New dimensions

Over the next few issues of IEEE Software, you may notice changes that we hope will add new dimensions to the magazine’s scope. Our highly regarded software reviews departments (Software Reviews and New Product Reviews) are being fine-tuned to bring you more technically oriented products. This means even greater emphasis on tools; leading-edge products for engineers, scientists, programmers, and their managers; and products designed for the specialist, developer, and knowledgeable computerist.

We are proud to increase our focus on two important dimensions in software engineering: quality assurance and human factors. MCC’s Vincent Shen, a recognized leader in software-quality research, will edit the Software QualityTime department, which will report important results, techniques, and practical experiences about software quality, validation and verification, and project management. This department will regularly feature brief reports by widely recognized people in this field.

Koffler Group’s Kathleen Potosnak, one of those rare individuals who is interested and experienced in human factors and software design, will edit the Human Factors department. She will cover the latest developments in user interfaces, human factors, and human-computer interactions technologies — and will help you apply this technology to everyday designs.

Both departments start this issue. Another new dimension I am personally working very hard to complete is the addition of Europeans, Asians, and other nationalities to IEEE Software’s editorial board to broaden the base of our technologists and give our readers insights into software developments around the globe.

Please let me know how we are doing: Your suggestions and concerns are valuable inputs to the process of adding new dimensions to IEEE Software.

Ted Lewis
Editor-in-Chief

SOFTWARE LETTERS

C++ impressions

To the editor:

The review of The C++ Programming Language (Book Reviews, July, p. 110) praises the book’s organization and readability but gives the reader almost no impression of the language itself. This is unfortunate because C++ is a relatively new programming language and the book is both its official definition and (for now, at least) its most readily available description.

The reviewers say, “The most useful feature of C++ is that it makes some of the things that are very complicated in C easier to accomplish,” but they forget to add “without any loss of efficiency.” This design goal (not feature) is achieved by several complementary language features.

For example, C++ has strong type checking to catch incorrectly formed function calls and incorrect use of pointers at compile time. This not only eliminates a class of runtime errors but also provides the basis for overloading functions with compile-time binding.

Also, C++ supports abstract data types with optional data hiding, member functions for controlled access, and automatic, programmer-defined instance initialization and destruction. This facilitates a more modular style of programming than C allows, with attendant improvements in reliability, maintainability, and reusability.

Binding function names to executable code can be postponed to runtime where necessary, allowing object-oriented programming to be combined with more conventional styles. Unlike most other object-oriented languages, C++ does not sacrifice compile-time type checking or impose a runtime performance penalty.

Also, the review had two incorrect or misleading impressions. First, C++ has no features specifically for windows. What it does have is a well-integrated set of features that let programmers cope with complexity, giving them the power, scope, and capability to build and interface to all kinds of complex software systems — including window managers. In this, C++ goes far beyond C. Second, C++ is already more widely used outside AT&T than within it and is used more in traditional software development than in “hard-core computer science environments,” contrary to the reviewers’ assertions.

The C++ Programming Language is aimed at people who want to write better programs now, who are looking for more efficient ways to construct programs, or who are simply interested in programming languages. I think these people will find the book interesting and useful.

Jonathan E. Shopiro
AT&T Bell Laboratories
Murray Hill, N.J.

Distressed by reference limit

To the editor:

I recently learned, with some distress, of the IEEE Software policy that limits submitted articles to 10 references. As an editor myself, I feel strongly that editors should do everything possible to encourage computer scientists to cite each other’s work (when it is genuinely related).

Computer science is a young field that suffers severely from the “not invented here” syndrome. Unlike their colleagues in more established fields such as mathematics and physics, computer scientists do not have a generally acknowledged set of intellectual leaders or landmark contemporary advances.

Instead, we have many competing traditions and a great deal of commercial hype. As a result, our representation and effectiveness at the national level is greatly diminished (for example, in the National Academies of

Correction

The label “Guest Editor’s Introduction” was inadvertently left off Will Tracz’s article “Reusability Comes of Age” on pp. 6-8 in the July issue. We regret the omission.

September 1987
migrating from assembly languages to higher order languages. It is now a major stimulus for developing and using very high-level languages and fourth-generation languages to reduce software costs.

5. Variations between people account for the biggest differences in software productivity.

Studies of large projects have shown that 90th-percentile teams of software people typically outproduce 15th-percentile teams by factors of four to five. Studies of individual programmers have shown productivity ranges of up to 26:1. The moral: Do everything you can to get the best people working on your project.

6. The overall ratio of computer software to hardware costs has gone from 15:85 in 1955 to 85:15 in 1985, and it is still growing. This relationship has done more than anything else to focus management attention and resources on improving the software process.

7. Only about 15 percent of software product-development effort is devoted to programming. In the early days, there was a 40-20-40 rule: 40 percent of the development effort for analysis and design, 20 percent for programming, and 40 percent for integration and test. Now, the best project practices achieve a 60-15-25 distribution. Overall, this relationship has been very effective in getting industrial practice to treat software product development as more than just programming.

8. Software systems and software products each typically cost three times as much per instruction to fully develop as does an individual software program. Software-system products cost nine times as much.

A software system contains many software modules written by different people. A software-system product is such a system that is released for external use. The discovery of this cost-tripling relationship has saved many people from unrealistically extrapolating their personal programming productivity experience into unachievable budgets and schedules for software-system products.

9. Walkthroughs catch 60 percent of the errors.

The structured walkthrough (software inspection) has been the most cost-effective technique to date for eliminating software errors. It also has significant side benefits in team building and in ensuring backup knowledge if a designer or programmer leaves the project.

I had a hard time picking number 10. I ended up with a composite choice:

10. Many software phenomena follow a Pareto distribution: 80 percent of the contribution comes from 20 percent of the contributors.

Knowing this can help a project focus on the 20 percent of the subset that provides 80 percent of the value for improvement. Some examples:

- 20 percent of the modules contribute 80 percent of the cost,
- 20 percent of the modules contribute 80 percent of the errors (not necessarily the same ones),
- 20 percent of the errors consume 80 percent of the cost to fix,
- 20 percent of the modules consume 80 percent of the execution time, and
- 20 percent of the tools experience 80 percent of the tool usage.

I think it has been a strong credit to the software metrics field that it has been able to determine and corroborate these and many other useful software metric relationships. And there are many useful new ones coming along. I look forward to reading about them in this department.

---

**Share your experience**

Whether your experience tends to confirm or refute the opinions expressed here, we encourage you to share that experience with us. We welcome short submissions (less than 750 words) that cover practical experiences, factors influencing productivity and quality, attempts to overcome some of the weaknesses in certain metrics, risks in using imperfect measures, and the like. We welcome both specific and general discussions of productivity and quality.

---

**SOFTWARE LETTERS**

Continued from p. 5

Science and Engineering), and research in our field is severely underfunded.

Although a liberal citation policy cannot solve such broad problems by itself, encouraging authors to cite related, historically significant research will help break down the narrow perspectives to which many of us are bound. In addition, it will encourage scholarship and intellectual honesty among authors. Also, having comprehensive bibliographies will greatly help those who want to learn more about the subject in an article.

It seems to me that increasing IEEE Software's citation limit could yield significant benefits to authors, readers, and our field as a whole.

Joseph A. Giguen
Program Manager
Computer Science Laboratory
SRI International
Menlo Park, Calif.

---

**Comments misread**

To the editor:

Douglas Schuler ("Real-World Coverage," Letters, July, p. 5) has misread my comment on SDI coverage ("Wasted Space?" Letters, May, p. 3). I merely proposed that the space normally devoted to the umpteenth rehash of arguments, whether pro or con, be severely reduced. Never did I ask that "topics be excluded" or suggest that anyone "disregard real-world implications."

My point was that these important issues could indeed be covered, minus the customary hot air, in about a third the space normally used. A perusal of the letter from Frederick Seitz ("SDI Advocate's View," Letters, July, p. 5) confirms this figure.

If I exist "in a vacuum," that analogy should be extended to say that many others have their heads in pressure cookers — they are always steamed and can't seem to get out anything intelligible.

As soon as new information on SDI is reported, I will be the first to acknowledge its newsworthiness.

John D. Wolf
Programmer
SCS, Inc.
Fort Wayne, Ind.

---

We welcome your letters. Send them to Letters Editor, IEEE Software, 10662 Los Vaqueros Cir., Los Alamitos, CA 90720. All submissions are subject to editing for style, length, and clarity.