The Future of the Computing Profession: Readers' E-mails

Neville Holmes, *University of Tasmania*

Where are we going as a profession?

In an essay written for this column's tenth anniversary in July ("The Future of the Computing Profession," pp. 88, 86-87), I focused on an issue I felt was of supreme importance, one that many of my earlier essays had also touched on, directly and indirectly. For quite a while it seemed I was wasting my time as I only provoked one short e-mail reaction. But when I was well into writing the essay I had originally intended for this month, two others arrived in quick succession.

In the hope of provoking more readers to think about where we are going as a profession, with the senders' agreement I decided to publish those messages here.

E-MAIL 1

I have just read your timely and accurate article in *Computer*'s July issue. While working in information security as an analyst, I've been struggling to understand the place or role of a computing professional. I program and do security-related tasks, as more of a technician on these matters.

As of August, I have taken on a new role in a research environment. The finer points in your article are exactly what the manager hired me to deal with. He has no one to do the implementation that can keep current in terms of the technology and tools of the trade. He has the big thinkers, but not what you describe. It seems like a better fit for me. So I totally agree with your sentiment.

Paul O'Neill
paul@codelogic.net

E-MAIL 2

In your July essay, you argue that the computing profession must first separate its professionals from its technicians, just as other professions do. For example, architects have builders, lawyers have clerks, and so on. In the computing profession, programmers would be the technicians. However, I must disagree. The computing profession is not like other professions in a fundamental way.

The computing profession is different because the product being produced is different. The result of an architect's efforts is a specification, not the building itself, so it is necessary to have a builder come in to execute the architect's specification, if the planned result is a building. In computing, the specification—what we call the program—is the end result. Once the program has been produced, the effort is done. There is no next step to be carried out.

Now, it's possible to argue that the result of the computing professional's effort is not the program, but what is it then? It certainly must not be a specification written in English. I have seen that and done that, and the design document written for the consumption of other humans is never sufficient for the implementation to succeed. Too many questions remain unanswered, too many subtle design decisions turn out to have major ramifications for the implementation, and perhaps most important, it is simply never precise enough.

No, the result of the computing professional's effort cannot be a design document. The architect does not draw some sketches about what the building looks like and then ask the builder to build it. The architect produces a detailed set of plans for the builder, and the builder follows them.

In my experience, by the time we get to the level of detail sufficiently precise for the implementation to actually correspond to the design, we might as well have written the program. What is a program anyway? It's nothing more than a specifica-

Continued on page 98

Continued from page 100

tion written in a formal language with unambiguous semantics. How else do we specify what the system is supposed to do than with some such specification? We may not be satisfied with the languages we have today to specify what a system should do—they may be too low-level, they may be too closely bound to a particular hardware platform—but the fault then is with the languages, not the process.

Uniquely, with computing as opposed to other professions, the computing professional need only do something once. For example, the architect may have draftsmen who add the electrical wiring details to the plans after the basic structure is done, and this needs to be done for every job. But the computing professional has the power of abstraction built into his tools.

I do not need programmers to implement a sort routine that I specify each time I need one. It need only be implemented once. The same is true for a more complex artifact—say, a database or webserver. If the abstraction is well understood, it can be used repeatedly. If it's not well understood, the computing professional must specify it to the level of detail that makes it well understood. And the formal language we use to specify it to the appropriate level of detail is and must be the program. There is nothing else.

That tools and techniques for programming are continually changing is a reflection both of improvement in the abstractions we understand and in the underlying hardware. For example, I use different languages now than I did 10 or 20 years ago. These newer languages let me build systems that do things that simply wouldn't have been possible then. I couldn't have managed the complexity required by the system.

This isn't an issue to be dealt with only by the programmer, though. This is an issue for the designer. Is the availability of multiple cores a technical detail, or does it fundamentally change the design? If new materials become available in construction, is this not of critical importance to the architect? Doesn't steel allow different construction possibilities than wood? If the building codes change, this is of importance to more than just the builder. It may alter what it's even possible to build. The same analysis is true for other professions you listed. Pathologists perform tests ordered by doctors because the tests have been standardized, but the work of carrying out the test still must be done.

When a computing professional comes to understand the solution to a problem well, that understanding is in the program itself.

But when a computing professional comes to understand the solution to a problem well, that understanding is in the program itself. The program may be analyzed and documented, but the program is its own end. If the desired result is the process specified by the program, we don't need a technician to carry it out. We need only execute the program on a computer. The computer is the technician.

This may be why the computing profession is, as you point out, so distinctively tethered to the computer. Sometime in the future, other fields may replace their technicians with computers as well. It may only be a matter of time. But they will only be able to do it with the help of computing professionals, those who are expert in making such systems possible. So I don't think we are headed toward irrelevancy just yet.

Jeffrey Olkin
jeffrey@olkin.net

E-MAIL 3

I had a few thoughts to share regarding the July The Profession column.

First, I don't think that OS makers will go back to the command-line interface, because—based on their view of their customers—that would be regressing. So I think the question to pose is how to incorporate the scripting elements of the command-line interface into a GUI-based system.

One thing I've always liked about Unix was the ability to string together relatively simple commands with pipes to perform complex operations, as you allude to in the column. If something similar could be done graphically, with connections (pipes) drawn between components (software libraries) to tie together a dataflow to accomplish a particular objective, we would be on the way there. I believe that such things exist for software development; the issue would be convincing the developers to build their OSs and GUIs to support it. So, rather than look back to the old paradigm, look forward to a new (old) one.

Second, you sketch out a picture of the computing technician but don't really describe what the professional level looks like. Perhaps that's in the 2007 article you cite. I didn't go back and check (I'm at home and the Web library access is at work).

Third, another article in *Computer*'s July issue (A. Gowan and H. Reichgelt, "Emergence of the Information Technology Discipline," IT Systems Perspectives, pp. 79-81) discusses the emergence of IT as a distinct discipline, as opposed to CS and IS. The authors' view is that IS drives the requirements (map this process into a computer system), while IT builds the required systems and networks, and CS acts as a bridge between the two (writing most of the software to get the job done). Is this a tri-level version of what you're proposing, or would you say that each discipline should have technicians and professionals within it? If the latter, what would those positions look like?

Fourth, one risk of what you propose is a rigid stratification. I think of the medical system here in the US, where there is a definite hierarchy of doctors, nurses, techs, and others, with many specializations fitting between the others, such as nurse practitioners, which are a step up from nurses but a step down from physicians' assistants, who are themselves a step down from doctors.

While there is specialization within the computing profession now, we can move around and, more particularly, up the chain, given the limitations of our own abilities, opportunities, and so on. Currently, this is often based on a mixture of experience and education, so that going back to get more advanced degrees is advantageous, but not always required. With greater professionalization, I would think that there would be more emphasis on formal schooling, and perhaps less on experience—which I'm not sure is an entirely good thing.

Jonathan Entner
jonathan.entner@cavtel.net

DIFFERING VIEWS

Jeffrey Olkin's view of the profession's structure is not mine, nor is his expectation of architects. Roger Fay, head of the School of Architecture and Design at the University of Tasmania, confirms my view. "The responsibility of the architect starts and ends with the client. With that in mind, the architect should be involved from the first tentative chats with the clients through to the completion of construction." This is not all. "Since buildings are complex objects, clients often need to be instructed on how to get the most out of them, so this is another postconstruction role for an architect."

Fay's depiction of the architect's role is very close to my view of what the computing professional's role should be. It is also very close to the UK view of the engineering professional summarized in Table 1 of *Computer*'s very interesting August 2010 Education column (S.T. Frezza, "Computer Science: Is It Really the Scientific Foundation for Software Engineering?" pp.98-101), which is very much like the view I was given when studying engineering some 60 years ago.

So I was somewhat dismayed by the redoubtable Peter Denning's essay in a recent *American Scientist* (tinyurl.com/2dtzcdv), in particular because of its illustration and its ignoring of the international standard definitions of *data* and *information*. The illustration puts engineering at home among the sciences and depicts computing as about to join them, a viewpoint contradicted by the Education essay. Using those standard definitions would have made Denning's discussion much simpler and the problems much clearer.

As for the *Computer* article Jonathan Entner mentioned, I only wish I had known of it at the time. It makes plain that CS and IT courses focus on the technology and are thus producing technicians or technologists, while IS courses focus on the client and thus produce what I deem proper computing professionals akin to engineers and architects. The only problem is that IS only considers clients in the business world and ignores many kinds of clients and partners (The Profession, Jan. 2007, pp. 114-116).

Much more could be said in response to these messages, but Paul O'Neill's e-mail encourages me to believe that some readers share my point of view. The issues are important, as can be seen by reading through the position statements of the CS election candidates in *Computer*'s August issue. As column editor, I would be delighted if readers continued this debate, either by sending e-mails like the ones included here, or by submitting essays of around 2,000 words for publication in this column. C

Neville Holmes *is an honorary research associate at the University of Tasmania's School of Computing and Information Systems. Contact him at neville.holmes@utas.edu.au.*