

Software Engineering and Litigation

Cosgrove Computer Systems Inc.
Software Engineering
7411 Earldom Avenue
Playa del Rey, California 90293-8058
(310) 823-9448, JCosgrove@computer.org

Introduction

Litigation involving computers and software has exploded. For good or otherwise, the legal system has discovered the world of computers and its practitioners. On balance, the net effect of this attention may be positive in that it provides the practitioners with an economic incentive to improve performance. In other words, the lawyers may 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 like this happened to the US automobile industry, and both the makers and users have benefited.

In the software engineering profession, *forensic engineering* refers to professional services associated with the legal system. In practice, this has two meanings. The first is the engineering support provided to the legal system, usually as an expert witness or consultant in litigation. The second meaning is the concern that all software professionals should have for the forensic (i.e., litigation potential) implications of their work. The role of the software engineering forensics expert, the implications for developers, and a proposed means for survival in a litigation-intensive computer world constitute the core issues. Since this is a new field, the use of disguised examples from actual cases is a primary basis for the explanations.

Background

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.” (reference 1) The pervasive nature of computers in every aspect of society has been noticed by the legal system. There has been a flood of litigation involving computers and software. In fact, software/computer forensic consulting has multiplied as also noted in DeMarco’s article. Usually this involves disputes over computer projects and contracts but often requires an expert opinion in unlikely matters. 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 dishonesty charges of plagiarism. Resolution of all of these disputes required computer expertise.

More important are the computer-related issues that threaten the economics of organizations and the health and safety of people in their day-to-day lives. This has been the case for some time, but only since the late 1990s has the legal system identified the litigation potential of computers and software.

TYPICAL ISSUES IN LITIGATION

Most engineered systems are defined by comprehensive plans and specifications prior to startup. Few software-intensive systems are! 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. Humphrey (reference 2, p.26) 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.” Thus, there is no clear answer to the inevitable legal ambiguities. Both parties are obliged to learn to live with these ambiguities and cooperatively resolve each issue in a timely manner. When this 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.

Project Disputes

One class of disputes involves conflicts between the parties as to what was agreed on -- including performance, look-and-feel, compatibility, schedule of features, interfaces and mutual responsibilities.

Software Engineering and Litigation

Recalling the typical state of the contractual definition at the start, whenever there is a breakdown in cooperative behavior between the parties, the resulting lawsuit is extremely difficult to litigate.

Contractual or Requirements Dispute

Often the dispute is not about specific contractual details but about mutual responsibilities when there are no applicable contractual provisions. The system is not performing, but each disputant claims that he has fulfilled his responsibilities or that the other party has kept him from doing so.

The Water System. One recent dispute involved a large water district, which was responsible for storage, filtration, treatment and distribution over a geographic extent of hundreds of square miles and several small municipalities. Several computers were distributed over this area, and they controlled and monitored the water in its various states. These computers were linked using a private network running a commercial product called a *supervisory control and data acquisition system* (SCADAS). The commercial product required considerable tailoring and system integration to adapt to the client's complex system topology. This tailoring had to be further integrated with the processing algorithms specified by the water district. The project was an eight- to nine-figure cost over several years with the computer portion in the seven-figure range.

The system functioned poorly, with a number of obscure, random, and intermittent symptoms. The electrical contractor was driven into bankruptcy from the effects of penalty clauses among other things. The supplier of the SCADAS commercial product was being sued for damages due to the system performance despite the fact that its total contractual revenue was less than six figures.

The real culprit turned out to be the chain of contractual responsibilities that starved the one contractor capable of correcting the problems caused by the now-bankrupt electrical contractor. The electrical contractor was not proficient in computer technology, having been hired for its electrical power experience. The computer configurations, the network and computer installation, the system testing, etc. had not been executed correctly with the result that the entire complex was fragile and not maintainable. However, the worst problem was that the electrical contractor was not able to continue payments to its suppliers that included the engineering design company responsible for tailoring and integrating the SCADAS. This critical contractor had effectively stopped work because of nonpayment. The water district refused to amend the contracting procedures to channel payments directly to these SCADAS engineers, insisting that the contractual obligations were paramount. The eventual result was that a very expensive settlement was negotiated to no party's advantage and did not address the system problems. New contractors were not able to correct the system's problems in a timely manner, and it continued to function at a fraction of its potential.

Cost/Schedule Overrun

Most software developers have long been told that they must meet impossible schedules and seldom take these mandates seriously (see "Managing Expectations" below). Thus, when a contract states that "time is of the essence," it is often viewed as just another impossible schedule, as noted by Humphrey. This is also true for early cost estimates that are often unrealistic. He explains that our culture has shown us that attempting to challenge these mandates may be unwelcome. Some projects really are dependent on a hard schedule and/or cost that cannot be compromised for important business reasons. If the supplier does not understand when this is true, the result can be litigation as this example demonstrates.

The School System. One case involved an educational product whose primary end users were public school systems. This type of customer had a fixed schedule for submission of working prototypes for evaluation before the school-year budget cycle. The potential unit cost also had a direct impact on the market potential because of the large volume needs and limited resources of this type of customer. Additionally, the customer's environment required a very robust and reliable design for obvious reasons. If either the cost or the schedule targets were missed, the business impact would be substantial. The supplier did not take these conditions seriously and failed at both while continuing to insist that the contractual volume commitments be honored, even at the higher price and missed schedule. The purchaser also faced major costs for reworking of their educational software products because the supplier compromised

Software Engineering and Litigation

important functional commitments. The contract was canceled primarily because of the missed schedule and cost targets, which changed the business potential to such an extent, that it was no longer worthwhile. The supplier still sued for breach of contract claiming that it had substantially performed on its part.

The case was finally settled with both parties absorbing their considerable losses and litigation expenses when email records showed that the supplier continually misrepresented the project's true status. The supplier's senior management was aware of the major technical shortcomings and knew that the cost and schedule commitments were impossible for most of the project's lifetime. Despite this awareness, they continued to assure the customer in writing that all was well. DeMarco and Lister describe a similar experience in the "Golly, How the Truth Will Out" section.

Unacceptable Performance or Quality

Fielding Buggy Systems. A dispute was triggered by a need for reliability that was poorly understood by the supplier. The senior executive responsible for the product line had prior experience in an industry that was primarily brand/image-driven. His strategy was to emphasize consumer promotion and aggressive market share penetration. His policy was to meet the shipment schedule and volume commitments in the marketing plan, regardless of the technical readiness of the systems. Even after early installation field reports indicated major reliability issues, he maintained the original, aggressive installation schedule at all costs, continuing to install known defective systems at a high rate. The customer was experiencing a major backlash from its installations because of the poor reliability and demanded that shipments be stopped until the problems were corrected.

When the supplier demanded that the customer continue to meet the volume obligations according to the original agreement, despite the admitted reliability problems, the purchaser canceled the agreement. The supplier then sued for breach of contract, ignoring the issue of implied performance in a contractual agreement that was silent on such obligations. The substantial business losses were compounded by the supplier's ignoring the importance of acceptable reliability, explicitly stated or not. Analysis of the system discovered serious design errors involving memory leaks leading to frequent system crashes. Despite this, the supplier continued to maintain that the system worked adequately for its purpose, and the litigation dragged on. The difficulty of establishing what "adequate reliability" meant was used to delay resolution of the issues as the costs to both parties climbed into seven figures.

Disputes Involving Computers

Explaining the Unexplainable

A humorous description of computer-intensive technical claims is that all of the parties are lying, but none of them know it. This seems to be particularly true of legal discourse involving computers. People have become so accustomed to being able to assert the most unsupportable conclusions from computer "facts," that they come to believe that almost anything can be true sometimes, so they may as well claim it. Since the complexity is usually very high, it is very difficult to "prove" that any assertion is false in a typical legal proceeding.

Dealing with Complexity. Sometimes the parties seem to deliberately focus on the most complex technical issues in order to obscure the real facts. In this way, almost any assertion might work, since few would be able to follow the facts and allegations. In order to avoid the trap of defending against assertions involving complex explanations, it is necessary to create simplified engineering analogies and respond in a common sense way. Often a brief technical counterargument to the complex assertion can be followed by an assertion on a related point that is described in simplified, analogous terms. This establishes solid credentials and questions the accuracy of the opposition. More importantly, the issues before the court are being clarified whereas the opposing side is adding to the confusion.

Missing Data. A recent example of unsupportable assertions was an opposing expert's formal declaration that the banding of the file allocation map on the internal drives of some business computers at issue was absolute evidence that critical business data had been maliciously deleted. Even though the "expert" was likely sincere in his assertion, he was unfamiliar with the operation of the disk optimizer

Software Engineering and Litigation

utility, which was regularly used on the drives in question. The optimization was the true cause of the banding. Explaining and defending the theory and operation of a disk optimizer to the legal teams on both sides serve as a perfect example of why mistaken assertions like these often go unchallenged.

Critical Evidence as Computer Resident

This is becoming more common for the simple reason that most engineering data, communications, and other records are kept on various computers -- either personal, business or both. Often, the fact that computer records are missing is the issue, as noted above. Also, creation/deletion/modification/last-access times to various files are often critical.

Fragments of Deleted Files. File fragments were found that contained evidence of business communications involved in establishing a new, competing business -- while still in the employ of the original employer. The employer's computer had been used for these communications, but the individual testified that he had not used the computer in this manner. Discovering the fragments that exposed him was critical to resolution of the case in favor of the client. Note that the court accepted the expert's testimony that he had actually found the data on the subject computer and that certain conclusions followed. It was critical that the expert's objectivity was maintained, or the evidence would have been discredited.

Computer-Related Human or Economic Losses

Predictable Economic Exposure. The classic example of economic loss involving computer systems is the Y2K date design error. This was a completely predictable error with the unique characteristic of a precise future date when it would become active. As a result, a large number of organizations decided on a shortsighted policy of avoiding any correction cost (either as an increment to work in progress or maintenance). The apparent reasoning was that the deciding managers would no longer be responsible for those operations in the future and could avoid blame for the increased costs. This is a typical, although usually hidden, rationale in many organizations, but this example was impossible to hide with coverup excuses. These decisions became part of many development projects, some implemented years before Y2K. Eventually, correcting the problem could no longer be avoided, and these expenses are now issues in large scale litigations because the costs of repairs were large multiples of the costs of proper designs during the original developments. One of these involved a customer of a major consulting firm who was suing that firm for not implementing a Y2K-compliant system during a project 10 years before the Y2K. Even though the consultants informed the client properly, the extra costs and schedule impact were unacceptable at the time. When the bill finally came due, the client asserted that the consultant firm should have mandated that a compliant design be implemented. This is not the first time that a client organization tried to shift the blame to the implementers when they would not face unpleasant realities. When the potential of future litigation is faced, the advantage of dealing with unrealistic expectations early becomes clearer.

The lesson to be learned is that this type of shortsighted thinking could be subject to lawsuits when the consequences could have been arguably predictable. It is not much different from deciding on a cheaper but riskier design that results in an accident that could have been avoided. This was even more blatant because the consequences were so obvious.

Disaster Recovery Planning. Other examples include failure to provide an effective business continuity plan that provides protection/recovery in the event of equipment failure or external disasters (e.g., weather, earthquake, malicious attack). When losses occur, stockholder suits are often filed against executives for failure in fiduciary duty. Since computer systems are usually part of the backbone of nationwide businesses, these losses (and litigations) attributable to computer-related causes are increasing.

Computer-Related Business Losses. One major lawsuit concerns an agreement that involved the development of a computer system that was installed at hundreds of locations. The developer rushed the fielding of the system before it had been adequately tested. Thus, the installed systems had very low reliability, which ultimately caused the users in the field to lose confidence in the systems and return them. This resulted in seven-figure losses followed by eight-figure lawsuits. Other competitors did not make the same mistake, and similar systems are now major revenue producers. The parties involved spend their corporate money and energies on the lawsuit while others make money in the business that the original

Software Engineering and Litigation

parties pioneered. DeMarco and Lister (reference 1) described the effect of this kind of litigation. "The legal fees were staggering. The opportunity costs, ... were even worse."

Accident Liability. Opportunities for litigation are obvious in such systems as large airliners, which have millions of lines of safety-critical software. Less obvious was a database design error which caused blood donor records to be overwritten, resulting in the distribution of AIDS-tainted blood. Another example was the use of unvalidated design software in a structural engineering firm responsible for the structural analysis of major public buildings. Most of these past accidents did not lead to the software components as being causative factors. This is changing as the nature of the software component becomes more widely known. Liability insurance rates for software developers have jumped sharply to reflect this.

Pilot Error or Design Error? The influence of the software design can be subtle, but still profound in some investigations. One illustration is that of an airline that killed most aboard when one of its planes ran into a mountain. The investigation was looking at a design weakness in the navigation system that allowed an abbreviation for an en route destination to be confused with a similar abbreviation also in the system. The ambiguous abbreviation selected the wrong destination, and the navigation system steered the airplane into the mountain before the crew could correct it. The litigation had to decide whether this was a culpable design error. Neumann (reference 6, p. 23, "Jury blames computers for Cali plane crash") reported that the navigation data supplier was 17% responsible, the navigation system supplier 8%, and the airline 75%. Since the stakes were very large (159 killed), these liabilities resulted in major impacts to each of these parties. Leveson in her landmark book *Safeware* (reference 5, p. 22, 2.1, *The Role of Computers in Accidents*) states, "A relatively new breed of hazards ... has appeared. ... Some of the hazards are passive until just the right combination of circumstances arrives."

Role of the Expert in Litigation

A good place to start is an article written by an attorney/expert, Phillip Kolczynski, who specializes in aviation. His article, "How to Be a Successful Expert Witness" (reference 3), treats the subject in language comfortable to technical readers and gives useful guidelines for getting started as an expert in any specialty. The information would have helped this expert avoid some earlier mistakes and is also helpful to the experienced forensic engineer.

Not an Advocate. The expert's role is different from those of the other members of the legal team. Even though the client pays the expert, there should be no hint of advocacy, only an objective evaluation of the evidence and a resulting opinion. Some compare this role to that of the CPA who is paid by the client but is expected to prepare tax information from user-supplied financial data in strict adherence to the tax laws. The expert is the only one who can give an opinion as evidence, and, if that opinion appears to be leaning toward advocacy, it will be discredited. An important point is that the expert has no privilege, and all material and conversations in connection with the case are subject to discovery.

Subject to Discovery. Every note, all conversations, all writings, some prior writings (papers given, etc.) must be "produced" (copies given) in their entirety. Typically, these writings are subject to interrogation during deposition and trial. The expert must have this in mind whenever any communications or writings (notes, emails, etc.) are made. Many attorneys specifically request that the expert only make rough notes and delay drafting an opinion until the later stages because they want to avoid producing the material until other aspects of the case are established. Conversations are also subject to discovery but are not as useful because the only opportunity to "discover" them is during testimony (deposition or trial), and that is usually in the later stages. Still, questions will be asked, and the line should not be crossed even in conversation.

Allowable Assistance. Some forms of direct assistance to the client's legal team are proper. One example is the preparation of questions for the attorney to pose to the opposing expert or whenever technical issues are being examined. This can be prior to the actual deposition or trial or as an advisor during the actual testimony. Since it is likely that the attorney needs the expert's understanding of the significance of technical information being examined, her or his role as an objective party is not compromised by assistance. This is proper as long as the goal is to discover the information and understand its significance. This is often a fine line to observe, but it is an invaluable role if done properly.

Software Engineering and Litigation

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 documented, process. At least, the accusation of negligence is unlikely if this is done. Also, the performance of the steps should be recorded. The method of defining the reasonable steps is described below.

Avoiding and Surviving Litigation

In 1996, Lawson (reference 4) 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.” Then “all reasonable steps” must be present in the engineering activities to defend against the action. Demarco & Lister (reference 1) suggest a similar strategy in that “[t]he things you do to win a litigation ... also are ... the principle things that you should do to avoid litigation.”

All Reasonable Steps. Simply stated, good engineering in the best sense, is the best legal defense. The typical culture of software development teams is seldom driven by practices which could be defended as “all reasonable steps” in court. Real life projects are defined by needs that are often independent of any achievable means. Humphrey (reference 2, p. 59 4.3) “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.” Even if the schedule is met, so many compromises are made that the quality is usually unacceptable, and a different basis for the litigation is established (see “Unacceptable .Performance or Quality” above). The solution is to insist on achievable expectations that enable the system to be engineered according to the “all reasonable steps” principle.

One approach is to follow a method where worst-case design principles are applied to software development. Litigation has been an important stimulus for manufacturers of products, affecting public safety, to apply worst-case design principles to their engineering practices. Thus, car manufacturers can defend themselves by citing the extensive research and testing to validate all designs before product release. Even though this is not perfect, major improvements in safety and reliability have resulted. It is likely that software engineering will soon be affected in the same manner.

Managing Expectations

Humphrey (reference 2, p.59) also expressed a cultural weakness concerning unrealistic expectations “... directed by top management to run a mile in two minutes, what should they do? ... many programmers start looking for their running shoes.” As long as this continues to be the case, “reasonable” cannot often be truthfully applied to software development.

Impossible Goals. Another case involved a customer who expected a state-of-the-art financial system to be developed in six months or less. The customer knew that it was unprecedented (never before accomplished), large (> 500 KLOC), critical (mistakes risking \$M/day), and ill-defined (tens of communication interfaces changing constantly). 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 life of the project. Soon after this, 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 is likely to follow. Avoiding the issue occurs because the process of educating the customer is always painful and often fatal to keeping the contract going. The alternative may be worse (i.e., litigation). 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.

Worst-Case Design Framework

Good engineering should always use worst-case design methods. The problem is in applying it to software engineering. One solution is to apply a set of principles that answers the “all reasonable steps” requirement.

Software Engineering and Litigation

This is described as a framework that can be applied to any type of software-intensive system, whether it is an airline navigation system or a database for business records. As noted above, both can be liability-threatening systems as can virtually any software project. One advantage of this approach is that using the framework makes it easy to explain the rationale in terms that noncomputer people can understand. This is critical in any potential legal proceeding.

Define the System States. Software can be described as being in various states -- often known as a *State Machine*. Most software is so complex that the number of states is too large to be useful if the low-level mechanisms are included. This method proposes only three high level states:

1. *Operational* – The system is operating normally, doing what it was designed to do and operating within its design limitations.
2. *Exception* – The system has been impacted by a condition outside its normal limitations and is expected to successfully deal with that condition, i.e., recover in a predictable manner.
3. *Nonoperational* – The system has been presented with one or more conditions that keep it from continuing to operate (i.e., not recoverable) but is expected to be able to transition to not operating in an orderly, (i.e., fail safe) manner to the greatest extent possible.

Note that the abovementioned method could be applicable to control-loss/crash-resistance designs for an automobile. The definitions above imply that those detailed boundary conditions, both recoverable and otherwise, are specified -- seldom done in software. Another implication is that there must be a specified response to the violation of these conditions. This does not have to involve complex issues as a later given example of a mistake in typing illustrates. Safety engineering methods such as *failure mode effects analysis* (FMEA) can be helpful in identifying the causes of violations of the boundary conditions; the challenge is to apply the results of that analysis to the software design.

Define the Behavior during State Change. This can be thought of as the “what if” part of the design. Using the states and their boundary conditions described above, what should the system do when confronted with a boundary condition violation? Note that this also might be a very useful response to questions posed in defense of a design under litigation. If there were design documents, which showed a good-faith attention to all predictable events, this would be of great help in absolving the developer of negligence even though mistakes may have been made. This applies to issues of any complexity.

For example, a common (simple) issue is the manner in which the system responds to invalid characters typed at the operator interface. Obviously, nothing typed at a user interface should ever cause the system to crash, but precisely defining valid and invalid data is critical as well as the way the system responds to it. A well-designed response would intercept the mistake in typing, give an understandable error message and give an immediate opportunity to reenter the data correctly. More complicated examples would treat losses of communication channels or devices, power transients, etc. Many systems are designed without any orderly requirements for these clearly predictable events. It would be very damaging to be forced to testify that the system corrupted valuable data or caused an accident because of a power transient or an operator typing error; both have been factors in disputes and accidents (see the airliner crash example above).

Summary

Litigation -- potential or otherwise -- involving computers and software is clearly going to be part of computer professionals' lives. The implication is that some changes are necessary in the way the business of computers is conducted -- or risk becoming part of that “lose-lose” litigation scenario. Actually, much of this change is good (good engineering practices, more reality, being painfully honest with the boss or client, etc.)-- will benefit all in the long run. Finally, all software professionals must accept the obligation to always apply principles of “worst-case engineering” -- used by other engineering disciplines -- or the legal profession will apply its professional skills to make clear why the computer profession should have done so.

Software Engineering and Litigation

References

1. T. DeMarco, and T. Lister, "Both Sides Always Lose: Litigation of Software-Intensive Contracts, " *CrossTalk Magazine*, February 2000. Available at www.stsc.hill.af.mil/Crosstalk/2000/feb/demarco.asp.
2. W. Humphrey, *Managing the Software Process*, Addison Wesley, Reading MA, 1990.
3. P. J. Kolczynski, *How To Be A Successful Expert Witness*. Available at <http://aviationlawcorp.com/content/successfulexpert.html>
4. H. W. Lawson, "An Assessment Methodology for Safety Critical Systems, " (unpublished paper). Available at Bud@damk.kth.se.
5. N. G. Leveson, *Safeware – System Safety and Computers*, Reading MA, Addison-Wesley, 1995.
6. P. G. Neumann, "Risks to the Public in Computers and Related Systems", *Software Engineering Notes*, *ACM SIGSOFT*, January (2001).