Reinventing Reinvention

Marc Lauritsen
From *The Capstone Letter*, Jan/Feb 1991

Until recently CAPS has been the province of a relatively small band of authors. Dozens of law students in Provo and Cambridge may have learned the basics and written systems, but only about ten of us have been authoring CAPS systems for more than a year. I understand that the number of serious CAPS authors at present may soon reach a hundred. In the next few years, it could exceed a thousand.

With this anticipated growth in our ranks, it's natural to ask: How can we help each other? What can we learn from each other? What wheels can we avoid reinventing?

Let me begin with a few preliminary observations.

The Talent Spectrum. Some practice system authors start out with a strong grounding in programming concepts and methods; others encounter those concepts and methods only through the prism of a tool like CAPS/Author. While it is difficult to generalize, some advantages and disadvantages seem to be tied to each background.

Previous programmers often come to CAPS with good instincts about system design and pre-formed intuitions about phenomena like repeat structures. On the other hand, they need to watch out for being wedded to programming modes that are inappropriate to the CAPS context, and they can have difficulty understanding the problems and appreciating the ideas of non-programmers.

Non-programmers bring a freshness of mind and enthusiasm to the authoring process. They manage to do some things they haven't yet learned are impossible. On the other hand, non-programmers have a hard time separating out what is great about CAPS from the wonders of modern software generally, and they may lack an appreciation of the importance of standards, code documentation, testing, and maintenance.

Many of us are in the situation of having lost our non-programmer innocence without quite having achieved the true grit status of working comfortably with, say, assembly code. In my case, a developing proficiency in CAPS has strengthened interests in software engineering, artificial intelligence, human-computer interface, and related subjects.

Happily, there is no apparent correlation between programming background (or lack thereof) and authoring success. I've seen representatives from each category of author do miserably and excellently. The determinative factors lie much deeper. Indeed, authoring prowess, like general intelligence, is hardly measured on a uni-dimensional scale. There are dimensions of creativity, elegance, speed, efficiency, consistency, parsimony, etc. There are many rooms in

the mansion of authoring excellence.

The Programming Community. CAPS authoring, of course, *is* a form of programming, albeit on a level even more removed from machine language than languages like C, Basic, and Pascal. And most of us can readily recognize that the knowledge, methods, and insights gained by "traditional" programmers through decades of programming practice and information science research are directly applicable to the work we do with CAPS. Nonetheless, I fear we may have too little traffic with these subjects.

We as a small programming community need to better recognize the commonalities we have with the software engineering world generally. We should not approach questions of development methodology, testing, and maintenance as though there had not already been forty years of international experience in such matters.

Few of us can afford to follow even a representative sample of the massive literature in the computer science field, or attend even a few of the dozens of major conferences on computer science each year. But we all gain from paying some attention to the software engineering community. In addition, we should be willing to make our own contributions to that larger community.

Now is a particularly appropriate time to draw on these connections. The programming world, recognizing its own heritage of needless reinvention, is abuzz with talk of new paradigms and techniques. These currently include object-oriented design, hypertext, interface builders, and reusable software components. Most of these ideas have been discussed and developed for well over a decade. Interestingly enough, CAPS incorporated modular programming concepts and hypertext features long before they became fashionable.

Sharing Ideas on Legal Informatics. While legal informatics has yet to congeal as a mature discipline, there are pockets of organization. The artificial intelligence and law community now has biennial conferences, a journal, and an international organization. The American Bar Association's computer-related interest groups link thousands of technophilic lawyers with newsletters, meetings, and on-line conferences. The legal education world has the Law and Computers Section in the American Association of Law Schools, the Center for Computer Aided Legal Instruction, and over twenty similar centers and consortia internationally. There is a great deal of information of benefit to CAPS authors being shared in these arenas.

The CAPS/Author Community. CAPS/Author (which I increasingly appreciate as by no means limited to the law office context) is not doing too badly. It has this specialized newsletter, an upcoming user conference, and at least one users' group -- the Boston-area CAPS Users Group (BACUG) that Cliff Jones and I organized last year. Revisions in our practices and in CAPS itself will be necessary, though, to reap the benefits of the emerging software development trends.

We should, for example, work to promote both voluntary and market-based libraries of CAPS elements, workfiles, and systems. These can serve as exemplars from which we can learn, and

as reusable objects we can copy. Successive tinkering with such objects should help advance the state of the art.

And while it is a lot easier now than it used to be to import parts of one workfile into another, I would like to see Capsoft support such things as "universal" elements that can be referenced by more than one workfile. (Scrivener, for instance, allows systems to incorporate external models by reference.) Updates and improvements to such elements would then benefit all workfiles using them. (It might be useful to think of this as vaguely analogous to the three-dimensional worksheets now being supported by most spreadsheet programs.) Non-trivial technical and managerial problems will of course have to be solved before this blurring of workfile boundaries can take place.

Wheel reinvention and redundant research are not without their advantages. Diversity, even eccentricity, promotes creativity. We wouldn't want practice system software to become rutted in sub-optimal procedures and techniques. The time has come, though, for us to widen the channels of communication -- among ourselves and with other software developers.

We are in a business devoted to promoting the systematization of law practice through appropriate technology. We espouse the value of form libraries, precedent files, and document management. We advise our clients not to needlessly re-do work that has already been done well. The same admonition applies no less to our own activities.

So let's not reinvent reinvention. There are too many good things worth rediscovering, and new things worth inventing.