Editor: Donald J. Reifer ■ Reifer Consultants ■ d.reifer@ieee.org

Software Engineering and the Law

John Cosgrove, P.E.

For good or otherwise, the legal system has discovered the world of computers and its practitioners. Anyone opening a daily newspaper knows that litigation involving computers and software has exploded in recent years. On balance, the net effect of this attention might be positive because it gives practitioners an economic incentive to improve the way we work. Indeed, lawyers might well be the ones who provide the incentives for realistic contractual commitments, worst-case software engineering development practices, and a total organizational commitment to quality. Something similar happened to the US automobile industry, benefiting both carmakers and the public.

The threat

Tom DeMarco and Tim Lister estimate that "costs of litigation are rising faster than any other aspect of software development," and "[l]itigation costs are … a larger component than coding."[1] The legal system has not overlooked computing's pervasive presence in every aspect of society. In the recent flood of computer-related litigation, software forensic consulting in particular has multiplied in recent years, as DeMarco and Lister also noted. Usually, this involves disputes over computer projects and contracts but often requires an expert opinion in unlikely matters. Recent forensic clients have included a divorce dispute needing an economic evaluation of a software product, a wrongful termination involving computer system crashes, production of evidence of gambling during business hours, and academic plagiarism charges. Resolving all these disputes required computer expertise.

More important are the computer-related issues that threaten our economic structure and our citizens' health and safety. These threats have been with us for some time, but only recently has the legal system identified the litigation potential of computers and software.

Why software is different

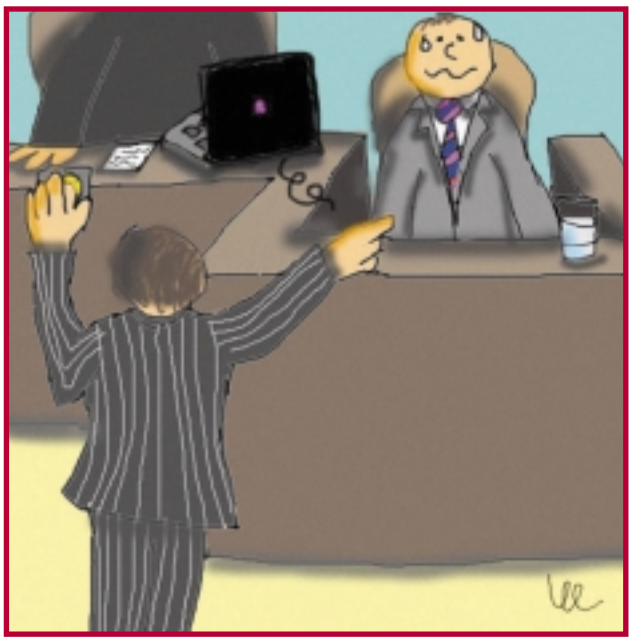
Most engineered systems start with comprehensive plans and specifications. Few software-intensive systems do!

This simple fact sets the stage for most of the issues leading to litigation. In fact, it is usually impossible to completely define most practical software systems. Watts Humphrey stated the dilemma: "With software the challenge is to balance the unknowable nature of the requirements with the business need for a firm contractual relationship."[2] Thus, there is no clear answer to the inevitable legal ambiguities. Both parties must learn to live with these ambiguities and cooperatively resolve each issue in a timely manner. When this understanding breaks down, litigation results, and the ultimate resolution is costly for both parties. DeMarco and Lister titled their article "Both Sides Always Lose: Litigation of Software-Intensive Contracts." The challenge as a software professional is to steer the parties away from this disastrous state.

0740-7459/01/$10.00 © 2001 IEEE

Explaining the unexplainable

As the wit said of computer-intensive technical claims: "All the parties are lying, but none of them knows it." That's doubly true of legal discourse involving computers. People have become so accustomed to asserting the most unsupportable conclusions from computer "facts" that they come to believe that almost anything can be true sometimes, so they might as well claim it. Because the complexity is usually very high, it is exceedingly difficult to "prove" any assertion false in a typical legal proceeding.



What to do

There are no guarantees, but if the record of the system development process shows "all reasonable steps," this is the best defense possible. Even though a well-documented process is no guarantee of quality, high quality and consistent results are almost always a result of a well-conceived, and usually well-documented, process. At least, the accusation of negligence is unlikely to hold if this is done. Also, the performance of the steps should be recorded. I'll describe the method of defining the reasonable steps in a moment.

Avoiding and surviving litigation

Several years ago, Bud Lawson proposed a method to define the engineering processes used to develop software-intensive, safety-critical systems. Simply stated, the method "assumes—a priori—that legal action has been brought against them for the product that they are about to produce."[3] Then, "all reasonable steps" must be present in the engineering activities to defend against the action. DeMarco and Lister suggest a similar strategy in that "[t]he things you do to win a litigation … also are … the principal things that you should do to avoid litigation."

Simply stated, good engineering in the best sense is the best legal defense. Typically, a software development team's culture is seldom driven by practices that a lawyer could defend as "all reasonable steps" in court. Real-life projects are defined by needs that are often independent of any achievable means. Humphrey's *Why Software Organizations Are Chaotic* describes a project: "the schedule … represented what was needed and had nothing to do with an achievable plan to make it work."[2] Even if a software-development team meets the schedule, it has made so many compromises that the quality is usually unacceptable, establishing a different basis for litigation. The solution is to insist on achievable expectations that enable the team to engineer the system according to the "all reasonable steps" principle.

As one approach, software developers might apply worst-case design principles to their development projects. Litigation has stimulated many manufacturers to apply worst-case design principles to their engineering practices. Thus, car manufacturers can defend themselves by citing the extensive research and testing they've undertaken to validate their designs before product release. Although not yet perfect, major improvements in safety and reliability have resulted. Software engineering likely will soon feel the same pressure.

As the wit said of computer-intensive technical claims: "All the parties are lying, but none of them knows it."

Managing expectations

Humphrey also expressed a cultural weakness concerning unrealistic expectations: "[D]irected by top management to run a mile in two minutes, what should they do? … many programmers start looking for their running shoes."[2] As long as this response continues, "reasonable" cannot often be truthfully applied to software development.

A recent case involved a customer who expected delivery of a "state-of-the-art" financial system in six months or less. The customer knew that the undertaking was unprecedented (never before accomplished), large (over 500K lines of code), critical (mistakes risking millions of dollars a day), and ill-defined (tens of communication interfaces changing constantly). Even so, the buyer terminated the contract and sued the supplier when it could not meet those expectations. The expert's role was to explain the reality of what the opposing side expected and the implications of the constant changes that the customer imposed on the developer during the project's life. Soon afterwards, the case settled. These types of expectations are not as rare as they should be. Many argue that some lack of realism is the norm in software development.

It is safe to say that, when unrealistic expectations are left alone, litigation will likely follow. Software developers avoid the issue because the

process of educating the customer is always painful and often fatal to keeping the contract going. The alternative—litigation—might be worse. The only solution is the painful process of confronting each perception in a patient, orderly way and documenting the mutual understanding or unresolved questions. It is only "good engineering" to insist on defining a project in achievable terms.

Warning

Litigation—potential or otherwise—involving computers and software is clearly going to become an increasingly important part of the computer professional's life. Consequently, we need to change the way we conduct the business of computers—or risk becoming part of an endless lose-lose litigation scenario. Actually, much of this change is good—good engineering practices, more reality, painful honesty with bosses and clients, and so forth—will benefit everyone in the long run. Finally, all software professionals must accept the obligation to always apply principles of worst-case engineering—in such common use by other engineering disciplines—or the legal profession will make it excruciatingly clear why the computer profession should have done so!

References

1. T. DeMarco and T. Lister, "Both Sides Always Lose: Litigation of Software-Intensive Contracts," *CrossTalk*, vol. 13, no. 2, Feb. 2000, pp. 4–6 ; www.stsc.hill.af.mil/Crosstalk/2000/feb/demarco.asp.
2. W. Humphrey, *Managing the Software Process*, Addison Wesley, Reading, Mass., 1990, p. 59.
3. H.W. Lawson, "An Assessment Methodology for Safety Critical Systems," contact the author at bud@damek.kth.se.

John Cosgrove, P.E., has been a software engineer for over 40 years and a self-employed consultant for more than 30. His specialties include forensic engineering, project management, software architecture, real-time critical systems, and hardware–software interfaces. He has extensive experience with aviation computer systems, including development of aircraft navigation systems and communication devices. He has a Masters of Engineering from UCLA and a BSEE from Loyola Marymount, Los Angeles. He is a California-registered Professional Electrical Engineer, a senior member of the IEEE Computer Society, and a member of ACM and the National Society of Professional Engineers. Contact him at Cosgrove Computer Systems, Inc., 7411 Earldom Ave., Playa del Rey, CA 90293-8058; jcosgrove@computer.org.