This is the new science of collocation and colligation that illuminates how texts work.

Future of Texts

Corpus linguists will have to work fast to keep up with the changing nature of texts. As texts become shorter, more fragmentary, and multimodal (using pictures, color, sound, kinetics as well as words), so strategies of interpretation and ways of reading will change.

A struggle is brewing too between author and reader, the producer and consumer of texts, which has many of the dimensions—political, economic, social, technological—that characterize postmodernity. On the one hand, multimodal texts need more attention by designers and editors to marshal disparate forms of information into a coherent whole. But against them is a movement—at times fundamentalist in fervor—that demands free access to "content," and argues that publishers, editors, and designers are part of a capitalist conspiracy to add cost and control access to knowledge. Digital texts may mark the death of design—which will become a matter of a reader's preference setting. But technology also gives publishers new freedom to reversion intellectual property, to make it look different to different categories of reader, and to sell text by the paragraph. The linguistic resources required to construct and interpret longer, unified texts—which collectively form institutionalized genres—may be lost in all but specialized domains such as the scientific article. Readers will be left to make sense of fragmentary, often contradictory information dispersed across different channels.

Will the Future Understand Us?

When Thomas Sebeok, an American specialist in semiotics, was asked in the 1980s to advise on a method of communicating the whereabouts of dangerous repositories of radioactive waste to generations 10,000 years hence, he concluded that there was no secure means of transmitting such knowledge over 300 generations. Instead, he recommended putting in place a relay system which ensured that "as the information begins to decay, it should be updated" and argued that any messages written in English should be designed for only three generations ahead—that is, 100 years (*9*). This may seem a short horizon—if a linguist were faced now with a typical text from the 22nd century, he or she would be unlikely to conclude that the language has radically changed in its core vocabulary or grammar. But we might not be able to make much sense of it.

References

1. D. Graddol, *The Future of English?* (British Council, London, 1997).
2. D. Graddol, in *English in a Changing World*, D. Graddol, U. Meinhof, Eds. AILA, Milton Keynes, 1999), pp. 57–68.
3. United Nations Environment Programme, Press Release, February 2001. Also at www.unep.org/Documents/Default.asp?DocumentID=192&ArticleID=2765.
4. D. Crystal, *Cambridge Encyclopedia of the English Language* (Cambridge Univ. Press, Cambridge, ed. 2, 2003).
5. D. Graddol, D. Leith, in *English: History, Diversity and Change*, D. Graddol, D. Leith, J. Swann, Eds. (Routledge, London, 1994), pp. 136–166.
6. European Commission, *Standard Eurobarometer 52*, http://europa.eu.int/comm/public_opinion/archives/eb/eb52/eb52_en.htm (2000).
7. N. Chomsky, *Syntactic Structures* (Mouton, The Hague, 1957).
8. E. Sapir, *Language* (Harcourt, Brace & World, New York, 1921), p. 39.
9. T. A. Sebeok, in *On Signs*, M. Blonsky, Ed. (Blackwell, Oxford, 1985), pp. 448–466.

VIEWPOINT

Software and the Future of Programming Languages

Alfred V. Aho

Although software is the key enabler of the global information infrastructure, the amount and extent of software in use in the world today are not widely understood, nor are the programming languages and paradigms that have been used to create the software. The vast size of the embedded base of existing software and the increasing costs of software maintenance, poor security, and limited functionality are posing significant challenges for the software R&D community.

We are living in a rapidly evolving information age. Computers, networks, and information pervade modern society. Some of the components are visible: Virtually every office and home is equipped with information devices such as personal computers (PCs), printers, and network connection devices. An increasing fraction of the population is using the Internet for tasks as varied as e-mail, messaging, searching for information, entertainment, and electronic shopping. The amount of information on the Internet is measured in exabytes.

Most of the infrastructure supporting the information age, however, is not evident. Today's information appliances such as TVs, organizers, and phones contain microprocessors and other forms of embedded computer systems. Telecommunications and Internet access systems are all controlled by networked computers. Wireless networks with voice and data capabilities are found the world over.

The information age has been thrust upon society, and everyone is being affected by the new technology. The information infrastructure is creating new opportunities for improving all aspects of life from childhood to old age. But the technology is also creating new challenges, especially in areas such as the security and privacy of information systems.

The Unappreciated Importance of Software

Few people appreciate the importance of software—until it breaks! The amount of software used by governments, companies, educational institutions, and people throughout the world is staggering. An individual system, such as a PC operating system, can consist of many tens of millions of lines of code. If we assume that there are 5 million programmers worldwide, each producing 5000 lines of new software a year (the industry average), then a conservative estimate is that the world is already using hundreds of billions of lines of software to conduct its affairs. Assuming that it costs somewhere between \$10 and \$100 to produce a line of working software, we see that the worldwide investment in software is in the trillions of dollars. A software system requiring tens of millions of lines of code would cost hundreds of millions of dollars to develop from scratch. The high cost of new software development is one of the principal drivers of the creation of open-source software, whose system development is essentially done for free by volunteer software specialists throughout the world. But open-source software has created another market oppor-

Department of Computer Science, Columbia University, New York, NY 10027, USA. E-mail: aho@cs.columbia.edu

tunity: companies that provide maintenance and customization services for users of open-source systems.

A more sobering figure lies in the number of defects in this huge embedded software base. Software managers estimate that the number of defects per million lines of delivered software ranges between 10 and 10,000. Assuming an embedded base of 500 billion lines of software, this would mean there are somewhere between 5 million and 50 billion defective lines lurking in the embedded base waiting to be triggered. It is not known how many of these are showstopper defects that result in the failure of a system, but all too often major critical systems fail because of software defects. It has been estimated that the Y2K software problem cost hundreds of billions of dollars to fix. Because software defects can result in huge economic losses, or in some cases the catastrophic failure of a system, a lot of software research is being devoted to more effective methods than testing for making reliable software.

How Programming Languages Have Evolved

The first programming languages were machine languages, consisting of commands expressed as sequences of 0's and 1's that told the computer what operations to execute. It was difficult and time-consuming to write programs in machine language, and once written, the programs were hard to decipher and modify. The first significant step toward a more human-friendly programming language was the invention of assembly languages in the 1950s. Assembly languages are still close to the machine, but when sequences of 0's and 1's were replaced by mnemonic commands, such as "load register X from memory location Y" or "store the contents of register X into memory location Y," programs became easier for humans to write and modify.

In the latter half of the 1950s, another major step was made in the design of more human-friendly languages. The higher-level languages Fortran, Cobol, and Lisp were invented: Fortran for scientific computation, Cobol for business data processing, and Lisp for symbol processing. After some initial resistance by hard-core assembly language programmers who felt that programs written in these languages would not run as fast as their assembly language programs, these languages became immensely popular in their respective communities. Fortran became the lingua franca of scientific computing, Cobol was for many years the world's most popular programming language, and Lisp became the mainstay of the artificial intelligence community. Their popularity stemmed from the fact that programs for the intended application areas could be written more rapidly, concisely, and easily in these languages than in assembler language. Indeed, these languages and their descendants continue to be widely used even today.

The 1960s saw the development of Algol 60, the first programming language with block structure. Although Algol 60 was never widely used, it is notable for the languages it influenced, including Pascal and Modula. Algol 60 also introduced BNF (for Backus-Naur Form), now a widely used grammatical notation for expressing the syntax of a language. For many years, Pascal was used as the introductory programming language in computer science departments.

John Kemeny, while at Dartmouth College, felt that every college student should know how to program; and in the early 1960s, he and Thomaz Kurtz created a simple imperative language called Basic, which was easy to learn. Basic spawned many dialects, the most notable of which is Visual Basic, arguably the world's dominant programming language today.

Another major development during the 1960s was the introduction of object orientation into programming through the creation of Simula 67 (*1*). Like Algol 60, Simula 67 itself was not widely used, but its influence was profound. Nearly every modern general-purpose programming language today supports object-oriented programming. Object orientation allows a programmer to focus on the design of the data (objects) and the interfaces to the data. It facilitates "plug and play" among software modules.

In the early 1970s, Dennis Ritchie at Bell Labs created the systems programming language C to implement the third version of the Unix operating system that he was codeveloping with Ken Thompson. In the 1980s, Bjarne Stroustrup, also at Bell Labs, created C++, an extension of C with object orientation. Because of their efficiency and early association with Unix, C and C++ became the most widely used systems programming languages.

Another genre of popular programming languages is the scripting languages: typeless languages with high-level primitives for manipulating data (*2*). Some of the most popular languages of today, such as awk, javascript, perl, php, python, sh, and tcl, are scripting languages. With these languages it is possible to specify programming tasks in a few lines of code that would otherwise take hundreds of lines in a lower-level language such as C or C++.

Trends in Modern Language Design

The newest major programming languages, Java and C#, build on the legacy of C and C++ but incorporate features that support the notion that the information infrastructure must be a robust distributed system. Many of the features in these languages are based on strong theoretical underpinnings, such as strong type systems and monitors, and on software design techniques that have been extensively explored by the computer science research community. Here are some of the design principles that the creators of these new languages have espoused (*3*).

Simplicity. Programmers want languages that are easy to learn, use, and understand. The newer languages tend to have support for features that make programming easier, such as automatic garbage collection; yet have a syntax that is familiar to C and C++ programmers.

Robustness. Because security and safety are of paramount importance in modern software systems, the new languages have strong type systems that allow more errors to be caught at compile time, and they restrict the use of pointers, which account for many of the vulnerabilities that hackers tend to exploit in C and C++ programs.

Portability. Programmers would like their programs to run on as wide a variety of machine architectures as possible, producing the same results. The newer languages have introduced mechanisms such as run-time bytecode interpreters, which allow the same program to be run, producing the same results, on different machine architectures.

Internet compatibility. Everyone would like to access software applications from anywhere on the Internet. The new languages either have class libraries or access to class libraries that facilitate the connection of programs with Internet protocols and applications.

Concurrency. Modern applications need to interact with many systems at the same time. The newer languages have multithreading and concurrency primitives that support the building of applications requiring simultaneous multiple threads of execution.

Languages of the Future

Software can be used in virtually any field of human endeavor. Just as there are different notations for dealing with chemistry, dance, law, mathematics, music, and so on, there will never be a single language for creating all forms of software. There are already thousands, perhaps tens of thousands, of programming languages in use today. Each field has at least one language that is used primarily by the practitioners of that field. Most college students today are familiar with languages for editing documents, formatting papers, creating presentations, and performing calculations. Each language is universal in the sense that it is capable of expressing any computation, but each language has been designed so it is easy to use for the applications it was designed for, be it hardware design or music synthesis. Naturalness and ease of use will most likely continue to be dominant influences in language design.

Some researchers are currently exploring more exotic forms of languages involving speech, gestures, pictures, and templates. It is not clear at this point whether multimedia will become a dominant feature in programming languages of the future, but it is quite clear that in certain application areas, these new forms of communication with computers are compelling.

Making Software Systems More Reliable

A fundamental question for the field of software development is whether there is a scientific basis for making reliable software. In 1956, von Neumann showed how more reliable hardware could be made from unreliable components by using redundancy (*4*), and earlier Shannon showed that unreliable communication over a noisy channel could be made more reliable by using error-detecting and -correcting codes (*5*). Today, redundancy and error-detecting and -correcting codes are routinely used.

No analogous technique is known for making reliable software. Although the production of software involves people as well as process and technology, the human component in the production of software has not been adequately understood or modeled. Some software researchers have advocated *N*-version programming, a technique where two or more people independently write a program from the same specification, but subsequently researchers have discovered that programmers tend to make the same kinds of mistakes even if they don't communicate with one another (*6*).

It is unlikely that humans will ever write software with zero defects. Researchers are actively exploring many techniques to make more reliable software systems, keeping the frailties of human programmers in mind. Static type checking and model checking provide promising avenues for detecting errors earlier in the software life cycle. A more ambitious approach is to see whether software systems can be designed to be resilient to errors. An even more ambitious approach is to design systems that automatically correct errors when they are detected. The goals of this research are laudable, but most likely it will be some time before we will see self-correcting software systems widely deployed in practice. What we can be certain about is that the embedded base of software will continue to grow in size, diversity, and functionality.

References and Notes

1. K. Nygaard, O. Dahl, in *Proceedings of the ACM SIGPLAN History of Programming Languages Conference*, R. L. Wexelblat, Ed. (ACM Monograph Series, New York, 1981), pp. 439–493.
2. J. Osterhout, *Higher Level Programming Languages for the 21st Century* (IEEE Computer, IEEE, New York, 1998), pp. 23–30.
3. For an overview of these principles, see http://java.sun.com/docs/overviews/java/java-overview-1.html.
4. J. von Neumann, in *Automata Studies*, C. E. Shannon, J. McCarthy, Eds. (Princeton Univ. Press, Princeton, NJ, 1956), pp. 43–98.
5. C. E. Shannon, *Bell Syst. Tech. J.* **27**, 379 (1948).
6. S. Brilliant, J. Knight, N. Leveson, *IEEE Trans. Software Eng.* **16**, 238 (1990).

VIEWPOINT

Of Towers, Walls, and Fields: Perspectives on Language in Science

Scott Montgomery

Language in science is in the midst of change and appears dominated by two contradictory trends. Globalization of scientific English seems to promise greater international unity, while growth of field-specific jargon suggests communicational diaspora. Real in part, each trend is complex and multileveled, and includes elements of convergence and divergence, along with important implications for the present and future of technical knowledge.

Science, it appears, has come to a historical crossroads. On the one hand, it would seem to have completed the Tower of Babel, its knowledge now reaching far beyond the heavens and, through the global spread of English, recovering the ancient dream of a single language for the wisdom of the nations. Yet, from another vantage, the very opposite is suggested: this great tower of unanimity broken and rebuilt into a thousand walls by the power of jargon, dividing the disciplines by the arcanity of specialist speech.

Two great trends of opposing force, two linguistic movements that annul each other's action. Is such a state of affairs real, and is it prevalent? What do the facts say, as far as we can discern them, and what are their implications?

These are not mere academic questions. Scientific knowledge exists because scientists are writers and speakers, users and sharers of language that, like all language, is constantly evolving. Words are the primary medium by which technical work is embodied, added to the corpus of professional understanding, and passed on. Whatever directly affects the speech of science and its development affects scientific endeavor near its core—its ability to express and render available its nuclear substance.

Take the case of English, then. How true is the claim that it constitutes an international language for science, an ever-expanding one? The answer is "very true," indeed, but with certain limits and qualifications.

1511 18th Avenue East, Seattle, WA 98112, USA. E-mail: scott.montgomery@prodigy.net

The Role of English

Dominant use of English in science must be understood within a larger frame. First, there is the advent of this tongue as a global language generally. British colonialism sowed the seeds early on, in North America, India, Australia, Hong Kong, and other centers of influence. Simultaneously, the Industrial Revolution gave English prominence in technological matters crucial to modernization. However, it has really been since World War II, which so greatly advanced U.S. military, economic, technological, and political sway (and thereby, cultural impact), that English has become linguistic capital for the larger world. Today, this tongue serves as lingua franca for a wide range of domains in daily experience: entertainment, advertising, travel and tourism, international business, telecommunications, the news media, computer technology. English is now the most popular, and most required, foreign language to be studied anywhere (*1*). Its uptake in technical circles, meanwhile, has been aided by the rise of "big science" in the United States and the resulting vast increase in scientific output. English, in a sense, has ridden a great wave of cultural and intellectual affluence.