SOFTWARE LIABILITY

Cem Kaner, J.D., Ph.D.
Copyright © 1997. All rights reserved.

Note: This paper is based on talks of mine at recent meetings of the Association for Software Quality's Software Division and the Pacific Northwest Software Quality Conference. The talks surveyed software liability in general and focused on a few specific issues. I've edited the talks significantly because they restate some material that you've seen in this magazine already. If you don't have those articles handy, check my website, `www.badsoftware.com`.

W. Edwards Deming is one of my heroes. I enjoyed and agreed with *almost* everything that I've read of his. But in one respect, I flatly disagree. In *Out of the Crisis*, Deming named seven "deadly diseases." Number 7 was “Excessive costs of liability, swelled by lawyers that work on contingency fees.” (Deming, 1986, p. 98).

Software quality is often abysmally low and we are facing serious customer dissatisfaction in the mass market (see Kaner, 1997e; Kaner & Pels, 1997). Software publishers routinely ship products with known defects, sometimes very serious defects. The law puts pressure on companies who don't care about their customers. It empowers quality adocates. I became a lawyer because I think that liability for bad quality is part of the cure, not one of the diseases.

Life is more complex than either viewpoint. It's useful to think of the civil liability system as a societal risk management system. It reflects a complex set of tradeoffs and it evolves constantly.

Risk Management and Liability

Let's think about risk. Suppose you buy a product or service and something bad happens. Somebody gets hurt or loses money. Who should pay? How much? Why?

The Fault-Based Approach

If the product was defective, or the service was performed incompetently, there's natural justice in saying that the seller should pay. This is a *fault-based* approach to liability.

First problem with the fault-based approach: How do we define "defective"? The word is surprisingly slippery.

I ventured a definition for serious defects in Kaner (1997a). I think the approach works, but it runs several pages. It explores several relationships between buyers and sellers, and it still leaves a lot of room for judgment and argument. More recently, I was asked to come up with a relatively short definition of "defect" (serious or not). After several rounds of discussion, I'm stalled.

I won't explore the nuances of the definitional discussions here. Instead, here's a simplification that makes the legal problem clear. Suppose we define a defect as failure to meet the specification. What happens when the program does something obviously bad (crashes your hard disk) that was never covered in the spec? Surely, the law shouldn't classify this as non-defective. On the other hand, suppose we define a defect as any aspect of the program that makes it unfit for use. Unfit for who? What use? When? And what is it about the program that makes it unfit? If a customer specified an impossibly complex user interface, and the seller built a program that matches that spec, is it the seller's fault if the program is too hard to use? Under one definition, the law will sometimes fail to

compensate buyers of products that are genuinely, seriously defective. Under the other definition, the law will sometimes force sellers to pay buyers even when the product is not defective at all.

This is a classic problem in classification systems. A decision rule that is less complex than the situation being classified will make mistakes. Sometimes buyers will lose when they should win. Sometimes sellers will lose. Both sides will have great stories of unfairness to print in the newspapers.

Second problem with the fault-based approach: We don't know how to define "competence" when we're talking about software development or software testing services. I'll come back to this later, in the discussion of professional liability.

Third problem: I don't know how to make a software product that has zero defects. Despite results that show we can dramatically reduce the number of *coding* errors (Ferguson, Humphrey, Khajenoori, Macke, & Matuya, 1997; Humphrey, 1997), I don't think anyone else knows how to make zero-defect software either. If we create too much pressure on software developers to make perfect products, they'll all go bankrupt and the industry will go away.

In sum, finding fault has appeal, but it has its limits as a basis for liability.

Technological Risk Management

It makes sense to put legal pressure on companies to improve their products because they can do it relatively (relative to customers) cheaply. In a mass market product, a defect that occasionally results in lost data might not cost individual customers very much, but if you total up all the costs, it would probably cost the company a great deal less to fix the bug than the total cost to customers. (Among lawyers, this is called the principle of the "least cost avoider." You put the burden of managing a risk on the person who can manage it most cheaply.)

I call this *technological risk management*--because we are managing the risk of losses by driving technology. Losses and lawsuits are less likely when companies make better products, advertise them more honestly, and warn customers of potential hazards and potential failures more effectively.

At our current stage of development in the software industry, I think that an emphasis on technological risk management is entirely appropriate. We save too many nickels in ways that we know will cost our customers dollars.

However, we should understand that the technological approach is paternalistic. The legal system decides for you what risks companies and customers can take. This drives schedules and costs and the range of products that are available on the market.

The technological approach makes obvious sense when we're dealing with products like the Pinto, which had a deadly defect that could have been fixed for $11 per car. It's entirely appropriate whenever manufacturers will spend *significantly* less to fix a problem than the social cost of that problem. But over time, this approach gets pushed at less and less severe problems. In the extreme, we risk ending up with a system that imposes huge direct and indirect taxes on us all in order to develop products that will protect fools from their own recklessness.

As we move in that direction, many companies and individuals find the system intolerable. Starting in the 1970's we were hearing calls for "tort reform" and a release from "oppressive regulations." The alternative is commercial risk management: let buyers and sellers make their own deals and keep the government out of it.

 Copyright © Kaner, 1997. All rights reserved.

Commercial Risk Management

This is supposed to be a free country. It *should* be possible for a buyer to say to a seller, "Please, make the product sooner, cheaper, and less reliable. I promise not to sue you."

The commercial risk management strategy involves *allocation* of risk (agreeing on who pays) rather than reduction of risk. Sellers rely on contracts and laws that make it harder for customers to sue sellers. Customers and sellers rely on insurance contracts to provide compensation when the seller or customer negligently makes or uses the product in a way that causes harm or loss.

This approach respects the freedom of people to make their own deals, without much government interference. The government role in the commercial model is to determine what agreement the parties made, and then to enforce it. (Among lawyers, this is called the principle of "freedom of contract.")

The commercial approach makes perfect sense in deals between people or businesses who actually have the power to negotiate. But over time, the principle stretches into contracts that are entirely non-negotiated. A consumer buying a Microsoft product doesn't have bargaining power.

Think about the effect of laws that ratify the shrink-wrapped "license agreements" that come with mass-market products. In mass-market agreements, we already see clauses that avoid all warranties and that eliminate liability even for significant losses caused by a defect that the publisher knew about when it shipped the product. Some of these "agreements" even ban customers from publishing magazine reviews without the permission of the publisher (such as this one, which I got with Viruscan, “The customer will not publish reviews of the product without prior written consent from McAfee.”)

Unless there is intense quality-related competition, the extreme effect of a commercial risk management strategy is a system that ensures that the more powerful person or corporation in the contract is protected if the quality is bad but that is otherwise indifferent to quality.

Without intense quality-driven competition, some companies will slide into lower quality products over time. Eventually this strategy is corporate suicide, but for a few years it can be very profitable.

Ultimately, the response to this type of system is customer anger and a push for laws and regulations that are based on notions of fault or of technological risk management.

Legal Risk Management Strategies are in Flux

Technological and commercial risk management strategies are both valid and important in modern technology-related commerce. But both present characteristic problems. The legal policy pendulum swings between them (and other approaches).

Theories of Software Liability

Software quality advocates sometimes argue that we should require companies to follow reasonable product development processes. This is a technological risk management approach, which is obvious to us because that's what we do for a living: use technology to improve products and reduce risks.

A "sound process" requirement fits within some legal theories, but not others. There are several different theories under which we can be sued. Different ones are more or less important, depending on the legal climate (i.e., depending on which legal approach to risk management is dominant at the moment).

 Copyright © Kaner, 1997. All rights reserved.

A legal "theory" is not like a scientific theory. I don't know why we use the word "theory." A legal theory is a definition of the key grounds of a lawsuit. For example, if you sue someone under a negligence theory:

- You must prove that (a) the person owed you a duty of care; (b) the person breached the duty; and (c) the breach was the cause of (d) some harm to you or your property.
- You must convince the jury that (a), (b), (c), and (d) are *all* more likely to be true than false Ties go to the defendant.
- If you prove your case, you are entitled to compensation for the full value of your injury or of the damage to your property.
- If the jury decides there is clear and convincing evidence that the defendant acted fraudulently, oppressively, maliciously, or outrageously, you can also collect punitive damages. These are to punish the defendant, not to compensate you. The amount of damages should be enough to get the defendant's attention but not enough to put it out of business. Punitive damages are rarely awarded in lawsuits--in a short course for plaintiffs' lawyers on estimating the value of a case, we were told to expect to win punitive damages in about 2% of the negligence cases that we try, and to expect small punitive damage awards in most of these cases. If a jury does assess major punitive damages, the trial court, an appellate court, and sometimes the state's supreme court all review the amount and justification of the award.

Every lawsuit is brought under a specifically stated theory, such as negligence, breach of contract, breach of warranty, etc. I provided detailed definitions of most of these theories, with examples, in Kaner, Falk, & Nguyen (1993). You can also find some of the court cases at my web site, along with more recent discussion of the law--check the course notes for my tutorial at Quality Week, 1997, at `www.badsoftware.com`.

Quality Cost Analysis

Any legal theory that involves "reasonable efforts" or "reasonable measures" should have you thinking about two things:

- We aren't just looking at a product in this case. The process used to develop the product is at least as important as the end result.
- The judge or jury are going to do a cost/benefit analysis if this type of case ever comes to trial.

We are, or should be, familiar with cost/benefit thinking, under the name of "Quality Cost Analysis" (Gryna, 1988; Campanella, 1990).

Quality cost analysis looks at four ways that a company spends money on quality: prevention, appraisal (looking for problems), internal failure costs (the company's own losses from defects, such as wasted time, lost work, and the cost of fixing bugs), and external failure costs (the cost of coping with the customer's responses to defects, such as the costs of tech support calls, refunds, lost sales, and the cost of shipping replacement products). Note that the external failure costs that we consider as costs of quality reflect the company's costs, not the customer's.

Previously (Kaner, 1996a), I pointed out that this approach sets us up to ignore the losses that our products cause our customers. That's not good, because if our customers' losses are significantly worse than our external failure costs, we risk being blindsided by unexpected litigation.

The law cares more about the customer's losses. A manufacturer's conduct is unreasonable if it would have cost less to prevent or detect and fix a defect than it costs customers to cope with it (Kaner, 1996b).

 Copyright © Kaner, 1997. All rights reserved.

Cost of quality analysis was developed by Juran as a persuasive technique. “Because the main language of [corporate management] was money, there emerged the concept of studying quality-related costs as a means of communication between the quality staff departments and the company managers” (Gryna, 1988, p. 42). You can use this approach without ever developing complex cost-tracking systems. Whenever a product has a significant problem, of any kind, it will cost the company money. Figure out which department is most likely to lose the most money as a result of this problem and ask the head of that department how serious the problem is. How much will it cost? If she thinks its important, bring her to the next product development meeting and have her explain how expensive this problem really is. There is no expensive cost-tracking system in place, but there's a lot of persuasive benefit here.

When the company's cost of external failures is less than the cost a customer will face, don't use these numbers to try to persuade management to fix the problem. The numbers aren't persuasive and they almost certainly underestimate the long term risks (litigation and lost sales). Instead, come up with some scenarios, examples that illustrate just how serious the problem will be for some customers. Make management envision the problem itself and the extent to which it will make customers unhappy or angry.

Survey of the Theories

Here's a quick look at theories under which a software developer can be sued:

- ***Criminal:*** The government sues the company for committing a criminal act, such as intentionally loading a virus on the customer's computer or otherwise tampering with the computer. For example, several years ago, Vault Corp. announced plans to release a new copy protection program that would unleash a worm that would gradually destroy your system if you illegally (in the program's opinion) copied the protected program. (see Kaner et. al., 1993 for details.) That was probably not illegal at the time, but today such a program probably would be.
- ***Intentional Tort:*** The company did something very bad, such as deliberately loading a virus onto your computer, or stealing from you, or telling false, insulting stories about you. The government might be able to sue the company under a criminal theory. You sue the company for damages (money, to be paid to you).
- ***Strict Liability:*** A product defect caused a personal injury or property damage. In this case, we look at the product's defectiveness and behavior, without thinking about the reasonableness of the process used to develop the product. No punitive damages are available. For example, suppose that the program controlling a car's brakes crashes and soon thereafter, so does the car. In a strict liability suit, we would have to prove that the program was defective, and the defect caused the accident. In a negligence suit, we also have to ask whether the manufacturer made a reasonable effort to make the brakes safe.
- ***Negligence:*** The company has a duty to take reasonable measures to make the product safe (no personal injuries or property damage), or no more unsafe than a reasonable customer would expect (skis are unsafe, but skiers understand the risk and want to buy skis anyway.) Under the right circumstances, a company can non-negligently leave a product in a dangerous condition.

  Proof of negligence can be quite difficult. No single factor will prove that a company was non-negligent. A court will consider several factors in trying to understand the level of care taken by the company (Kaner, 1996b). Kaner, Falk, & Nguyen (1993) list several factors that will probably be considered in a software negligence case, such as:

 Copyright © Kaner, 1997. All rights reserved.

- Did the company have ctual knowledge of the problem? (*No one likes harm caused by known defects.)*
- How carefully did the company perform its safety analysis? *(The wrong answer is, "Safety analysis? What safety analysis?")*
- How well designed is the program for error handling? (*The law expects safety under conditions of foreseeable misuse. 90% of industrial accidents are caused by "user errors." Manufacturers have to deal with this, not whine about dumb users.)*
- How does the company handle customer complaints? (*Jurors will sympathize with mistreated customers.)*
- What level of coverage was achieved during testing? (*There are so many different types of coverage. Using judgment is more important than slavishly achieving 100% on one type of coverage. Kaner, 1996b.)*
- Did the product design and development follow industry standards? (*In negligence, failure to follow a standard is relevant if and only if the plaintiff can show that this failure caused the harm.)*

  It's worth asking whether current industry standards, such as IEEE standards, are appropriate references. Do they realistically describe what the industry does or should do?
- What is the company's bug tracking methodology? (*Does it have one?)*
- Did the company use a consistent methodology? (*If not, how does it make tradeoffs?)*
- What is the company's actual level of intensity or depth of testing? (*Did it make a serious effort to find errors?)*
- What is its test plan? *(How did the company develop it? How do they know it's good? Did they follow it?)*
- What does the documentation say about the product? *(Does it warn people of risks? Does it lead them into unsafe uses of the product?)*

· ***Fraud:*** The company made a statement of fact (something you can prove true or false) to you. It knew when it made the statement that it was false, but it wanted you to make an economic decision (such as buying a product or not returning it) on the basis of that statement. You reasonably relied on the statement, made the desired decision, and then discovered that it was false. In the case of *Ritchie Enterprises v. Honeywell Bull* (1990), the court ruled that a customer can sue for fraud if technical support staff convinced him to keep trying to make a bad product work (perhaps talking him out of a refund), by intentionally deceiving him after the sale.

· ***Negligent Misrepresentation:*** Like fraud except that the company made a mistake. It didn't know that the statement was false when it made it. If the company had taken the care in fact-finding that a reasonable company under the circumstances would have taken, it would not have made the mistake. *Burroughs Corp. v. Hall Affiliates, Inc*. (1982) is an example of this type of case in a sales situation.You have to establish that the company owed you a duty to take care to avoid accidentally misinforming you. This is often very difficult to prove, especially if the company made a false statement about something that it was not selling to

 Copyright © Kaner, 1997. All rights reserved.

you. However, independent test labs have been successfully sued by end customers for negligently certifying the safety of a product (Kaner, 1996b).

- ***Unfair or Deceptive Trade Practice:*** The company engaged in activities that have been prohibited under the unfair and deceptive practices act that your state has adopted. For example, false advertising, or falsely stating or implying that the product has been endorsed by someone, or falsely claiming that a new upgrade will be released in a few weeks, are all deceptive trade practices. You may have to show that the company has repeatedly engaged in this misconduct--the theory may require evidence of a "practice", a pattern of misconduct, not just one bad event. You can receive a refund and repayment of your attorney fees. Some states allow additional statutory damages. For example, in Texas, a successful plaintiff can collect up to three times her actual damages. This is the law under which Compaq has recently been sued (*Johnson v. Compaq*, 1997). According to the plaintiff, Compaq sold a computer with a warranty that stated that Compaq would not charge for calls about software defects. He claims that Compaq's support staff told the plaintiff that he had to pay up to $3 per minute for all calls about software, whether they involved defects or not. Based on his observation of the AOL/Compaq message board and on other sources, the plaintiff alleged that Compaq was also refusing to provide free support to other people when they called about genuine software defects.
- ***Unfair Competition:*** The definition varies across states. For example, in California anyone can file an unfair competition suit, so long as they can prove that the company engaged in a pattern of illegal activity. In some other states, only a competitor can sue, and only for some narrower list of bad acts. In *Princeton Graphics v. NEC* (1990), Princeton successfully sued NEC for claiming that its Multisync monitor (the first one) was VGA-compatible. Princeton and NEC had the same problems with VGA, and Princeton chose not to advertise itself as VGA-compatible.
- ***FTC Enforcement:*** The Federal Trade Commission can sue companies for unfair or deceptive trade practices, unfair competition or other anti-competitive acts. Most defendants these cases without admitting liability. Recent FTC cases have been settled against Apple Computer (*In the Matter of Apple Computer*, 1996) and against the vendor of a Windows 95 optimization program that allegedly didn't provide any performance or storage benefits (*In the Matter of Syncronys Software*, 1996). Occasionally, the FTC sues over vaporware announcements that appear to be intended to mislead customers.
- ***Regulatory:*** The Food and Drug Administration, for example, requires that certain types of software be developed and tested with what the FDA considers an appropriate level of care. My understanding is that development process is important to the FDA.
- ***Breach of Contract:*** In a software transaction, the contract specifies obligations that two or more persons have to each other. (In legal terms, a "person" includes humans, corporations, and other entities that can take legally binding actions.) Contracts for non-customized products are currently governed under Article 2 (Law of Sales) of the Uniform Commercial Code (UCC). Contracts for services, including custom software, are covered under a more general law of contracts.

  UCC transactions carry implied terms. For example, products normally come with an implied warranty of merchantability (the product will be fit for ordinary use, it will conform to the claims on the packaging and in the manual, and it will pass without objection in the trade.) This reflects a basic American public policy, that some modest standards of integrity should be applied to the sales of goods. To disclaim an implied warranty of merchantability (i.e. to legally effectively say that there is no such warranty), a merchant seller must put a conspicuous disclaimer in the contract. (A company that sells software in the ordinary

course of its business would be a software merchant.) Court cases have almost always required this notice to be given to the customer in a way that makes it conspicuous (likely to be seen) before or at the time of sale (White & Summers, 1995, volume 1).

The specific validity of shrink-wrapped software warranty disclaimers that are not visible to the customer until after the sale was considered in two opinions, *Step-Saver v. Wyse Technology and The Software Link* (1991), and *Arizona Retail v. The Software Link* (1993). The courts ruled that an implied warranty comes with a product at the time of sale unless it is conspicuously disclaimed, and that a conspicuous disclaimer that is not available to the customer until after the sale is merely a proposal to modify the contract, and is not part of the contract unless the customer agrees. A recent case, *ProCD, Inc. v. Zeidenberg* (1996), appears to say that anything in a shrink-wrappred license is valid but that case didn't consider warranty disclaimers. It will be a major departure from UCC history if a court enforces a warranty disclaimer that was not available to be seen by the customer before the sale.

The UCC also says that express warranties cannot be disclaimed. An express warranty is any statement of fact (something you can prove true or false) by the seller to the buyer about the product that becomes part of the basis of the bargain. The phrase "basis of the bargain" is to be interpreted expansively. The exact rules vary from state to state, but if a reasonable customer would interpret the seller's pre- or post-sale statements as factual descriptions of the product that the customer has bought, and would be even slightly influenced by the statements in deciding whether to buy or keep the product, then you should think of them as "basis of the bargain" statements. (See Kaner, 1995a, Kaner & Pels, 1996, and the court case, *Daughtrey v. Ashe,* 1992.)

- ***Magnuson-Moss Warranty Improvement Act:*** Consumers of goods are entitled to a clear statement of the warranty's terms. For goods costing $15 or more, merchant sellers are required to make warranties available to customers before the sale. *The Code of Federal Regulations* specifies ways in which sellers can meet this requirement (such as having a binder with a copy of each product's warranty, and a conspicuous sign that informs customers where the binder is.) The Software Publishers Association's own handbook of software contracts (Smedinghoff, 1993) states that for consumer software (any off-the-shelf product that is commonly used for personal, family or household use), the Magnuson-Moss Act probably applies.

Software service providers can also be sued. A "software service provider" is a "person" that writes custom software, maintains or supports software, trains other people to use software, does software testing or certification, or enters into other contracts involving software in which a significant component of the benefit to be provided by the seller involves human labor. Service providers can be sued in many of the ways that I listed for products, above, but also for:

- ***Negligent or grossly negligent provision of services:*** The service provider has a duty of ordinary care to the customer, to make reasonable efforts to provide the service. Basically, this means that the service provider should try to do the job well and it should not make mistakes that any reasonable person would recognize as mistakes. A service provider that holds itself out as having specific expertise will have its mistakes judged by what mistakes other reasonable people with this specific expertise would have made, if they were making reasonable efforts to do a good job.
- ***Malpractice:*** This is a negligence standard, with a twist. The quality of services provided is judged against a professional standard. Mistakes or other failures in delivering service that would not have been made by an ordinary professional in the field are malpractice. The existence of professional standards makes proof of malpractice

 Copyright © Kaner, 1997. All rights reserved.

easier than proof of simple negligence. If a customer wins a malpractice case, he'll probably get more money than in a basic service provider negligence case. There is currently no theory of computer-related malpractice (Kaner, 1996d) because we have no computer-related professions, and no generally accepted standards.

Some software quality advocates are calling for professionalization. The say that software quality "engineers" should be licensed by the government and held to professional standards. If you are thinking along these lines, please consider the problem that we need a solid basis for distinguishing unacceptable from acceptable practices. Otherwise, professional liability will be a lottery: you will be sued for practices that you consider good. Here are some examples of disagreements:

- I regard the development of detailed written test scripts as an industry worst practice.
- I am not aware of any evidence that ISO 9000-3 has resulted in improved software quality and I would not recommend it to a client as a framework for improving quality.
- I believe that the reasonable practices and standards vary substantially among the several different industries that we lump together as "software." I valued my training for the ASQ-CQE because it helped me appreciate the diversity of good practices across industries. I think the CSQE's view of the software world is erroneously monolithic.

I'm not alone in these views, but many of you will disagree with me. *That's the point of these examples.* We do not have a consensus on professional practices.

One last word of caution. If a person who is not a licensed professional identifies herself to potential clients as a professional, they can sue her for malpractice as if she were a member of that profession. If you call yourself a "quality engineer" (in California) or an "engineer" (some other states), you might discover yourself at the wrong end of an engineering malpractice suit. The suit will challenge judge and jury to figure out what the professional knowledge and standards a software quality engineer would have if there was such a profession and if it had generally accepted practices. Your lawyer would make a lot of money on this case.

Uniform Commercial Code Revisions

This paper is based on my talks at ASQ (Kaner, 199h) and PNSQC. I spent a fair bit of time at those meetings on Article 2B, the proposed revisions to the Uniform Commercial Code. I've written about that proposal in this magazine (Kaner, 1996c) and in other publications that you may have read (such as Kaner, 1997f; Kaner & Lawrence, 1997). I'll write a survey of Article 2B for this magazine again as it gets closer to its final form.

At the moment, I think Article 2B is a disaster in the making. If you're interested in details, Kaner (1997g) is my best current commentary. Article 2B is a moving target, significantly revised every two months. Earlier detailed analyses appear in Kaner 1997b, 1997c, and 1997d.

The only reason that I mention 2B here is to bring us back to the issue of risk management that opened this paper. Article 2B illustrates this decade's trend, which is an almost exclusive focus on commercial risk management. 2B is so heavily biased partially because so few of us on the technology side are making ourselves available to explain technological risk management to the commercial lawyers who are drafting legislation. The same problems are showing up in the Uniform Electronic Transactions Act, which will govern electronic commerce. Lawyers need concrete examples and clear explanations of how law can be used to encourage technological improvement, or they will stick with purely commercial approaches. If the only tools that lawyers have are hammers, they will pass laws declaring that all non-hammers are nails.

 Copyright © Kaner, 1997. All rights reserved.

REFERENCES

Arizona Retail Systems, Inc. v. The Software Link (1993) Federal Supplement, *volume 831*, p. 759.

Burroughs Corp. v. Hall Affiliates, Inc. (1982) Southern Reporter, Second Series, *Volume 423*, p. 1348 (Supreme Court of Alabama).

Campanella, J. (Ed.) (1990). *Principles of Quality Costs*, 2nd Ed. ASQC Quality Press.

Daughtrey v. Ashe (1992) South Eastern Reporter, Second Series, *volume 413*, 336 (Supreme Court of Virginia).

Deming, W.E. (1986). *Out of the Crisis.* MIT Press.

Ferguson, P., Humphrey, W.S., Khajenoori, S., Macke, S., & Matuya, A. (1997) Results of Applying the Personal Software Process, *IEEE Computer*, May, 1997, p. 24-31.

Gryna, F. M. (1988) "Quality Costs" in Juran, J.M. & Gryna, F. M. (1988, 4th Ed.), *Juran's Quality Control Handbook*, McGraw-Hill.

Humphrey, W.S. (1997) *Comments on Software Quality*. Distributed to the National Conference of Commissioners on Uniform State Laws for their Annual Meeting, July 25 – August 1, 1997, Sacramento, CA. Available at www.webcom.com/software/issues/guide/docs/whsq.html.

In the Matter of Syncronys Software. (1996), Docket C-3688. Complaint at www.ftc.gov/os/9610/c3688cmp.htm. Decision and order at www.ftc.gov/os/9610/c3688d&o.htm.

In the Matter of Apple Computer, Inc. (1996), Docket C-3763. Complaint at www.ftc.gov/os/9708/c3763cmp.htm. Decision and order at www.ftc.gov/os/9708/c3763ord.htm.

Johnson v. Compaq (1997). Class action complaint filed against Compaq Computer in North Carolina. For the text, see users.aol.com/Cclass450/index.htm. This is a remarkable series of documents, and it is kept updated.

Kaner, C. (1997a). What is a Serious Bug? Defining a "Material Breach" of a Software License Agreement. (unpublished). *Meeting of the NCCUSL Article 2B Drafting Committee*, Redwood City, CA, January 10-12, 1997. (abbreviated version, *Software QA, 3*, #6.) Available at www.badsoftware.com/uccdefect.htm.

Kaner, C. (1997b). Remedies Provisions of Article 2B. (unpublished). *Meeting of the NCCUSL Article 2B Drafting Committee*, Redwood City, CA, January 10-12, 1997. Available at www.badsoftware.com/uccrem.htm.

Kaner, C. (1997c). Proposed Article 2B: Problems from the Customer's View: Part 1: Underlying Issues. U*CC Bulletin, January*, 1-8. Available at www.badsoftware.com/uccpart1.htm.

Kaner, C. (1997d). Proposed Article 2B: Problems from the Customer's View: Part 2: List of Key Issues. *UCC Bulletin, February*, 1-9. Available at www.badsoftware.com/uccpart2.htm.

Kaner, C. (1997e). Liability for Bad Software and Support. *Support Services Conference East*, Nashville, TN, March 12, 1997. Available at www.badsoftware.com/support1.htm.

Kaner, C. (1997f). Not Quite Terrible Enough Software. (unpublished). *Annual Meeting of the Software Engineering Process Group*, San Jose, CA, May 1997. Available at www.badsoftware.com/sepg.htm.

Kaner, C. (1997g), Article 2B is Fundamentally Unfair to Mass-Market Software Customers, circulated to the American Law Institute for its Article 2B review meeting, October, 1997. Available at www.badsoftware.com/ali.htm.

 Copyright © Kaner, 1997. All rights reserved.

Kaner, C. (1997h) Legal Issues Related to Software Quality. Proceedings of the Seventh International Conference on Software Quality, Montgomery, AL, October, 1997. Available, with accompanying slides, at www.badsoftware.com.

Kaner, C. (1996a). Quality Cost Analysis: Benefits and Risks. *Software QA, 3*, #1, 23 et seq. Available at www.badsoftware.com/qualcost.htm.

Kaner, C. (1996b). Software Negligence and Testing Coverage. *Proceedings of STAR 96 (Fifth International Conference on Software Testing, Analysis, and Review)*, Orlando, FL, May 16, 1996, 313 et seq. Portions of this paper were originally published in Kaner, (1995b) "Software Negligence & Testing Coverage", *Software QA Quarterly*, Vol. 2, #2, p. 18. Available at www.badsoftware.com/coverage.htm.

Kaner, C. (1996c). Uniform Commercial Code Article 2B: A new law of software quality. *Software QA, 3*, #2, 10 et seq. Available at www.badsoftware.com/uccsqa.htm.

Kaner, C. (1996d). Computer Malpractice. *Software QA, 3*, #4, 23 et seq. Available at www.badsoftware.com/malpract.htm.

Kaner, C. (1995a) Liability for Defective Documentation. *Software QA, 2*, #3, 8 et seq. Available at www.badsoftware.com/baddocs.htm.

Kaner, C., Falk, J., & Nguyen, H.Q. (1993). *Testing Computer Software*, 2nd Edition, International Thomson Computer Press.

Kaner, C. & Lawrence, B. (1997). UCC Changes Pose Problems for Developers. IEEE Software, *March/April*, 139-142.

Kaner, C. & Pels, D. (1997), Software Customer Dissatisfaction, *Software QA, 4*, #3, 24 et seq. Available at www.badsoftware.com/stats.htm.

Kaner, C. & Pels, D. (1996). User documentation testing: Ignore at your own risk. *Customer Care, 7*, #4, 7-8.

Princeton Graphics v. NEC Home Electronics (1990) Federal Supplement, *volume 732*, p. 1258 (United States District Court, Southern District of New York).

ProCD, Inc. v. Zeidenberg (1996) Federal Reporter, Third Series, *volume 86*, p. 1447 (7th Circuit Court of Appeals.)

Ritchie Enterprises v. Honeywell Bull, Inc. (1990) Federal Supplement, *Volume 730*, p. 1041 (United States District Court, Kansas).

Smedinghoff, T.J. (1993). *The SPA Guide to Contracts and the Legal Protection of Software.* Software Publishers Association.

Step-Saver Data Systems, Inc. v. Wyse Technology and The Software Link, Inc., (1991) Federal Reporter, Second Series, *volume 939*, p. 91 (Third Circuit Court of Appeals).

White, J.J. & Summers, R.S. (1995), *Uniform Commercial Code* (4th Edition), Practitioner Treatise Series, West Publishing Corp.

 Copyright © Kaner, 1997. All rights reserved.