. Note well that esperanto differs from this fundamentally, being conversational and rich in idiom. It has even developed dialects.

In the matter of Semantic Web vocabularies, I was not proposing the use of E-speranto, only the use of its phonemic synthesis. I was suggesting the construction of a single vocabulary to act as an intermediary between the various vocabularies in various fields and languages. Of course, idiosyncracies and ambiguities would crop up in these various vocabularies, but the single intermediary vocabulary would provide a norm to allow catering for these natural and often admirable inconsistencies.

Because no one would directly use the central vocabulary, it would be possible to keep it consistent, even with communities inconsistently using a vocabulary in the same field and original language. It might not be politically or socially easy, but the benefits would surely persuade enough people of good will to work together to administer the vocabulary.

FLY BOTTLES

Regarding the article by Bertrand Meyer titled “Seven Principles of Software Testing” (Software Technologies, Aug. 2008, pp. 99-101), I think many of the problems we programmers have are philosophical.

In “The Nightingale of Keats,” Borges said, “Coleridge observes that all men are born Aristotelians or Platonists. The latter feel that classes, orders, and genres are realities; the former, that they are generalizations. For the latter, language is nothing but an approximative set of symbols; for the former, it is the map of the universe.”

Maeterlinck’s bees are Platonists. They are transfixed by the ideal sun. The flies are more Aristotelian—they believe that random behavior has its place, and eventually escape. Wittgenstein wrote, “What is your aim in philosophy? To show the fly the way out of the fly-bottle.”

Programmers are covert Platonists. We imagine a perfect abstract program and write something down that we hope matches our imagination. Then we run one test, and if it works, we say, “Yup, this is the program I imagined.” We will continue to encounter disappointments and malfunctions as long as we persist in this approach to reality.

Testing is a way out of the fly-bottle, but it doesn’t always work. Some of the flies die before they make it. If the exit is small and the flask large, random flying about might take too long. The flies need a map, a theory of exits—or they need to avoid entering bottles. Systematic, principle-led testing is like mapping the bottle; avoiding errors is the way to stay out of the bottle in the first place.

My friend Roger once showed me his plan for a software system. He planned so many months to write the code, so many to debug, and so on. I said, “You mean you plan to write code with bugs mixed in, and then strain the bugs out?” He replied, “Sounds kind of dumb when you put it like that.”

Tom Van Vleck
thvv@multicians.org

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