

# Computers, Programming, and People

Neville Holmes, University of Tasmania

In the *Technology Quarterly* that *The Economist* bundled with its 22 September 2001 issue I found an odd article called “A Lingua Franca for the Internet” (pp. 12-16; <http://www.economist.com/>). I found the article’s pedagogical lead-in—a somewhat fanciful history of program coding—peculiar. Likewise, the contradiction between the article’s title and its closing clause, “expect a whole alphabet soup of languages within the next decade,” seemed odd.

More peculiar still was the contrast between the first-page highlighted claim that “the business of writing software is becoming steadily easier,” and the author’s observation regarding object orientation’s widespread adoption that “The price to pay for objects is ... making the language bulkier and more cumbersome to use.”

An amusing appendix, “Programmer’s progress,” reflected this discord. Attributed to one Joe Garrick, it showed progress using different versions of the banal anthropomorphic Hello World program. The first entry, in Beginner’s All-Purpose Symbolic Instruction Code, read as follows:

```
10 PRINT "HELLO WORLD"
20 END
```

Garrick showed programmers’ growing experience beyond this Basic sample by increasing the code’s length and



**The computer can be a commercial artifact or a household tool, but which should it be?**

complexity. At the progression’s high end—the Seasoned Professional and Master Programmer categories—Garrick depicted the entries as excerpts torn from complete versions too large to be shown whole.

## COMPUTER ARCHITECTURE

*The Economist* article made a strong impression on me because recent teaching experience had brought me face-to-face with similar contradictions. Last semester, I acquired a second-semester, first-year course called “Computer Organisation and Architecture.” Believing that learning can best be done by doing, I sought to build in a strong practical component.

The textbook I selected had a fairly detailed description of John von Neumann’s IAS computer, complete with several diagrams and a table of operation codes. Therefore, I based two major programming assignments on direct use of simulated IAS machines, simplified to one instruction per 20-bit word. Students would write the first

program in hexadecimal machine code, the second in symbolic code.

I was completely unprepared for the difficulties many of the class experienced with this assignment, including some of the better students. For example, despite detailed online instruction and repeated demonstrations of the process in class, many students could not learn how to do bootstrapping. They seemed unable to distinguish between the commands used to operate the machine and its programs’ instructions. Although some students gave thanks when I finally provided a command specifically for loading and

starting programs in one step, my contribution left others yet more perplexed.

Some students shunned certain kinds of instructions, notably those with address-modifying operations—the IAS computer has no index registers—and those with immediate operands. In symbolic code, many students insisted on putting data definitions up front but still expected to start their programs at their load point.

Many students struggled to fix in their minds a clear image of the machine on which their programs would run. Discussions confirmed the impression that their prior and continuing instruction in Java programming now interfered with their understanding of the IAS machine’s sequentiality. Two possibilities follow:

- there is no point to having professionals understand the nature of the machines they use and code for, or

*Continued on page 110*

```
public class HelloWorld extends javax.swing.  
    JComponent {  
    public static void main (String[] args) {  
        javax.swing.JFrame f = new  
            javax.swing.JFrame("HelloWorld");  
        f.setSize(300, 300);  
        f.getContentPane().add(new HelloWorld());  
        f.setVisible(true);  
    }  
    public void paintComponent(java.awt.Graphics g) {  
        g.drawString ("Hello, World!", 125, 95);  
    }  
}
```

**Figure 1. Java Hello World applet.**

- there is some significant general benefit to understanding these machines.

Sound arguments can be marshaled to support either view.

### COMPUTERS ARE IRRELEVANT

Now part of consumer society, and marketed globally like hamburgers and soft drinks, computers are designed to be *sold* rather than *used* (Robert Glass, "Of Model Changeovers, Style, and Fatware," *Comm. ACM*, Sept. 2001, pp. 17-20; <http://doi.acm.org/>). Likewise, developers have designed the software that runs on them to have built-in obsolescence so that both commercial and private sales will continue to grow.

The industry's main software products are already so gigantic that supporting their continuing growth, or developing competitors for them, presents more of a management problem than a technical one. Object-oriented techniques and component technology suit such an industry ideally. Accessing the Internet—arguably computers' most popular use—involves exploiting user interaction and delivering stimulating visual experiences. Elaborate special-purpose software packages provide the perfect tools for developing such applications.

Software marketers, e-businesses, and their technicians consider having an understanding of how the underly-

ing machinery works to be utterly irrelevant. For them, the most important skills are marketing, management, and graphical expertise.

Under this regime, computer organization and architecture should be confined to engineering schools. There, the most talented of an increasingly innumerate and illiterate public can learn to design and produce extraordinarily complex computers powered by microchips whose impressive clock speeds and astronomical transistor numbers make for such impressive ad copy.

### COMPUTERS ARE RELEVANT

Although computers now play a major role in the consumer market, their potential *noncommercial* usefulness to society and the world remains virtually untapped. Commercial computers and software could be exploited for purposes quite different than those their makers intend.

Cheap and obsolete computers, particularly when networked, could do much to help reduce poverty and inequity by supporting educators in poor and underprivileged sectors of society. They could also assist organizations that promote such poverty reduction and education, such as PCs for Kids (<http://www.pcsforkids.org/>).

Yet, if educators are to somehow reverse society's growing illiteracy and innumeracy, they will not do so through their use of commercial software. They

must use their own knowledge and control of a variety of programs, foreseeing and prescribing the computer uses that will meet their objectives.

These potential uses can best be discovered by people who can program novel applications such as drill and practice for skills fundamental to literacy and numeracy. This trend, once started, could become self-perpetuating: Any increase in numeracy or literacy would likely increase the number of people able to create innovative software.

This approach requires many people to develop uses for computers that extend beyond strictly commercial and profitmaking ones. Such applications may well be for personal or private use, but they present technical challenges rather than organizational ones. Solving these challenges will require a wider understanding of what computers can do and how they do it.

### PROGRAMMING AND PEOPLE

We put computers to novel use by programming them for such tasks. For professionals, the coding of programs has long been indirect, using coding schemes that make a virtue of their distance from the reality of the machines on which they will run. Compare the Java *Hello World* applet in Figure 1—described by its source as "minimalist"—with the Basic example I cited earlier.

Coding schemes like this one are unsuited to popular programming. Quite opaque to the inexpert, they fall prey to the whims and fancies of professional fashion because programming techniques and tools undergo continual development and frequent reinvention.

Some coding schemes such as Basic were originally intended for popular use, while others such as Cobol were intended for use by ordinary business people. These languages, however, have been oriented to objects rather than to the processes that make a sequential machine what it is. An ex-colleague with very long experience in

the computing industry recently described Visual Basic to me as “truly appalling.” Certainly, it’s quite unsuited to popular use given that, according to Bruce McKinney, “Visual Basic designers have chosen to pile more and more doodads on a weak foundation, knowing that doodads, not foundations, sell boxes” (<http://www.vb-zone.com/upload/free/features/vbpj/1999/mckinney/mckinney1.asp>).

As the Java example suggests, object orientation is valuable in the development of complex programs because it brings discipline to the exploitation of the increasingly complex services offered by operating systems and servers. This observation implies that machine-independent coding schemes are unsuitable for popular use—or quickly become so.

## COMPUTERS AND PEOPLE

To exploit computers for the popular good, we must provide machine-oriented program coding in a stable and simple context. This requirement rules out using the object-oriented Java virtual machine, the hypercomplex Itanium with its 13-bit operation code, or similarly complicated approaches. Popular computing could be based instead on international standards for

- a simple, stable computer organization and architecture to be provided by simulation or emulation within commercial systems;
- a symbolic programming system supporting the coding, development, and testing of programs for that architecture;
- an operating system providing services according to the standard organization; and
- a macrocoding system to simplify interfacing with the operating system and to support exchange and reuse of code fragments.

Standards for these four components would need professional consideration and international negotiation. The following rudimentary ideas are an illus-

tration of possibilities, not a final design.

The computer architecture should be as simple as possible to program with, but versatile. For this task, a single two-operand instruction format would be suitable, without registers but with provision for indirect addressing. Operand tagging would greatly reduce the number of operation codes. Storing instructions and operands separately might be wise as well.

**The profession should strive to make it easy for people to exploit the computer on their own terms in their own culture.**

The symbolic programming system should be completely straightforward, but adaptable to different writing systems. Any complexity should be subsumed by the operating system or concealable within the macrocoding system.

The operating system should be completely in the background, and it could be an existing standard such as Posix. The OS should support program testing and tracing, and it should allow other programs to invoke tested programs as though they form part of the OS. Programmers should be able to ignore the host operating system’s complexities while still having access to stored files of all kinds.

The macrocoding system should make the organization of the notional computer, and the services of the notional operating system, available to the programmer in simple format. Finally, the macrosystem should let skilled programmers code macrodefinition sets to provide toolboxes for simplifying application coding in specialist areas.

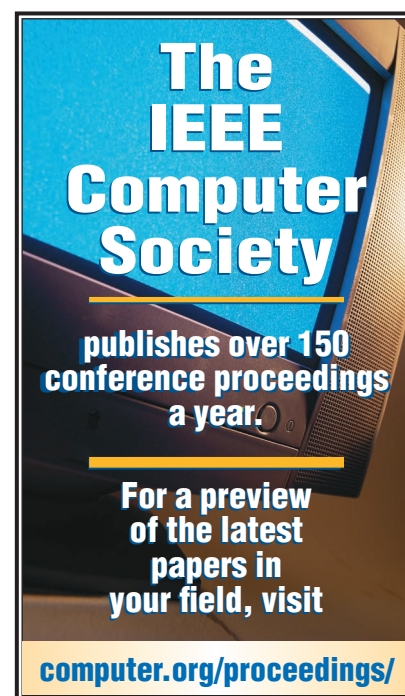
**F**uture possibilities for the computing industry and profession fall into a spectrum. At one end,

the computer and its software primarily function as commercial artifacts, with users mainly filling the role of consumer. At this end, the computer is irrelevant to most computing professionals, and professional education focuses on developing and managing complex program suites for the ever more complex stimulation of, interaction with, and control of users and customers.

At the spectrum’s other end, the computer serves as a household tool and the user controls its uses. Here, the profession strives to make it as easy as possible for everyone to exploit the computer on their own terms and in their own culture, and it may even champion teaching the computer’s direct use in elementary schools.

As things stand, the industry seems to be going in the first direction. From any ethical and moral standpoint, the profession should be pushing to move in the opposite one. ■

*Neville Holmes is an honorary research associate at the University of Tasmania’s School of Computing. Contact him at [neville.holmes@utas.edu.au](mailto:neville.holmes@utas.edu.au).*



**The IEEE Computer Society**

**publishes over 150 conference proceedings a year.**

**For a preview of the latest papers in your field, visit**

**[computer.org/proceedings/](http://computer.org/proceedings/)**