The Law of Software Quality

Cem Kaner
J.D., Ph.D., ASQ-C.Q.E

ASM/SM
San Jose, CA
March, 2000

kaner@kaner.com
www.kaner.com (technical)
www.badsoftware.com (legal)
408-244-7000

About Cem Kaner

I'm in the business of improving software customer satisfaction. **I approach customer satisfaction from several angles. I've been a programmer, tester, writer, teacher, user interface designer, software salesperson, and a manager of user documentation, software testing, and software development, an organization development consultant and an attorney focusing on the law of software quality. These have provided many insights into relationships between computes, software, developers, and customers.**

I do the following types of work**:**

- ☒ **Training:** I offer this course (*Black Box Software Testing*), and shorter courses on
  - » ***Recruiting Software Testers*** (1 day)
  - » ***Concise Test Planning*** (1 day at conferences, 1.5 days at customer sites)
  - » ***Test Automation*** (2 days — I co-teach this with Doug Hoffman)
- ☒ **Consulting to software companies:** Primarily, I help software companies do better testing. I sometimes help companies improve their user documentation or help them with broader software development management issues.
- ☒ **Legal services:** I represent authors, programmers, and testers (I help consultants negotiate deals or negotiate with dissatisfied clients). Occasionally, I represent dissatisfied customers. I do a lot of work on legislation governing software contracting or electronic commerce.

Education:

- ☒ J.D. (law degree, 1993). Elected to the American Law Institute, 1999.
- ☒ Ph.D. (experimental psychology, 1984) (trained in *measurement theory* and in *human factors*, the field concerned with making hardware and software easier and safer for humans to use).
- ☒ B.A. (primarily mathematics and philosophy, 1974).
- ☒ Certified in Quality Engineering (American Society for Quality, 1992). *Senior Member*, American Society for Quality. Examiner(1994, 1995) for the California Quality Awards.
- ☒ I am also the founder and co- host of the Los Altos Workshops on Software Testing and the Software Test Managers' Roundtable.

Author:

- ☒ ***Testing Computer Software*** *(*1988; 2nd edition with Hung Nguyen and Jack Falk,1993). This received the *Award of Excellence* in the Society for Technical Communication's *Northern California Technical Publications Competition.*
- ☒ ***Bad Software: What To Do When Software Fails*** (with David Pels)
- ☒ I've also managed three tech pubs groups, and my staff have won several STC awards.

Copyright Notice and Disclaimer

These notes include some legal information, but you are not my legal client and I am not providing specific legal advice in the notes or in the course. Even if you ask me a question about a specific situation, you must understand that you cannot possibly give me enough information in a classroom setting for me to respond with a competent legal opinion. I may use your question as a teaching tool, and answer it in a way that I believe would "normally" be true but my answer could be completely inappropriate for your particular situation. I cannot accept any responsibility for any actions that you might take in response to my comments in this course. If you need legal advice, please consult your own attorney.

I chose the court cases for this course for their instructional value. I'm not trying to teach you a full survey of computer liability law in a day. I'm not trying to make sure that you have the most up-to-date cases or that you are exposed to all of the key legal theories.

Instead, my goal is to highlight certain issues that I think will be of interest to software developers, that might change how you develop, document, and test products.

A Look Ahead: Suggestions / Lessons

- ☒ Be reasonable. Do reasonable things.
- ☒ Test your documentation, collaterals, marketing materials, tech support answer books, and sales training books
- ☒ You can't deliver a perfect product, so plan for problems
- ☒ You can be sued for things that you didn't anticipate or intend, so identify and manage your risks
- ☒ Plan to do what you will do. Don't write that your plan is to follow an overly constrained life cycle or that you will develop an over-ambitious test plan. If you're not going to do it, don't create an erroneous expectation in your documentation.
- ☒ Scale your documentation, development formality, and testing according to the risks your product presents to your customers
- ☒ Don't expect to be able to bury test docs or design controversies if you get sued. If something is bad, make clear that it's bad.
- ☒ Investigate all customer complaints
- ☒ Don't rely on disclaimers
- ☒ Get sued in contract, not in tort

Contents

Introductions

Please introduce yourself.

What's your grounding in:

- ☒software development?
- ☒marketing?
- ☒customer care?
- ☒management?
- ☒law?

What kinds of software do you develop, sell or support? Price range?

Is your software custom-developed or mass market?

What two things do you most want to walk away with from this course?

Section 1.

Lawsuits and Quality

Customer Dissatisfaction

The Canadian government recently completed a study of the claims made on the packaging of consumer software:

Incorrect (and "potentially false or misleading") claims were made by 65% of all the software titles tested.

Study by Industry Canada's Competition Bureau. For the full study, go to http://strategis.ic.gc.ca/FBP and search for "software".

Computer-related complaints made Better Business Bureau's top 10 for 1995, even higher than used car dealers. We did worse in 1996.

(The BBB's data for 1997 merged computing with consumer electronics, making comparisons with the 1995 and 1996 data difficult. The combined totals yield higher ranks (more complaints), of course.)

Sophisticated Customers Have Trouble Too

Albert Stark lays out problems that *software support staff* encounter when *they* try to buy and install problem management systems. Support staff provide an interesting example, because they're usually pretty talented at making things work.

Stark points out that:

- ☒ "The system will not do everything promised."
- ☒ "System functionality is typically overstated."
- ☒ "You'll need to purchase additional modules to get the functionality you need."
- ☒ "Features you need are scheduled for a future release."
- ☒ "The out-of-box reality is less than expected."
- ☒ "You'll need to purchase additional hardware."
- ☒ "The software will be more complex than it appeared during the sales cycle."
- ☒ "System customization will not go smoothly" even though "Vendors can make customization look easy."

In a parallel session at the same conference, speaker asked an audience of publishers' technical support staff how many of them would trade in their problem management system if they could. Over half the attendees raised their hands.

Dissatisfaction Costs Publishers Money

- 1996--200 million calls to tech support.
- The industry spends about $25 per call.
- Software companies spend about $3 per minute providing support for PC-based products, and $5 per minute (or more) for UNIX and mainframe products.
- In companies that have pushed many complainers to the internet, handling the issues raised by live calls cost as much as $150 to $400 per incident (averages reported at a 1999 Support Services Conference).
- Customer complaints have skyrocketed. Over 7 years, ratio of support to total employees in computer-related companies has gone from 1 in 12 to 1 in 6.

For references and additional data, see Kaner & Pels, Bad Software: What To Do When Software Fails, Wiley, 1998.

Customers Have Legitimate Problems

- In those 200 million calls for support, software customers spent over 3 billion minutes on hold.
- This is tip of the iceberg because most American customers don't complain.
- Cross-industry study: Complaining software customers left on hold for longer than any other industry studied, even airlines and gov't offices.
- At peak times, 85% of calls into tech support get busy signals.
- 58% of support staff get less than 1 week of training before independently handling phone calls.
- Complaints involving software / hardware from more than one vendor take 3 to 18 times as long to resolve.
- Business' cost of ownership of a PC is often estimated at $8000 to $11,000 per year.

We Don't Recognize Our Costs

- ☒At 1997 ASP Customer Support Conference, 90% of attendees said they believe they are delivering a reasonable level of customer service & support.

  How can their perceptions differ so much from their customers?

- ☒1994 study by Help Desk Institute, 82% of responding support organizations said that they didn't know their cost-per-call (what they spend per complaint).

- ☒Few companies have problem resolution systems that report support cost for a given bug in the field.

Many of us Don't Look at Long-Term Costs

☒Customer dissatisfaction with quality significantly reduces a company's sales, but several (in my experience, most) companies ignore the dissatisfaction-associated revenue risks because they don't know how to estimate their magnitude.

The degree to which people underestimate long-term effects is illustrated by the following example.

☒ Microsoft spent $500,000,000 bringing its customer support from blecch to world class. But customer perceptions still rank MS near average as a support provider. Therefore, there might not be an obvious immediate payoff in sales volume. Result--a leading magazine said,

"Despite lots of wishful thinking to the contrary, spending money to upgrade a company's service reputation remains a lousy investment."

But it was in the same period that MS took over leadership in the office applications market, typically in competition with publishers that were intent on cost-reducing their technical support. I don't think that MS would have had a chance of stealing its competitors' customers if they had paid more attention to preserving their customers' loyalty.

We Aren't Collecting Data.

Capers Jones, Patterns of Software Systems Failure & Success:

> The number one root cause of cancellations, schedule slippages, and cost overruns is the chronic failure of the software industry to collect accurate historical data from ongoing and completed projects. This failure means that the vast majority of major software projects are begun without anyone having a solid notion of how much time will be required.
>
> Software is perhaps the only technical industry where neither clients, managers, nor technical staff have any accurate quantitative data available to them from similar projects when beginning major construction activities. . . .
>
> A result that is initially surprising but quite common across the industry is to discover that the software management community within a company knows so little about the technology of software planning and estimating that they do not even know of the kinds of tools that are commercially available.

Some Customers Sue

There have been several recent lawsuits involving bad software and / or bad support.

Many software products are sold with disclaimers that purport to deny all possible liability for defective products. Sometimes these are cited by customer support staff to complaining customers. Sometimes they are cited in project team meetings to reassure testers or customer service staff that shipping the product with an awful bug will result in a manageable state of affairs.

Every court in America that has ruled on the shrink-wrapped warranty disclaimer (in software or in hard goods) has said it is ineffective. If your software is seriously flawed, you release it at your company's non-trivial risk.

Lawsuits and Quality

Deadly Diseases

In his classic book, “Out of the Crisis,” W.E. Deming named seven “Deadly Diseases” that afflict American businesses. Number 7 was “Excessive costs of liability, swelled by lawyers that work on contingency fees.” (p. 98)

- ☒*Are people who sue unreasonable?*
- ☒*Do lawsuits force companies to overspend on quality or safety?*
- ☒*How do we decide that a company overspends on quality or safety?*

Lawsuits and Quality

Debunking Myths

As long as we think of lawsuits as demonic threats to the public, we'll be too busy listening to our prejudices to understand the laws governing quality.

So, let's do a little debunking . . . .

Lawsuits and Quality

Litigation Crisis?

1994 Annual Report of the
Judicial Council of California

(This was heavily cited in some political campaigns as proof of a litigation crisis)

Superior Court Civil Filings:

1983-84	1992-93	***increase***
561,916	684,070	122,154 (21.7%)

(1983-84 is the first of the 10 years in this study. 1992-93 is the last of the 10 years.)

Lawsuits & Quality

Litigation Crisis?

Superior Court Civil Filings:

_____ _____ _____

Personal injury, death, property damage:

_____ _____ _____

Other civil petitions (Child Support):

_____ _____ _____

Lawsuits & Quality

McDonalds and Spilled Coffee

(Courtesy of the Consumers Attorneys of California)

There is a lot of hype about the McDonald's scalding coffee case. No one is in favor of frivolous cases or outlandish results;however, it is important to understand some points that were not reported in most of the stories about the case. McDonald's coffee was not only hot, it was scalding capable of almost instantaneous destruction of skin, flesh and muscle. Here is the whole story.

Stella Liebeck of Albuquerque, New Mexico, was in the passenger seat of her grandson's car when she was severely burned by McDonald's coffee in February 1992. Liebeck, 79 at the time, ordered coffee that was served in a styrofoam cup at the drivethrough window of a local McDonald's.

After receiving the order, the grandson pulled his car forward and stopped momentarily so that Liebeck could add cream and sugar to her coffee. (Critics of civil justice, who have pounced on this case, often charge that Liebeck was driving the car or that the vehicle was in motion when she spilled the coffee; neither is true.) Liebeck placed the cup between her knees and attempted to remove the plastic lid from the cup. As she removed the lid, the entire contents of the cup spilled into her lap.

The sweatpants Liebeck was wearing absorbed the coffee and held it next to her skin. A vascular surgeon determined that Liebeck suffered full thickness burns (or third degree burns) over 6 percent of her body, including her inner thighs, perineum, buttocks, and genital and groin areas. She was hospitalized for eight days, during which time she underwent skin grafting. Liebeck, who also underwent debridement treatments, sought to settle her claim for $20,000, but McDonald's refused.

During discovery, McDonald's produced documents showing more than 700 claims by people burned by its coffee between 1982 and 1992. Some claims involved third degree burns substantially similar to Liebecks. This history documented McDonald's knowledge about the extent and nature of this hazard.

McDonald's also said during discovery that, based on a consultants advice, it held its coffee at between 180 and 190 degrees fahrenheit to maintain optimum taste. He admitted that he had not evaluated the safety ramifications at this temperature. Other establishments sell coffee at substantially lower temperatures, and coffee served at home is generally 135 to 140 degrees.

McDonalds and Spilled Coffee

Further, McDonald's quality assurance manager testified that the company actively enforces a requirement that coffee be held in the pot at 185 degrees, plus or minus five degrees. He also testified that a burn hazard exists with any food substance served at 140 degrees or above, and that McDonald's coffee, at the temperature at which it was poured into styrofoam cups, was not fit for consumption because it would burn the mouth and throat. The quality assurance manager admitted that burns would occur, but testified that McDonald's had no intention of reducing the "holding temperature" of its coffee.

Plaintiff's expert, a scholar in thermodynamics applied to human skin burns, testified that liquids, at 180 degrees, will cause a full thickness burn to human skin in two to seven seconds. Other testimony showed that as the temperature decreases toward 155 degrees, the extent of the burn relative to that temperature decreases exponentially. Thus, if Liebeck's spill had involved coffee at 155 degrees, the liquid would have cooled and given her time to avoid a serious burn.

McDonald's asserted that customers buy coffee on their way to work or home, intending to consume it there. However, the company's own research showed that customers intend to consume the coffee immediately while driving.

McDonald's also argued that consumers know coffee is hot and that its customers want it that way. The company admitted its customers were unaware that they could suffer third degree burns from the coffee and that a statement on the side of the cup was not a "warning" but a "reminder" since the location of the writing would not warn customers of the hazard.

The jury awarded Liebeck $200,000 in compensatory damages. This amount was reduced to $160,000 because the jury found Liebeck 20 percent at fault in the spill. The jury also awarded Liebeck $2.7 million in punitive damages, which equals about two days of McDonald's coffee sales.

Postverdict investigation found that the temperature of coffee at the local Albuquerque McDonald's had dropped to 158 degrees fahrenheit.

The trial court subsequently reduced the punitive award to $480,000 or three times compensatory damages even though the judge called McDonald's conduct reckless, callous and willful. Subsequent to remittitur, the parties entered a postverdict settlement.

Litigation Over Bad Quality

The essence of quality-related litigation is a customer seeking to transfer losses caused by a defective product back to the company that made the defect or sold it.

Lawsuits and Quality

The Economics of Quality: Quality/Cost Analysis

The ***Cost of Quality*** associated with a product is the total amount that the company spends to achieve and cope with the quality of that product. It includes investments in improving quality and expenses arising from inadequate quality.

One of the key goals of quality engineering is to minimize the total cost of quality associated with a product or project.

Lawsuits and Quality

Quality-Related Costs

Prevention	***Appraisal***
Cost of preventing software errors, documentation errors, an any other sources of customer dissatisfaction	All costs of all types of inspection (testing).
Internal failure	***External failure***
ALL costs of coping with errors discoveed during development.	All costs of coping with errors discovered, typicall by your customers, after the product is released.

Lawsuits and Quality

Examples of Quality-Related Costs

Prevention	***Appraisal***
Staff training Requirements analysis Early prototyping Fault-tolerant design Defensive programming Usability analysis Clear specification Accurate internal documentation Pre-purchase evaluation of the reliability of development tools	Design review Code inspection Glass box testing Black box testing Training testers Beta testing Test automation Usability testing Pre-release out-of-box testing by customer service staff
Internal Failure	***External Failure***
Bug fixes Regression testing Wasted in-house user time Wasted tester time Wasted writer time Wasted marketer time Wasted advertisements Direct cost of late shipment Opportunity cost of late shipment	Technical support calls Answer books (for Support) Investigating complaints Refunds and recalls Interim bug fix releases Shipping updated product Supporting multiple versions in the field PR to soften bad reviews Lost sales Lost customer goodwill Reseller discounts to keep them selling the product Warranty, liability costs

Quality Cost Analysis: Risks of this Approach

Quality Cost analysis provides several benefits:

- ☒ new opportunities to find / create common ground with other groups in the company
- ☒ analysis tool for budgeting and planning quality-related activities
- ☒ excellent communication tool for senior management

But there are some risks, too, because we might unwittingly open ourselves up to devastating lawsuits:

- ☒ Don't think you've solved customer dissatisfaction problems by driving down support costs.
- ☒ Don't bet that you can safely rely on disclaimers.
- ☒ ***Don't consider only your own external failure costs. Know the costs to your customer.***

Lawsuits and Quality

The External Failure Cost of the Pinto

Benefits and Costs Relating to Fuel Leakage Associated with the Static Rollover Test Portion of FMVSS 208	
Benefits -- Savings	
180 burn deaths	$200,000 each
180 serious burn injuries	$67,000 each
2100 burned vehicles	$700 each
Total Benefit	$49.5 million
Costs	
11 million cars	$11 each
1.5 million trucks	$11 each
Total Costs	$137 million

Lawsuits and Quality

Customers' External Failure Costs are Important

Seller: external costs	***Customer: failure costs***
These are the types of costs absorbed by the seller that releases a defective product.	***These are the types of costs absorbed by the customer who buys a defective product.***
Technical support calls Preparing answer books Investigating complaints Refunds and recalls Interim bug fix releases Shipping updated product Supporting multiple versions in the field PR to soften harsh reviews Lost sales Lost customer goodwill Reseller discounts to keep them selling the product Warranty, liability costs Gov't investigations	Wasted time Lost data Lost business Embarrassment Frustrated employees quit Demos or presentations to potential customers fail because of the software Failure during tasks that can only be done once Cost of replacing product Reconfiguring the system Cost of recovery software Cost of tech support Injury / death

One way to make an argument based on customer costs is to evaluate costs to an in-house group of users.

Influencing Others Based on Costs

It's probably impossible to fix every bug. Sometimes the development team will choose to not fix a bug based on their assessment of its risks for them, without thinking of the costs to other stakeholders in the company.

- ☒ Probable tech support cost.
- ☒ Risk to the customer.
- ☒ Risk to the customer's data or equipment.
- ☒ Visibility in an area of interest to reviewers.
- ☒ Extent to which the bug detracts from the use of the program.
- ☒ How often will a customer see it?
- ☒ How many customers will see it?
- ☒ Does it block any testing tasks?
- ☒ Degree to which it will block OEM deals or other sales.

To argue against a deferral, ask yourself which stakeholder(s) will pay the cost of keeping this bug. Flag the bug to them.

Lawsuits and Quality

Litigation & the use of Quality / Cost Analysis

Benefit: Quality Cost Analysis gives a systematic framework for managing all quality-related aspects of the product.

Risk: too easy to ignore customers' costs. If your external failure costs are substantially less than your customers' you may make an unreasonable fix-vs.-ship decision.

Litigation: Reasonable customers have reason to sue if your product's failures cost them more than their cost and aggravation from litigation. This isn't a lottery. It's an effort to transfer some of the losses caused by your defective product back to you.

Cem Kaner, Ph.D., J.D. **P.O. Box 1**
Attorney at Law **Santa Clara, CA 9505**
kaner@kaner.com **408-244-7000**

Quality Cost Analysis: Benefits and Risks

Published in *Software QA*, 3, #1, 1996, p. 23.

> "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."[1]

Joseph Juran, one of the world's leading quality theorists, has been advocating the analysis of quality-related costs since 1951, when he published the first edition of his *Quality Control Handbook*. Feigenbaum made it one of the core ideas underlying the Total Quality Management movement.[2] It is a tremendously powerful tool for product quality, including software quality.

What is Quality Cost Analysis?

Quality costs are the costs associated with preventing, finding, and correcting defective work. These costs are huge, running at 20% - 40% of sales.[3] Many of these costs can be significantly reduced or completely avoided. One of the key functions of a Quality Engineer is the reduction of the total cost of quality associated with a product.

Here are six useful definitions, as applied to software products. Figure 1 gives examples of the types of cost. Most of Figure 1's examples are (hopefully) self-explanatory, but I'll provide some additional notes on a few of the costs[4]:

Prevention Costs: Costs of activities that are specifically designed to prevent poor quality. Examples of "poor quality" include coding errors, design errors, mistakes in the user manuals, as well as badly documented or unmaintainably complex code.

Note that most of the prevention costs don't fit within the Testing Group's budget. This money is spent by the programming, design, and marketing staffs.

- ***Appraisal Costs:*** Costs of activities designed to find quality problems, such as code inspections and any type of testing.

---

1 Gryna, F. M. (1988) "Quality Costs" in Juran, J.M. & Gryna, F. M. (1988, 4th Ed.), ***Juran's Quality Control Handbook,*** McGraw-Hill, page 4.2.

2 Feigenbaum, A.V. (1991, 3rd Ed. Revised), ***Total Quality Control,*** McGraw-Hill, Chapter 7.

3 Gryna, F. M. "Quality Costs" in Juran, J.M. & Gryna, F. M. (1988, 4th Ed.), ***Juran's Quality Control Handbook,*** McGraw-Hill, page 4.2. I'm not aware of reliable data on quality costs in software.

4 These are my translations of the ideas for a software development audience. More general, and more complete, definitions are available in Campanella, J. (Ed.) (1990), ***Principles of Quality Costs,*** ASQC Quality Press, as well as in Juran's and Feigenbaum's works.

Design reviews are part prevention and part appraisal. To the degree that you're looking for errors in the proposed design itself when you do the review, you're doing an appraisal. To the degree that you are looking for ways to strengthen the design, you are doing prevention.

Failure Costs: Costs that result from poor quality, such as the cost of fixing bugs an the cost of dealing with customer complaints.

Internal Failure Costs: Failure costs that arise before your company supplies its product to the customer. Along with costs of finding and fixing bugs are many internal failure costs borne by groups outside of Product Development. If a bug blocks someone in your company from doing her job, the costs of the wasted time, the missed milestones, and the overtime to get back onto schedule are all internal failure costs.

For example, if your company sells thousands of copies of the same program, you w probably print several thousand copies of a multi-color box that contains and describes the program. You (your company) will often be able to get *much* better deal by booking press time in advance. However, if you don't get the artwork to the printer on time, you might have to pay for some or all of that wasted press time anyway, and then you may have to pay additional printing fees and rush charges to get the printing done on the new schedule. This can be an added expense of many thousands of dollars.

Some programming groups treat user interface errors as low priority, leaving them until the end to fix. This can be a mistake. Marketing staff need pictures of the product's screen long before the program is finished, in order to get the artwork for the box into the printer on time. User interface bugs – the ones that will be fixed lat – can make it hard for these staff members to take (or mock up) accurate screen shots. Delays caused by these minor design flaws, or by bugs that block a packaging staff member from creating or printing special reports, can cause the company to miss its printer deadline.

Including costs like lost opportunity and cost of delays in numerical estimates of the total cost of quality can be controversial. Campanella (1990)[1] doesn't include these in a detailed listing of examples. Gryna (1988)[2] recommends against including costs like these in the published totals because fallout from the controversy over them can kil the entire quality cost accounting effort. I include them here because I sometimes find them very useful, even if it might not make sense to include them in a balance sheet.

External Failure Costs: Failure costs that arise after your company supplies the product to the customer, such as customer service costs, or the cost of patching a released product and distributing the patch.

External failure costs are huge. It is much cheaper to fix problems before shipping the defective product to customers.

---

1 ***Principles of Quality Costs***, ASQC Quality Press, Appendix B, "Detailed Description of Quality Cost Elements."

2 "Quality Costs" in Juran, J.M. & Gryna, F. M. (1988, 4th Ed.), ***Juran's Quality Control Handbook***, McGraw-Hill, pages 4.9 - 4.12.

Some of these costs must be treated with care. For example, the cost of pub relations efforts to soften the publicity effects of bugs is probably not a huge percentage of your company's PR budget. You can't charge the entire PR budget as a quality-related cost. But any money that the PR group has to spend to specifically cope with potentially bad publicity due to bugs is a failure cost.

I've omitted from Figure 1 several additional costs that I don't know how to estimate, and that I suspect are too often too controversial to use. Of these, my two strongest themes are cost of high turnover (people quit over quality-related frustration – this definitely includes sales staff, not just development and support) and cost of lost pride (many people will work less hard, with less care, if they believe that the final product will be low quality no matter what they do.)

- ***Total Cost of Quality:*** The sum of costs: Prevention + Appraisal + Internal Failure + External Failure.

Figure 1. Examples of Quality Costs Associated with Software Products.

Prevention	***Appraisal***
Staff training Requirements analysis Early prototyping Fault-tolerant design Defensive programming Usability analysis Clear specification Accurate internal documentation Evaluation of the reliability of development tools (before buying them) or of other potential components of the product	Design review Code inspection Glass box testing Black box testing Training testers Beta testing Test automation Usability testing Pre-release out-of-box testing by customer service staff
Internal Failure	***External Failure***
Bug fixes Regression testing Wasted in-house user time Wasted tester time Wasted writer time Wasted marketer time Wasted advertisements[1] Direct cost of late shipment[2] Opportunity cost of late shipment	Technical support calls[3] Preparation of support answer books Investigation of customer complaints Refunds and recalls Coding / testing of interim bug fix releases Shipping of updated product Added expense of supporting multiple versions of the product in the field PR work to soften drafts of harsh reviews Lost sales Lost customer goodwill Discounts to resellers to encourage them to keep selling the product Warranty costs Liability costs Government investigations[4] Penalties[5] All other costs imposed by law

[1] The product is scheduled for release on July 1, so your company arranges (far in advance) for an advertising campaign starting July 10. The product l too many bugs to ship, and is delayed until December. All that advertising money was wasted.

[2] If the product had to be shipped late because of bugs that had to be fixed, the direct cost of late shipment includes the lost sales, whereas the opportunity cost of the late shipment includes the costs of delaying other projects while everyone finished this one.

[3] Note, by the way, that you can reduce external failure costs without improving product quality. To reduce post-sale support costs without increasing customer satisfaction, charge people for support. Switch from a toll-free support line to a toll line, cut your support staff size and you can leave callers c hold for a long time at their expense. This discourages them from calling back. Because these cost reductions don't increase customer satisfaction, the seller's cost of quality is going down, but the customer's is not.

[4] This is the cost of cooperating with a government investigation. Even if your company isn't charged or penalized, you spend money on lawyers, etc.

[5] Some penalties are written into the contract between the software developer and the purchaser, and the developer pays them if the product is late or l specified problems. Other penalties are imposed by law. For example, the developer/publisher of a computer program that prepares United States taxe liable for penalties to the Internal Revenue Service for errors in tax returns that are caused by bugs or design errors in the program. The publishers are treated like other tax preparers (accountants, tax lawyers, etc.). See *Revenue Ruling 85-189* in *Cumulative Bulletin* 1985-2, page 341.

What Makes this Approach Powerful?

Over the long term, a project (or corporate) cost accounting system that tracks quality-related costs can be a fundamentally important management tool. This is the path that Juran and Feigenbaum will lead you down, and they and their followers have frequently and eloquently explained the path, the system, and the goal.

I generally work with young, consumer-oriented software companies who don't see TQM programs as their top priority, and therefore my approach is more tactical. There is significant benefit in using the language and insights of quality cost analysis, on a project/product by project/product basis, even in a company that has no interest in Total Quality Management or other formal quality management models[1].

Here's an example. Suppose that some feature has been designed in a way that you believe will be awkward and annoying for the customer. You raise the issue and the project manager rejects your report as subjective. It's "not a bug." Where do you go if you don't w to drop this issue? One approach is to keep taking it to higher-level managers within produc development (or within the company as a whole). But without additional arguments, you'll often keep losing, without making any friends in the process.

Suppose that you change your emphasis instead. Rather than saying that, in your opinion, customers won't be happy, collect some other data[2]:

Ask the writers: Is this design odd enough that it is causing extra effort to document? Would a simpler design reduce writing time and the number of pages in the manual?

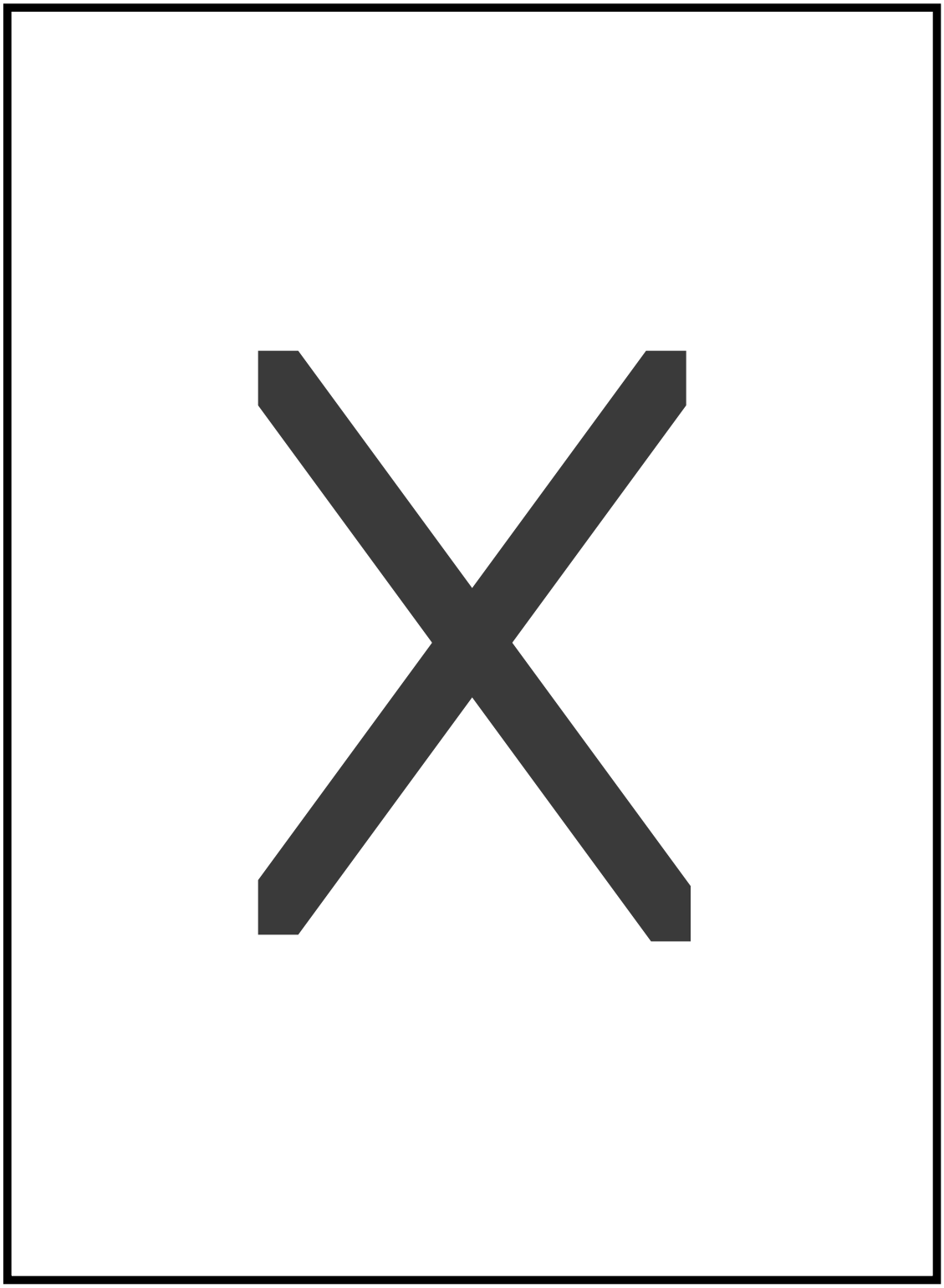
Ask the training staff: Are they going to have to spend extra time in class, and write more supplementary materials because of this design?

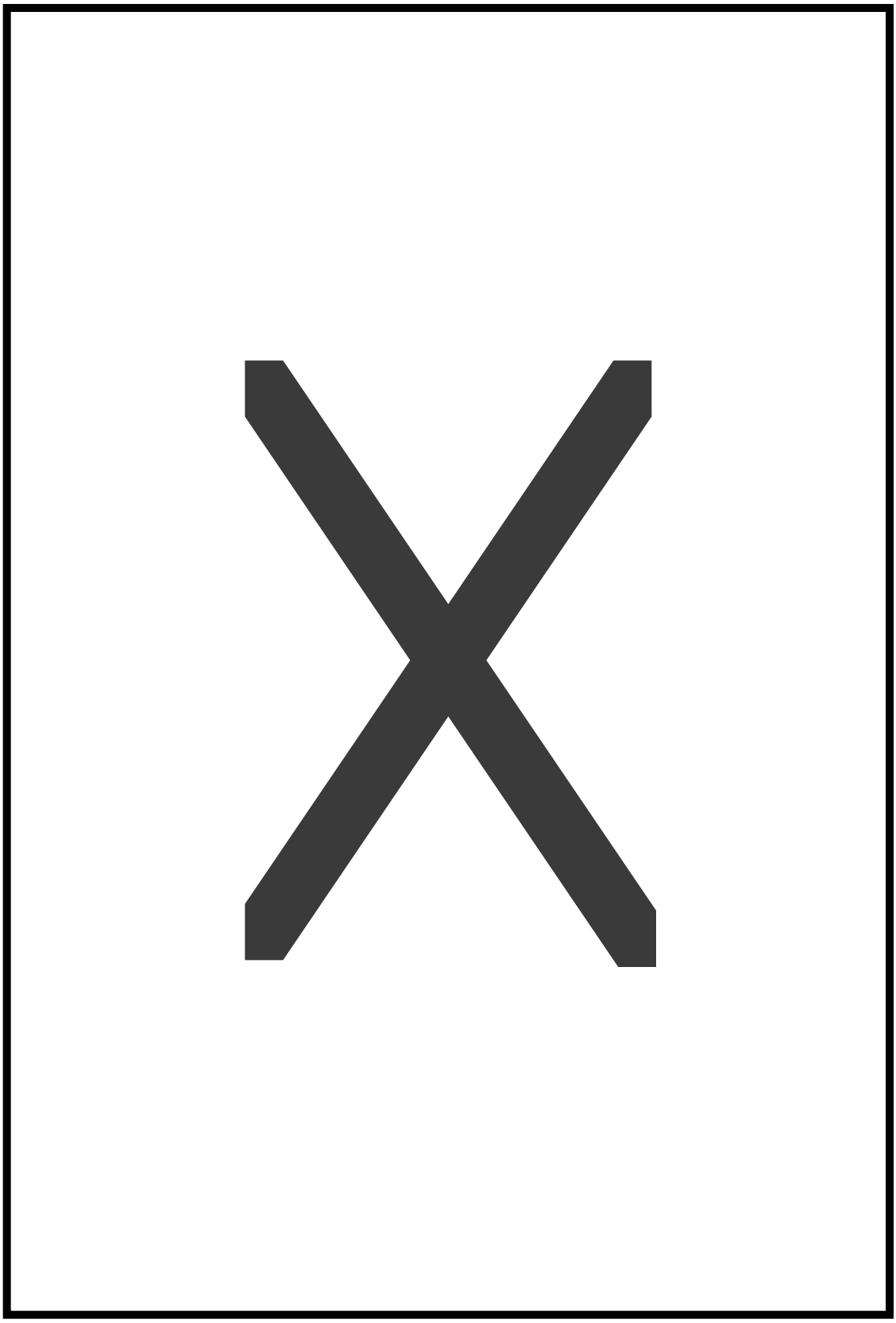
- ***Ask Technical Support and Customer Service:*** Will this design increase support costs? Will it take longer to train support staff? Will there be more calls for explanations or help? More complaints? Have customers asked for refunds in previous versions of the product because of features designed like this one?

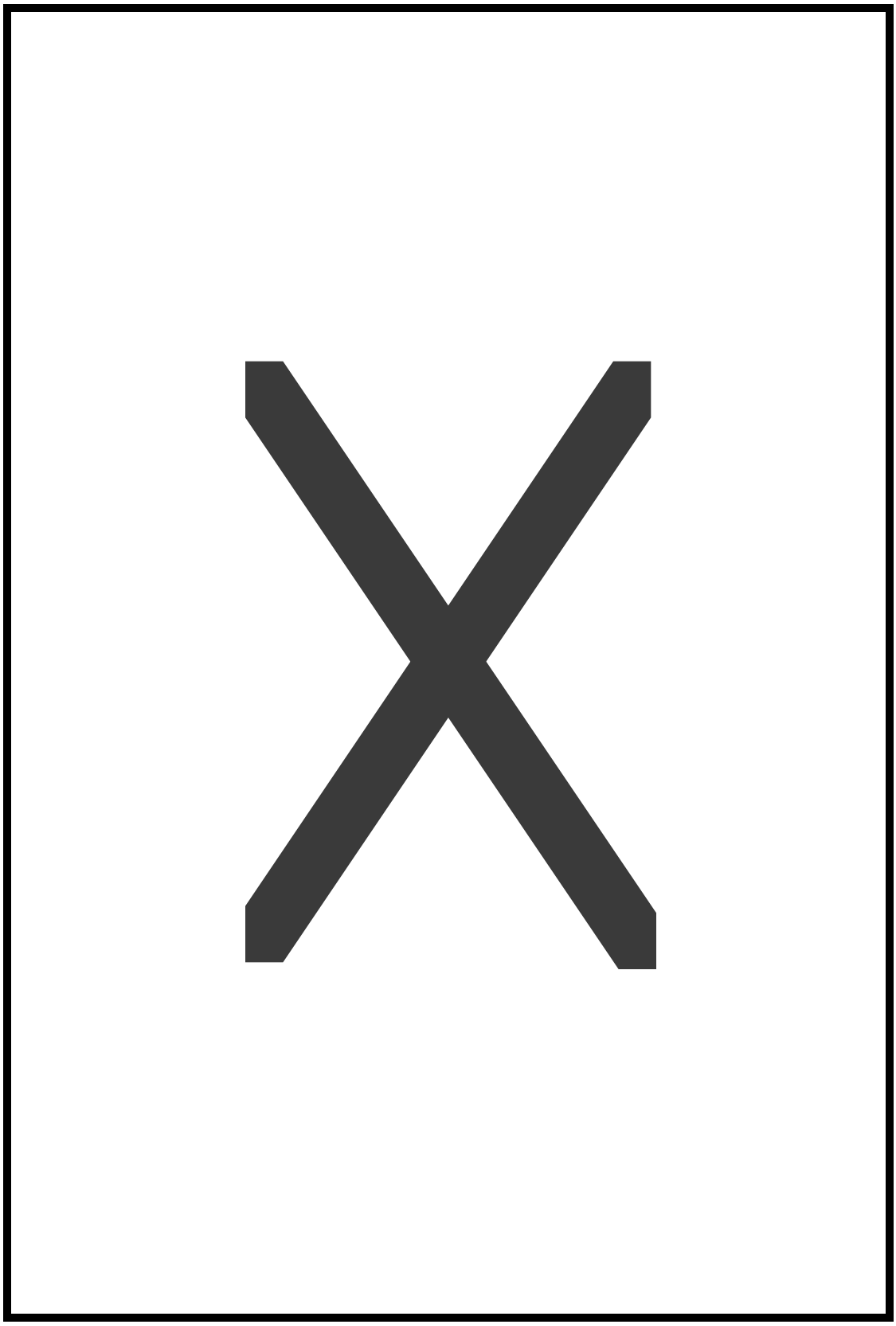
---

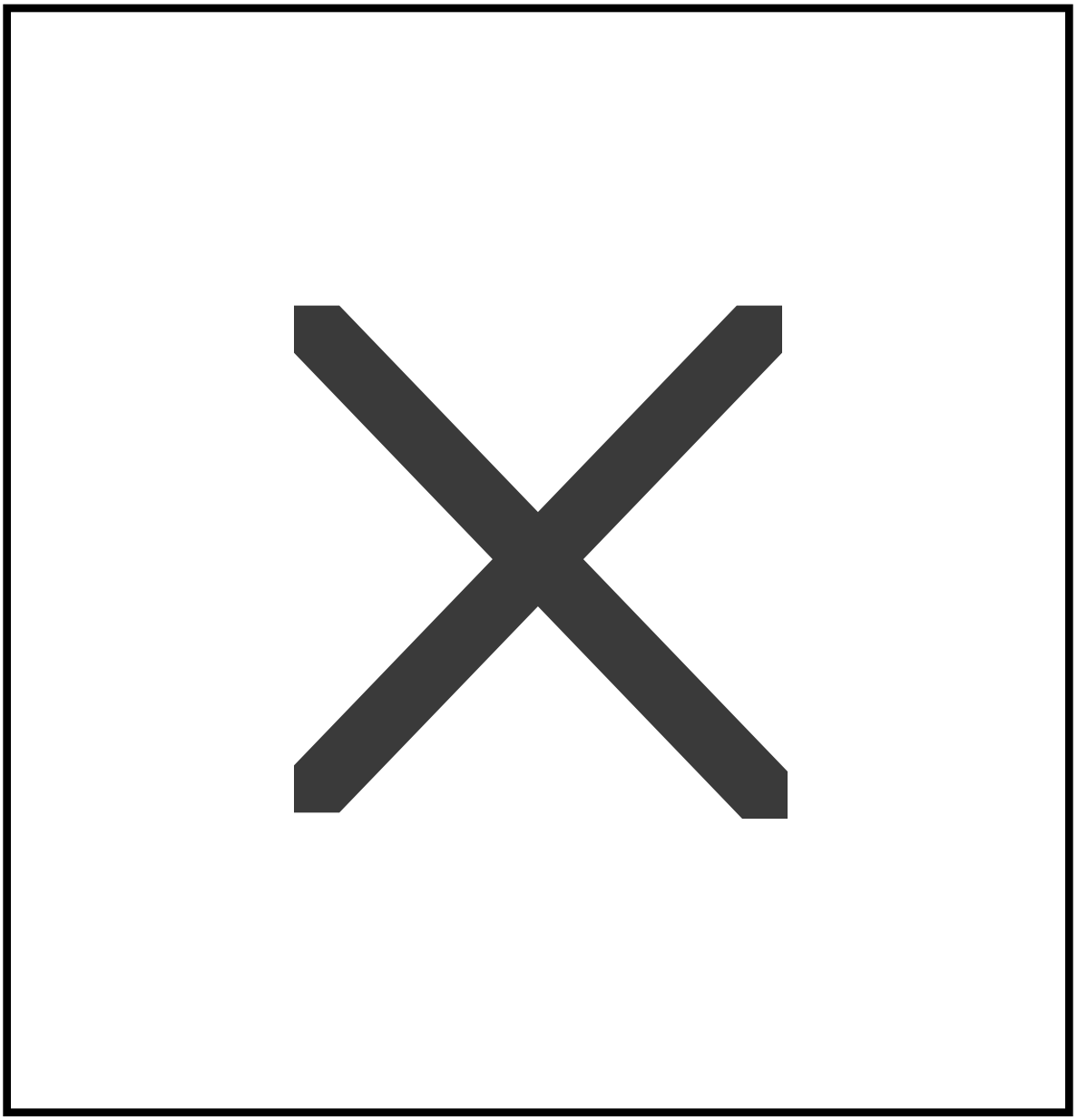
1 I am most definitely not saying that a tactical approach is more practical than an integrated, long-term approa Gryna notes that there are two common approaches to cost-of-quality programs. One approach involves one-sl studies that help the company identify targets for significant improvement. The other approach incorporates quality cost control into the structure of the business. (Gryna, 1988, in Juran, J. M. & Gryna, F. M. (1988, 4th Ed.), ***Juran's Quality Control Handbook***, McGraw-Hill, pages 4.2 onward.) The one-shot, tactical approach can prove the benefit of the more strategic, long-term system to a skeptical company.

2 Be sensitive to how you do this. If you adopt a tone that says that you think the project manager and the programming staff are idiots, you won't enjoy the long-term results.









Notes

Law of Software Quality

Section 2.

Some Interesting Lawsuits

Some interesting lawsuits

Family Drug Store

Family Drug Store of New Iberia, Inc. v. Gulf States Computer Services, Inc.

563 So.2d 1324 (Louisiana Court of Appeal, 1990).

Must a publisher of a poorly designed product satisfy its customers?

Some interesting lawsuits

Family Drug Store

Family Drug Store of New Iberia

The plaintiffs are a couple of pharmacists who bought a computer program known as the Medical Supply System from Gulf States. After they realized what they had bought, they asked for, and then sued for, a refund. Here were some of the problems of the system:

"(1) all data had to be printed out, and could not be viewed on the monitor;

(2) the information on the monitor would appear in code;

(3) numerical codes were needed in order to open a new patient file

(4) the system was unable to scroll.

The court found that the seller had not in any ay misrepresented the system, and that it as not useless even though it was awkward to use. Further, the price of the software

Some interesting lawsuits

Step-Saver Data Systems

Step-Saver Data Systems v. Wyse Technology and The Software Link.

939 F.2d 91 (3rd Circuit, United States Court of Appeals) 1991.

Is a shrink-wrap warranty disclaimer valid?

Some interesting lawsuits

Step-Saver Data Systems

Step-Saver (a vertical market reseller) repeatedly bought Multilink Advanced, an allegedly MS-DOS compatible operating system, from The Software Link (TSL). On each box was a disclaimer: the software was sold AS IS, without warranty; TSL disclaimed all express and implied warranties; and a purchaser who didn't agree to this disclaimer should return the product, unopened, to TSL for a refund.

Step-Saver sued TSL, claiming that Multilink Advanced was not MS-DOS compatible. TSL argued that Step-Saver had accepted the terms of the warranty disclaimer when it opened each package, and therefore Step-Saver could not sue.

Should this disclaimer be valid for the first sale? Subsequent sales?

Some interesting lawsuits

Burroughs v. Hall Affiliates

Burroughs Corporation v. Hall Affiliates

423 So. 2d 1348 (Supreme Court of Alabama, 1982).

Is publisher liable for negligent (rather than fraudulent) misrepresentations?

Some interesting lawsuits

Burroughs v. Hall Affiliates

Hall imports artificial flowers, and bought a Burroughs B80-40 computer in 1977 to handle its accounting and inventory functions. The system didn't work and Hall sued claiming that Burrough's salespeoples' representations about the system were fraudulent. The court listed 4 representations:

1. the machine would do inventory and accounting simultaneously
2. the machine was capable of multiprogramming
3. the machine was capable of operating a terminal display unit in a data communications environment
4. the machine and all of its component parts were new.

The court found that there was no evidence of *intentional* misrepresentation.

Should Burroughs be held liable for its mistake?

Some interesting lawsuits

Therac-25

The Therac-25 accidents--

Nancy Leveson, ***Safeware: System Safety & Computers***. Addison Wesley, 1995, Appendix A.

Should a software developer be liable for injurious user errors?

Some interesting lawsuits

Therac-25

Therac 25 is a linear accelerator that does radiation therapy. There are two settings:

- ☒ concentrated beam, high intensity X-rays, and
- ☒ lower energy electron beams used to destroy tumors that are closer to the surface.

Therac 25 stems from Therac 20 and Therac 6, which had hardware interlocks. You couldn't set the high energy level without having everything else consistent. Here, the hardware interlock was gone, and the software could be overridden, which it was. 6 major accidents. The incidents involved some degree of user error (wrong setting, ignored user error message).

Should AECL (Atomic Energy Canada) be held liable anyway?

Some interesting lawsuits

GM v. Johnston

General Motors v. Johnston

592 So.2d 1054 (Supreme Court of Alabama, 1992)

What's a company's responsibility for hazards discovered after product release?

Some interesting lawsuits

GM v. Johnston

Johnson purchased a new Chevrolet 2500 pickup in 1988, drove it for less than 200 miles, over two days. With his 7 year old grandson in the truck, he pulled up to a stop sign, started the truck again, and it stalled. A larger truck collided with his truck, injuring Johnson and killing the grandson.

GM had received reports of stalling problems in vehicles like this one, and a dealer service bulletin advised dealers that "rolling, hunting or surging idles" could be fixed by replacing the PROM. The software on the modified PROM was changed from the original software. This PROM controlled the fuel injector. GM chose to replace the PROM for complaining customers, but not to announce a recall and make the change widely available.

Should GM have issued a recall or a warning to Johnson?

Six interesting lawsuits

Winter v. G.P. Putnam

Winter v. G.P. Putnam's Sons

938 F.2d 1033, (9th Circuit, United States Court of Appeal, 1991.)

Is a publisher liable for injury-causing errors in its publications?

Six interesting lawsuits

Winter v. G.P. Putnam

Winter became seriously ill from picking and eating mushrooms after relying on *The Encyclopedia of Mushrooms*, published by Putnam. Putnam did not verify the material in the book and did not intentionally include the error.

Should Putnam be held liable for this unintended misinformation?

Notes

Law of Software Quality

Section 3.

Ground Rules of Lawsuits

Ground Rules

Overview of a Lawsuit

Something bad happens
Pre-complaint negotiations
File a complaint
<Class certification hearing>
Test for failure to state a claim
File answer to the complaint
Discovery
Summary judgment motion
Trial
Damages
<Hearing on costs / fees>
Appeal

Ground Rules

Due Process

Due process.

☒The claim must fit the precise requirements of a clear legal rule.

Due Process

☒No person shall . . . be deprived of life, liberty, or property without due process of law. 5th Amendment.

☒Nor shall any State deprive any person of life, liberty, or property, without due process of law. 14th Amendment.

Ground Rules

Discovery

Discovery is open.

☒The parties to the trial get to find out about each other.

- ☒ The plaintiff can ask about anything that is not privileged, if the query is calculated to lead to the discovery of admissible evidence.
- ☒ No privilege against self-incrimination in civil controversy, and no privilege ever for a corporation.
- ☒ Your marketing plans, customer support records, bug tracking system, and internal design review documents are discoverable.
- ☒ There is a limited exception for after-corrected defects
- ☒ There is a limited privilege for corporate self-examination.

Ground Rules

Proof

Plaintiff must prove her case by

☒**a *preponderance of the evidence***
(more likely to be true than not)

or by

☒***clear & convincing evidence*.**

If the plaintiff doesn't prove her case, the defendant wins.

Notes

Law of Software Quality

Section 4.

The Key Legal Theories

The Key Legal Theories

1. Intentional torts
2. Contracts
3. Misrepresentation
4. Consumer protection
5. Negligence
6. Strict products liability
7. Malpractice

Key Legal Theories

1. Intentional Torts

Intentional unlawful interference with, or harm to, a person or her property, reputation, privacy, or business relations.

Examples:

- ☒Battery
- ☒Conversion
- ☒Computer tampering
- ☒Libel
- ☒Fraud

2. Contract Theories

A *contract* is an agreement between two or more people (or companies) that creates obligations to do or to provide particular things.

A software contract can involve *goods* (such as a program bought at a store) or *services* (such as custom programming), or some mix of the two (such as a program that comes with a maintenance contract).

Key Legal Theories

3. Misrepresentation

- ☒ False representation by the seller
- ☒ of a material (important) fact
- ☒ that the plaintiff justifiably relies on
- ☒ and as a result, the plaintiff is damaged.

Misrepresentation can be:

- ☒ Innocent
- ☒ Negligent
- ☒ Fraudulent

A misrepresentation is fraudulent if the maker

- ☒ knows or believes that the matter is not as he represents it to be, or
- ☒ does not have the confidence in the accuracy of his representation that he states or implies, or
- ☒ knows that he does not have the basis for his representation that he states or implies

Key Legal Theories

4. Consumer Protection

Deceptive Trade Practices

Unfair Competition

Additional Warranty Rules

Additional Leasing Rules

Additional Negative Option Rules

False Claims Act

Uniform Deceptive Trade Practices Act

> A person engages in deceptive trade practices when s/he represents that goods or services have sponsorship, approval, characteristics, ingredients, uses, benefits, or quantities that they do not have.

Key Legal Theories

5. Negligence

Elements of a negligence case:

- ☒Duty:
  - » *products must not create an unreasonable risk of injury or property damage.*
- ☒Breach
- ☒Causation
- ☒Damages

Key Legal Theories

Contracts vs Negligence

Contracts

Law of quality

Duty is to give the customer what s/he paid for.

Likely types of suits:

- ☒ corrupts or loses its own data
- ☒ doesn't work; never delivered
- ☒ erroneous reports
- ☒ bugs that waste time or make the program hard to use
- ☒ compatibility features don't work
- ☒ cost-reduction promises aren't realized

Negligence

Law of safety

Duty is to make products that are not unreasonably unsafe.

Likely types of suits:

- ☒ corrupts or loses data obtained from some other program
- ☒ damages connected peripherals
- ☒ injures the user
- ☒ injures customer who follows its directions
- ☒ embedded software causes accidents
- ☒ UI design causes accidents

Key Legal Theories

6. Strict Products Liability

A company is liable to a victim if it sells a product that is:

- ☒defective, and
- ☒unreasonably dangerous, and
- ☒the cause of personal injury or property damage

A design may be unreasonably dangerous if it fails to meet the safety expectations of a reasonable consumer.

Key Legal Theories

7. Malpractice

Malpractice is the failure to exercise the skill and knowledge normally possessed by members of a profession or trade.

- **Programmer malpractice?**

  Getting licensed means that we will finally qualify to be sued in malpractice.

- **Professional advice malpractice?**

  The more your advertising & docs make a customer think she can replace a professional with your product, the better her chances of success in a malpractice suit if your program gives bad advice.

Notes

Law of Software Quality

Section 5.

A Quick Scan of Damages

Damages

You can collect different types of damages under different legal theories. Let's very quickly scan through a few damages definitions:

- ☒Contract damages
- ☒Compensatory & punitive
- ☒Economic & non-economic
- ☒Statutory
- ☒Costs & attorney fees

Contract Damages

Benefit of the bargain

- ☒ The difference between the purchase price (or the value stated by the seller) and the actual value of the product

Incidental damages

- ☒ include costs of returning a defective product and finding a replacement; handling expenses, processing expenses.

Consequential damages

- ☒ includes economic losses and costs of injuries and property damage

Damages

Compensatory & Punitive

Compensatory damages

- ☒ These make good the loss without giving the buyer or victim any profit.

Punitive damages

- ☒ These are awarded to punish the defendant, not to compensate the plaintiff.
- ☒ These are rarely awarded, hard to get, and closely scrutinized by trial and appellate judges. You must prove, by clear & convincing evidence, that the defendant's behavior was fraudulent, oppressive, or outrageous.

Damages

Economic & Non-Economic

Economic losses

- ☒benefit of the bargain damages, repair costs, incidental expenses, down time, loss of use, lost profits

Non-economic damages

- ☒payment for suffering, pain, fear, loss of consortium

Damages

Statutory

Damages that are recoverable by the plaintiff in an amount that is determined by statute rather than by the amount of harm suffered by the plaintiff.

Costs & Attorney Fees

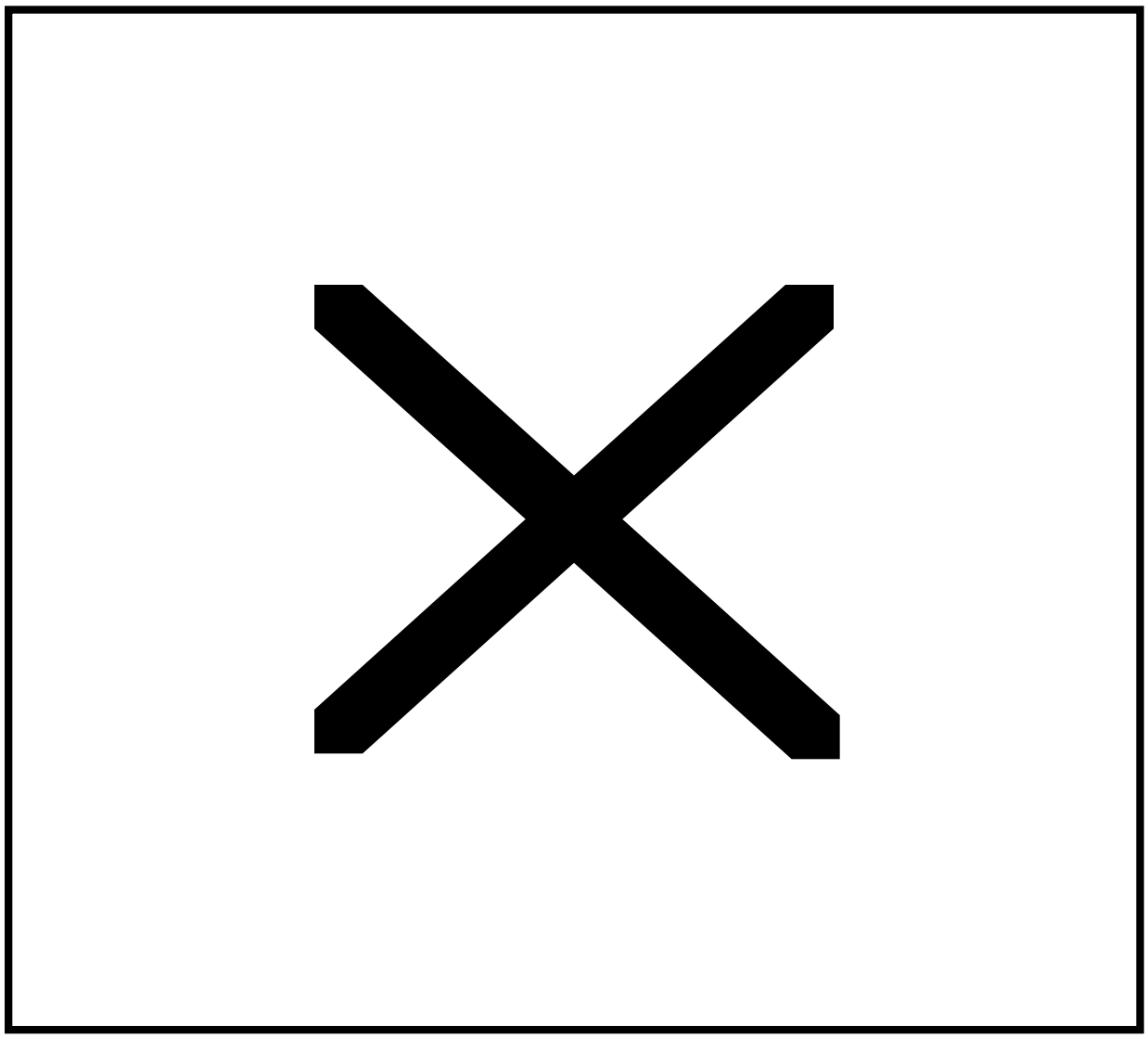
Costs

- ☒Courts normally award costs

Attorney fees

- ☒Normally unavailable, but
  - » may be provided by contract
  - » unfair trade practice suits
  - » several other statutes

Damages Available Under Different Legal Theories



Notes

Law of Software Quality

Section 6.

Intentional Torts

Key Legal Theories

1. Intentional Torts

Intentional unlawful interference with, or harm to, a person or her property, reputation, privacy, or business relations.

Examples:

- ☒Battery
- ☒Conversion
- ☒Computer tampering
- ☒Libel
- ☒Fraud

Intentional Torts

Computer Tampering, Conversion

Computer tampering

- ☒ Unlawful access to the computer of another, or introduction of software that causes damage to the computer of another.

Conversion case

- ☒ Clayton X-Ray Co. v. Professional Systems Corp. (WD 43583, Mo. Ct. App., W.D. 8/6/91; 9 Computer Law 38). PSC introduced a time bomb b/c Clayton had not fully paid for the software. The bomb shut down the system. PSC refused to turn the system back on until Clayton paid for it. Jury awarded punitive and compensatory damages.
- ☒ This is not a case of time bomb in the initial product. This is a time bomb introduced after the fact, by intrusion into plaintiff's computer.

Notes

Law of Software Quality

Section 7.

Contracts:

Failure to Perform the Contract

Contracts

Fundamentals of Contract Law

- ☒ offer
- ☒ counter-offer
- ☒ acceptance
- ☒ consideration
- ☒ battle of the forms
- ☒ warranty
- ☒ modification
  - » the pre-existing duty rule
- ☒ oral contracts are valid (except under statute of frauds)
- ☒ contracts by conduct are valid
- ☒ parole evidence

Contracts

Uniform Commercial Code

Article 2 of the Uniform Commercial Code governs sales of goods.

- ☒ gap fillers, and implied terms
- ☒ implied warranty of merchantability
- ☒ implied warranty of fitness for a particular purpose
- ☒ battle of the forms rules:
  - » contract by conduct
  - » forms as proposals for modification
  - » materiality
- ☒ modification rules
- ☒ some rules apply only to merchants

Article 2 has been consistently applied to COTS transactions, but much less often to custom service contracts (e.g. custom software, consulting, etc.)

Contracts

Family Drug Store v. Gulf States Computer

The basic holding of this case is that a computer program can be extremely awkward to use and badly designed without imposing liability on the seller.

Two pharmacists bought a computer program known as the Medical Supply System from Gulf States. After they realized what they had bought, they asked for, and then sued for, a refund.

The court found that the seller had not in any ay misrepresented the system, and that it as not useless even though it was awkward to use. Further, the price of the software was about $2500 compared to $10,000 for other packages. *The plaintiffs had gotten what they'd paid for.*

Contracts

Family Drug Store v. Gulf States Computer

Don’t take *Family Drug Store* to mean that a bad user interface is not a breach of contract. Other cases have found a breach of contract because of a bad UI.

What counts in the contract case is what was promised.

Contracts

Custom Software Development: Waterfall Model

Under this model, software development proceeds in discrete stages:

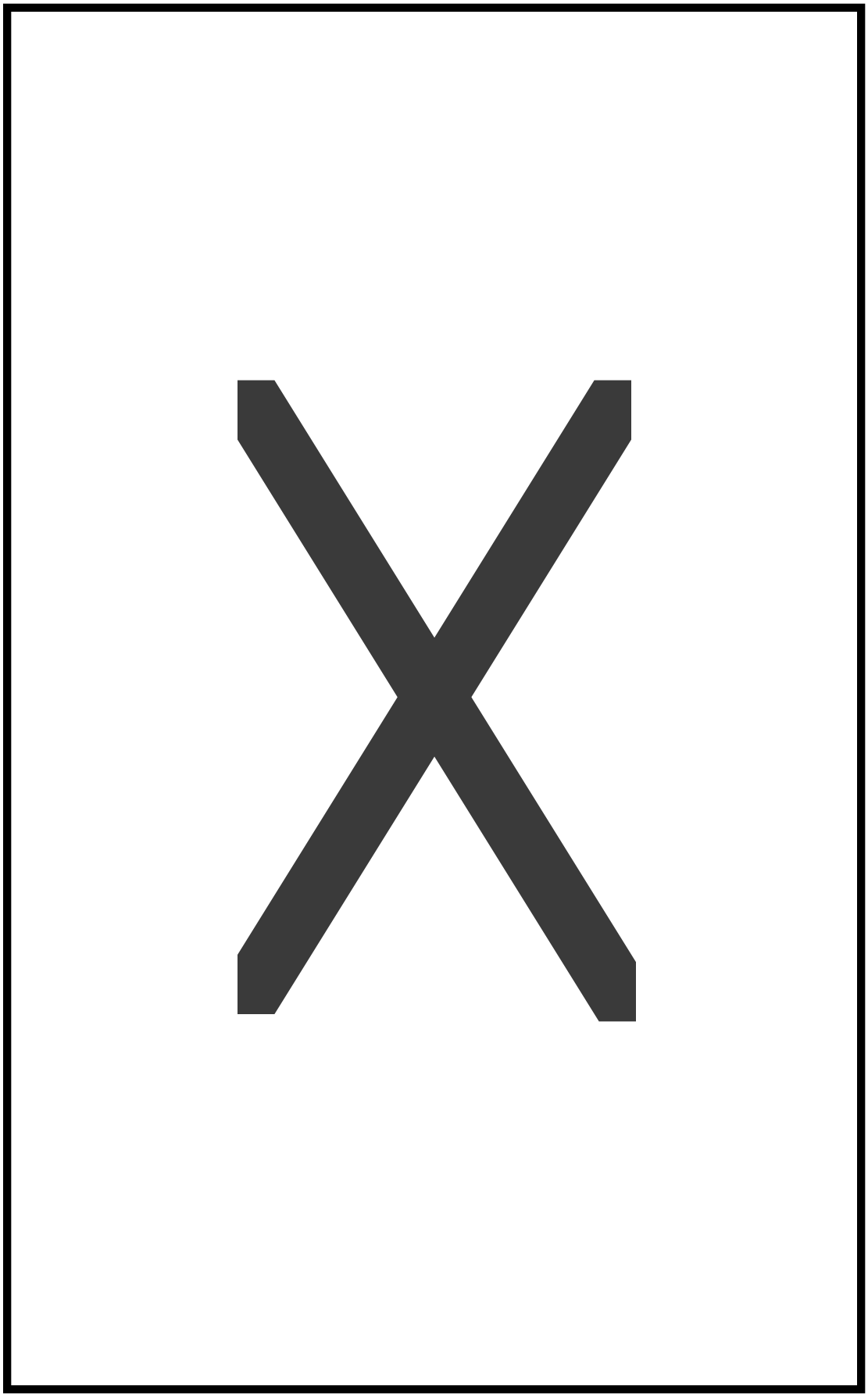
- ☒requirements analysis
- ☒specification
- ☒design
- ☒coding
- ☒testing
- ☒product release / delivery
- ☒post-sale support

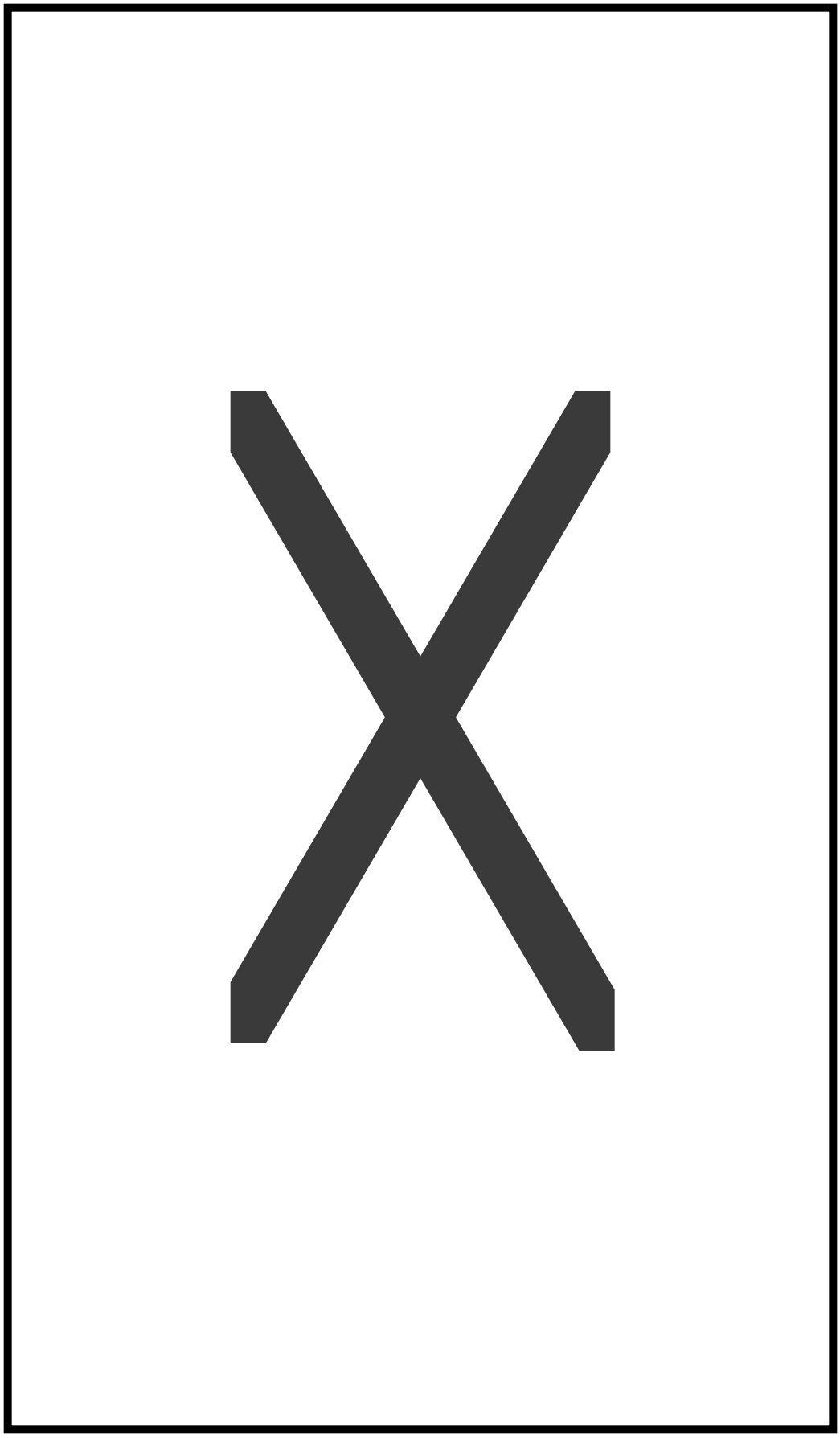
This approach may not be optimal for development, but it seems clear for contract control.

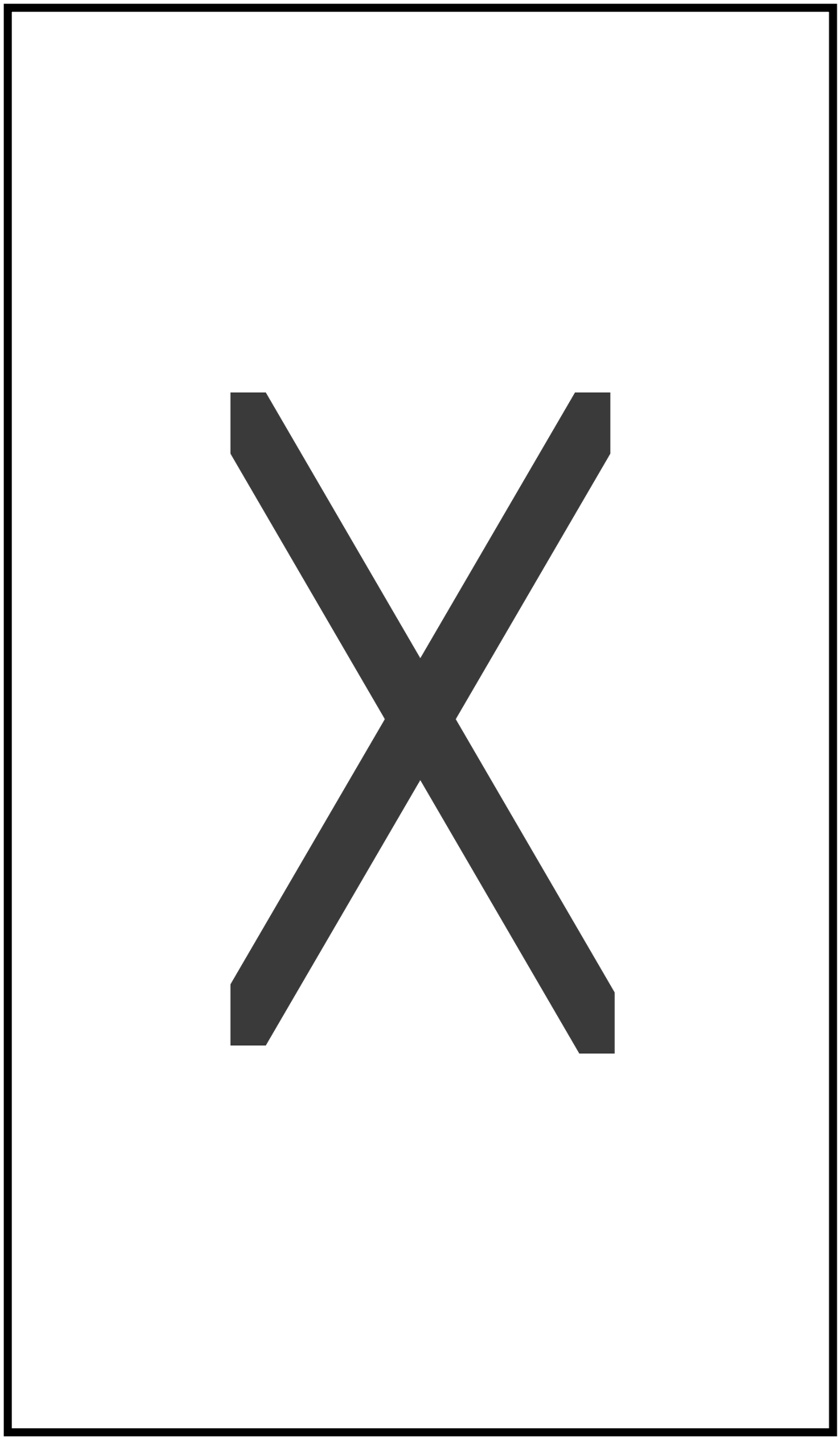
Contracts

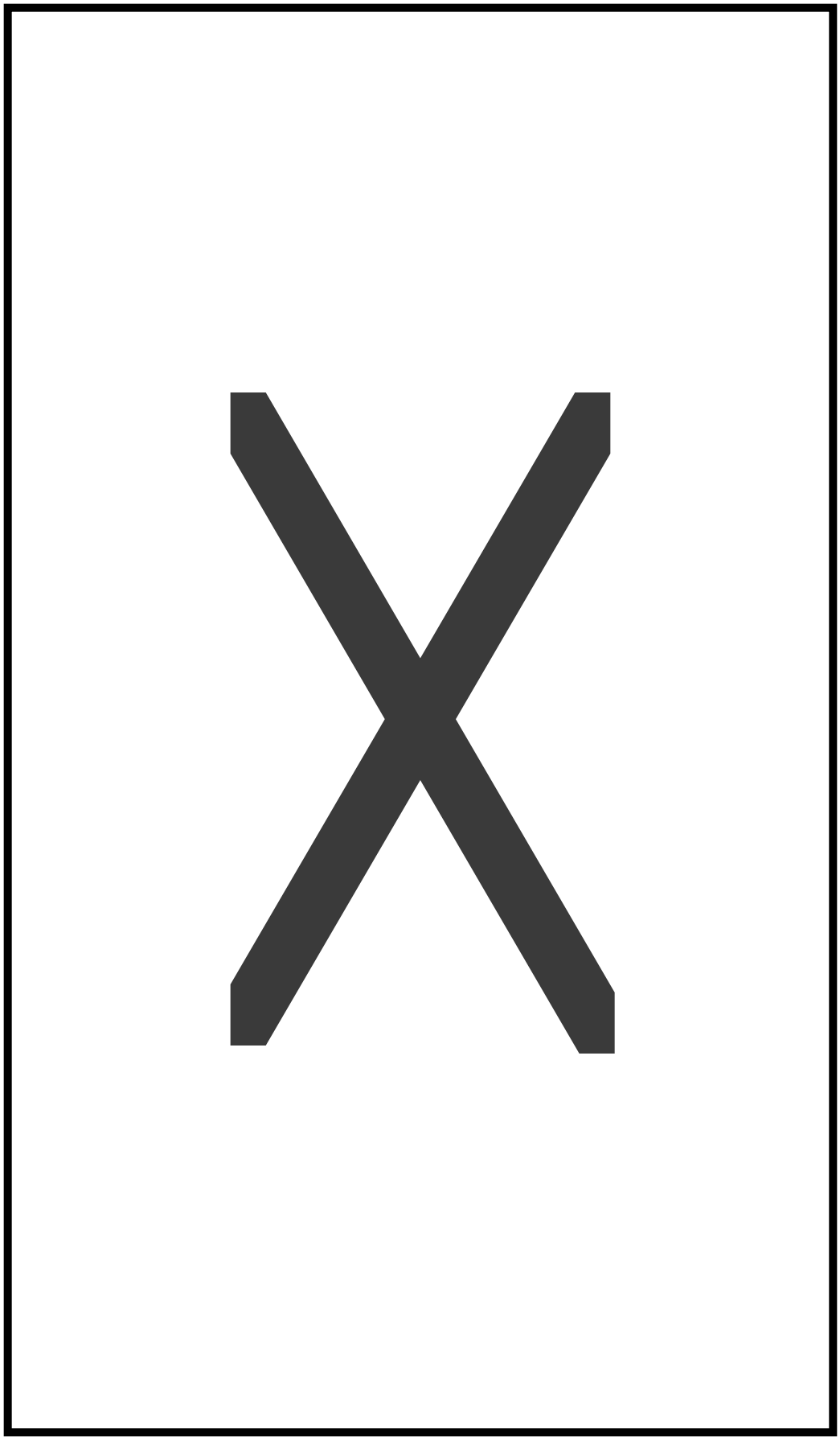
Custom Software Development: Dispute Resolution

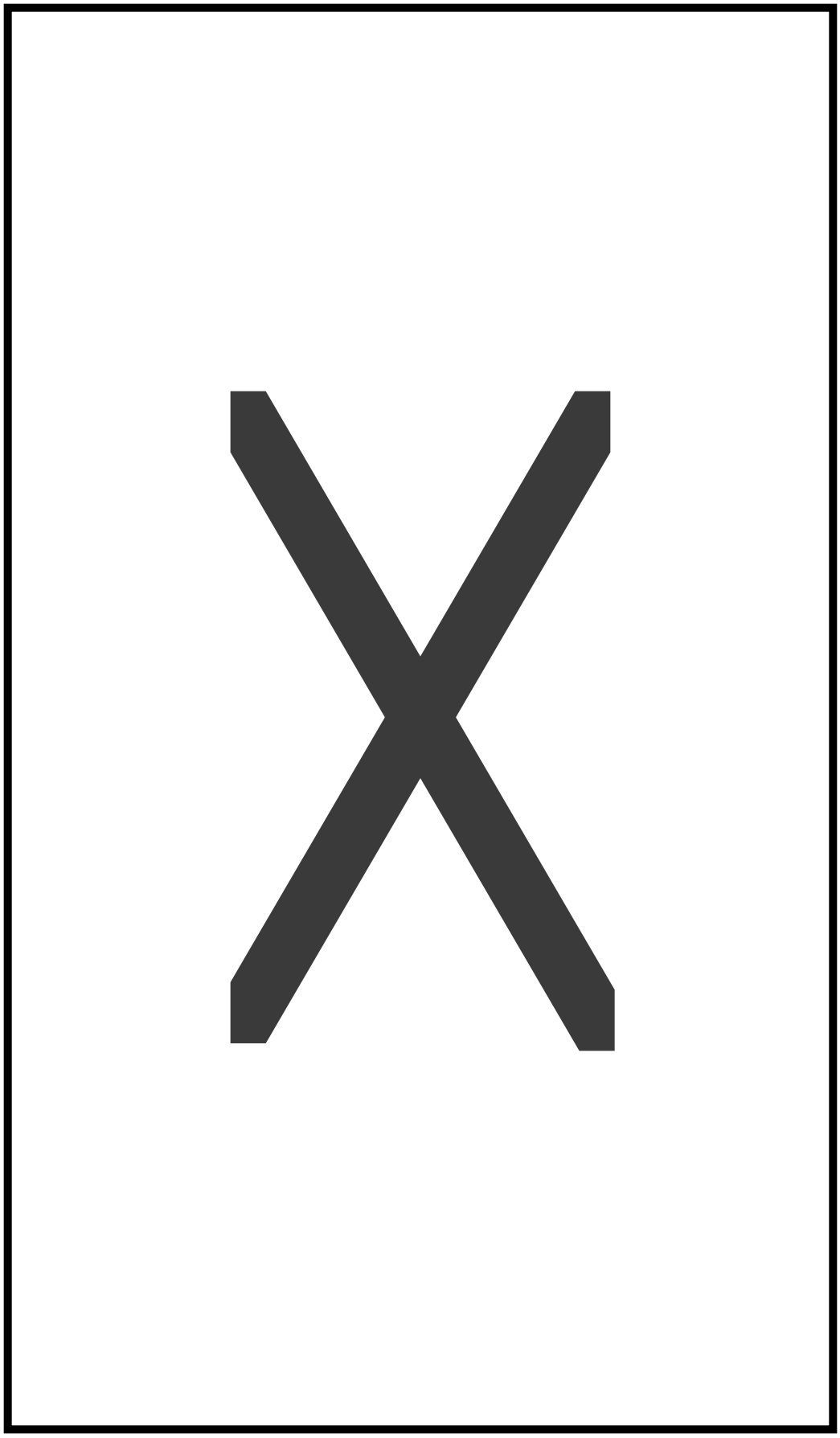
- The typical contract considers the possibility of a complete breakdown of the agreement and manages risks and ambiguities through dispute resolution clauses (forum selection, arbitration, etc.)
- Contracts are less likely to include dispute resolution procedures to follow mid-project, if needed, that are designed to save the project.











(ii) the software fails to perform in conformance with the specifications and this failure either deprives the licensee of a significant benefit of the product or results in costs to the licensee that exceed the price paid for the software;

(iii) where the specifications are silent, the software's performance is unreasonable and it results in costs to the licensee that exceed the price paid for the software. The licensee has the burden of demonstrating that a reasonable licensor would consider the software's performance to be unreasonable.

(d) If the contract is between merchants, and it contains specification documents, then a breach is material if:

(i) the software fails to perform in conformance with and in the time required by express performance standards or specifications;

(ii) the software fails to perform in conformance with the specifications and this failure either deprives the licensee of a significant benefit of the product or results in costs to the licensee that exceed the price paid for the software;

(iii) where the specifications are silent, the software's performance is unreasonable and it results in costs to the licensee that exceed the price paid for the software. The licensee has the burden of demonstrating that a reasonable licensor would consider the software's performance to be unreasonable.

(e) If the contract is not between merchants, and the licensor provides the specification documents that are incorporated in the contract, then a breach is material if:

(i) the software fails to perform in conformance with and in the time required by express performance standards or specifications;

(ii) the software fails to perform in conformance with the specifications and this failure either deprives the licensee of a significant benefit of the product or results in costs to the licensee that exceed the price paid for the software;

(iii) the software fails to perform in conformance with the end user documentation or other documentation delivered to the licensee and this failure either deprives the licensee of a significant benefit of the product or results in costs to the licensee that exceed the price paid for the software;

(iv) where the specifications and other documentation are silent, the software's performance is unreasonable and as a result, it either deprives the licensee of a significant benefit of the product or it results in costs to the customer that exceed the price paid for the software. The licensee has the burden of demonstrating that a reasonable person would consider the software's performance to be unreasonable.

(f) If the contract is for a mass-market license, then a breach is material if:

(i) the software fails to perform in conformance with the end user documentation or other documentation delivered to the licensee and this failure either deprives the licensee of a significant benefit of the product or results in costs to the customer that exceed the price paid for the software;

(ii) where the documentation is silent, the software's performance is unreasonable and as a result, it either deprives the licensee of a significant benefit of the product or it results in costs to the licensee that exceed the price paid for the software. The licensee has the burden of demonstrating that a reasonable person would consider the software's performance to be unreasonable.

(g) A material breach of contract occurs if the cumulative effect of nonmaterial breaches by the same party satisfies the standards for materiality.

(h) If there is a breach of contract, whether or not material, the aggrieved party is entitled to the remedies provided for in this article and the agreement.

What Happens from Here?

By the time you read this proposal, I will have circulated it to the Article 2B Drafting Committee. They'll probably consider it at the January 10-12 Drafting Committee meeting at the Sofitel Hotel in Redword City, California. The next meeting of the Committee will be in Atlanta from February 21 to 23, 1997. I will compile comments that people send me, and will summarize them for this meeting. You can also attend either meeting yourself. Few of the attendees are non-lawyers, but you are welcome speak if you have something informative to say.

This process will continue for a few more months (four meetings are scheduled in 1997), probably resulting in legislation that is introduced in the state legislatures in 1998. Whether you or I participate in this process or not, the result will include rules that govern software quality, laying out the ground rules under which we decide whether bugs are features and whether they need to be fixed. We can influence the process.

To read the latest draft of Article 2B, and to send comments directly to Ray Nimmer the Drafting Committee's Reporter, visit the Article 2B home page at www.law.uh.edu/ucc2b

APPENDIX

The following notes weren't included in the SQA article but did appear in the memo actually considered by the UCC Drafting Committee.

How Should We Define a Serious Defect?

Failure to conform to specifications is a common theme in legal books, but many of the software development contracts provide vague, incomplete specifications that w change over time without being updated in the contract itself. Discussions within the software development community consistently recognize that most failures in commercial software products are due to errors in the specifications or requirements A widely used number is that 80% of the money spent fixing or dealing with softwar problems can be traced back to requirements errors.

As a result, texts that focus on software errors don't limit themselves to failure to meet a specification (this type of failure is called *nonconformance*). Here are some examples from well respected texts in the field:

IEEE (1989), *IEEE Standard Dictionary of Measures to Produce Reliable Software*, ANSI/IEEE Standard 982.1-1988, p. 13:

> Defect: A product anomaly. Examples include such things as (1) omissions and imperfections found during early life cycle phases and (2) faults contained in software sufficiently mature for test or operation. See also *fault*.

IEEE (1994), *IEEE Standard Classification for Software Anomalies*, IEEE Standard 1044-1993, p. 3.

> Anomaly: Any condition that deviates from expectations based on requirements specifications, design documents, user documents, standards, etc., or from someone's perceptions or experiences. Anomalies may be found during, but not limited to, the review, test, analysis, compilation, or use of software products or applicable documentation.

Grady, Robert B. & Caswell, Deborah, L. (1987), *Software Metrics: Establishing a Company-Wide Program*. PTR Prentice-Hall, p. 78

> A defect is any flaw in the specification, design, or implementation of a product. . . . If a flaw could not possibly have been detected, or if it could have been detected and would not have been corrected then it an enhancement. Defects do not include typographical or grammatica errors in engineering documentation.

Ishikawa, Kaoru (translated by David J. Lu) (1985), *What is Total Quality Control? The Japanese Way* Prentice-Hall. (Ishikawa is the leading Japanese quality control theoriest):

On page 46 ff. he explains why he doesn't trust quality as measured i terms of compliance with standards and specifications. The problem is that these are not true measures of the quality of the product. He works through an excellent example dealing with a role of newsprint. The true measure of quality is whether the paper rips while on the rotary press. Published standards, in terms of such things as tensile strength, provide only secondary measures of how the product will perform in the field. Continuing, on page 56, "There are no standards—whether they be national, international, or company-wide—that are perfect. Usually standards contain some inherent defects. Consumer requirements also change continuously, demanding higher quality year after year. Standards that were adequate when th were first established, quickly become obsolete. [¶]We engage in QC satisfy customer requirements"

Jones, Capers (1991), *Applied Software Measurement*, McGraw-Hill, page 273.

A software defect is simply a bug which if not removed would cause a program or system to fail or to produce incorrect results. Note: the very common idea that a defect is a failure to adhere to some user requirements is unsatisfactory because it offers no way to measure requirements defects themselves, which constitute one of the larger categories of software error.

Mundel, August, B. (1991), *Ethics in Quality*, ASQC Quality Press, p. 164.

Any variation from the specifications is a nonconformity . . . . There is a group of nonconformities which represent serious threats to the welfare of users and bystanders. These nonconformities are called *defects*, and they not only can cause injury but may also result in the manufacturers, designers, or sellers being sued under the product liability laws. There are also a class of defects called *design defects* which can be responsible for customer dissatisfaction, loss, injury or death. Despite the fact that all of the product conforms to the design the product is faulty and is not properly designed.

Myers, Glenford J. (1976), *Software Reliability: Principles & Practices*, John Wiley & Sons, pp. 4-6. This is one of the seminal books in the software testin / quality control literature.

One common definition is that a software occurs when the software does not perform according to its specifications. This definition has one fundamental flaw: it tacitly assumes that the specifications are correct. This is rarely, if ever, a valid assumption: one of the major sources of errors is the writing of specifications. . . . [¶] A second common definition is that an error occurs when the software does no perform according to its specifications providing that it is used within its design limits. This definition is actually poorer than the first one....

> [¶] A third possible definition is that an error occurs when the software does not behave according to the official documentation or publications supplied to the user. Unfortunately, this definition also has several flaws. There exists the possibility that the software *does* behave according to the official publications but errors are present because both the software and the publications are in error. A second problem occurs because of the tendency of user publications to describe only the manual for a time-sharing system that states, "To enter a new command press the attention key once and type the command." Suppose that a user presses the attention key twice by accident and the software system fails because its designers did not plan for this condition. The system obviously contains an error, but we cannot really state that the system is not behaving according to its publications. [¶] The last definition that is sometimes used defines an error as a failure of the software to perform according to the original contract or documentation. Although this definition is an improvement over the previous three, it still has several flaws . . . written user requirements are rarely detailed enough to describe the desired behavior of the software under all possible circumstances. [¶] There is, however, a reasonable definition of a software error that solves the aforementioned problems: *A software error is present when the software does not do what the user reasonably expects it to do."*

Roetzheim, William H. (1991) *Developing Software to Government Standards* Prentice-Hall, p. 146

> "Software defects can be divided into four broad categories: (1) requirements defects, (2) design defects, (3) code defects, and (4) documentation defects." See also Dunn, Robert (1984) *Software Defect Removal*, McGraw-Hill, p. 6-7 for the same distinctions.

Notes

Law of Software Quality

Section 8.

Contracts:
Breach of Warranty

Contracts: Warranties

Express Warranty

A warranty is a statement of fact, either articulated or implied by law, respecting the quality or character of the goods to be sold.

Express Warranties are defined in the Uniform Commercial Code (2-313). *Under the Uniform Commercial Code an express warranty is:*

2-313(a) Any affirmation of fact or promise made by the seller to the buyer which relates to the goods and becomes part of the basis of the bargain . . .

2-313(b) Any description of the goods which is made part of the basis of the bargain . . .

2-313(c) Any sample or model which is made part of the basis of the bargain.

Contracts: Warranties

Implied Warranties

Under U.C.C. 2-314, a warranty that goods are merchantable is implied in a contract for their sale.

- ☒Merchantability requires that the program do what a reasonable customer would expect it to do (and that it be salably packaged).
- ☒The seller can exclude the warranty, but it must be done correctly.

California Civil Code 1792.4 *(a) No sale of goods . . ., on an "as is" . . . basis, shall be effective to disclaim the implied warranty of merchantability . . . unless a conspicuous writing is attached to the goods which clearly informs the [consumer], prior to the sale, in simple and concise language.*

Contracts: Warranties

Warranty Disclaimers

NoName Software warrants the diskettes on which the programs are furnished to be free from defects in the materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by your proof of purchase.

The entire liability of NoName Software, and your exclusive remedy, shall be replacement of any diskette which does not meet the Limited Warranty and which is returned freight prepaid, to NoName Software. NoName Software does not warrant that the functions contained in the program will meet your requirements or that the operation of the programs will be uninterrupted or error-free. THE PROGRAMS CONTAINED IN THIS PACKAGE ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK RELATED TO THE QUALITY AND PERFORMANCE OF THE PROGRAMS IS ON YOU. IN THE EVENT THERE IS ANY DEFECT, YOU ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION. IN NO EVENT SHALL NoName BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGE, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

This Agreement constitutes the complete and exclusive statement of the terms of the agreement between you and NoName Software. It supercedes and replaces any previous written or oral agreements and communications relating to this software. No oral or written information or advice given by NoName Software, its dealers, distributors, agents or employees will create any warranty or in any way increase the scope of the warranty provided in this agreement, and you may not rely on any such information or advice.

Contracts: Warranties

Documentation is an Express Warranty

You can't disclaim an express warranty -- you are accountable for your claims.

Uniform Commercial Code 2-316 (1):

> Words or conduct relevant to the creation of an express warranty and words or conduct tending to negate or limit warranty shall be construed whenever reasonable as consistent with each other; but . . . negation or limitation is inoperative to the extent that such construction is unreasonable.

Contracts: Warranties

Testing Documentation: Why is it Important?

Errors in the manual increase risks of warranty liability.

Additionally, testing the documentation improves the reliability of the program.

Contracts: Warranties

Warranties & Misrepresentations: What Must You Test?

Advertisements

Published specifications

Interviews

Box copy

Fax-backs

Manual

Help system

Warranty

Web pages

Readme

Advice given to customers on Internet, CompuServe or AOL

Contracts: Warranties

Disclaimer Analysis: Limit Duration of Warranties

"NoName Software warrants the diskettes on which the programs are furnished to be free from defects in the materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by your proof of purchase."

The Magnuson-Moss Warranty -- Federal Trade Commission Improvement Act (15 USC 2308) allows warrantors to limit implied warranties to the same duration as express warranties as long as the duration is reasonable and not unconscionable.

The Song-Beverly Act sets a minimum of 60 days.

Contracts: Warranties

Shrink-Wrapped Disclaimers

Shrink-wrapped warranty disclaimers are ineffective because they are hidden. This is not a special problem for software; it is standard UCC practice. See, for example, B. Clark & C. Smith, *The Law of Product Warranties*, Warren, Gorham & Lamont, section 4.03.

- ☒Conspicuousness avoids post-sale change of the contract
- ☒Conspicuousness is an incentive for competition

FTC data: huge percentage of customers pay attention to warranty terms when they comparison shop. Claims that consumers ignore warranty terms are simply incorrect.

Contracts: Warranties

Shrink-Wrap Warranty Disclaimer Ruled Invalid

In the case of *Step-Saver Data Systems, Inc. v. Wyse Technology and The Software Link, Inc*., the United States Court of Appeals for the Third Circuit held that a disclaimer of all express and implied warranties, printed on the outside of the box, was not binding on a mail order customer.

Step-Saver repeatedly bought Multilink Advanced, an allegedly MS-DOS compatible operating system, from The Software Link (TSL). On each box was a disclaimer: the software was sold AS IS, without warranty; TSL disclaimed all express and implied warranties; and a purchaser who didn't agree to this disclaimer should return the product, unopened, to TSL for a refund.Step-Saver sued TSL, claiming that Multilink Advanced was not MS-DOS compatible.

TSL argued that Step-Saver had accepted the terms of the warranty disclaimer when it opened each package, and therefore Step-Saver could not sue.

Step-Saver made each purchase by telephone. The Court ruled that the essential terms of the sale (such as price and quantity) were set out during the calls. The warranty disclaimer on the box arrived later. Under Section 2-207 of the Uniform Commercial Code, the disclaimer was merely a proposal by TSL to add a term to the contract. Step-Saver was not required to accept this proposal. Having already bought the product, Step-Saver could open it and use it without agreeing to this new warranty disclaimer. Nor did it matter that Step-Saver placed additional orders after seeing the disclaimer. TSL never insisted that Step-Saver agree to the disclaimer during purchase negotiations (the telephone calls), therefore the disclaimer was not part of any contract.

The Court said:

> TSL has raised a number of public policy arguments focusing on the effect on the software industry of an adverse holding concerning the enforceability of the box-top license. We are not persuaded that requiring software companies to stand behind representations concerning their products will inevitably destroy the software industry.

Contracts: Warranties

Additional Lecture Notes on Shrink-Wrap Warranty

Arizona Retail Systems v. The Software Link: 831 F. Supp. 759 (D. Arizona, 1993). Similar facts to Step-Saver Data, but --- ARS initially ordered a demo copy of the product. The sale didn't become final until 30 days had passed or ARS opened the shrinkwrap on the full (not demo) version of the program. The disclaimer was on the outside of the shrinkwrap. Then ARS ordered several more, in lots of 20(?). The court ruled that:

- ☒ As to the first purchase, the disclaimer was valid because it was conspicuously made part of the terms of the purchase.
- ☒ As to the later orders, by phone, the disclaimer was not part of the bargain. TSL did not demand this during the telephone orders, and shipped without getting this agreement. Therefore it was not part of the contract.

The most interesting part of the case was the briefs. TSL's reply brief's strongest citation was to McCrimmon v. Tandy Corp. 313 SE.2d 15 (Ga 1991). Here, though, the disclaimer was conspicuously on the sales invoice that was handed to the customer at the cash register. The customer received it at the time of sale, before he walked out the door, and the court held that this made it part of the original purchase agreement.

Frank M. Booth, Inc. v Reynolds Metals Co., No. Civ. S-89-0048 (DFL) (E.D. Cal. January 9, 1991) and Diamond Fruit Growers, Inc. v. Krack Corp. 794 F.2d 1440 (9th Cir. 1986). Both cases involved a warranty disclaimer by the seller on the seller's purchase order confirmation, and there was a different set of warranty provisions on the buyer's purchase order. The seller (Reynolds) said in the paper that seller's acceptance of the deal was expressly conditioned on the other party's agreement to the terms and conditions of the seller's form, and if the purchaser didn't respond within 10 days, it would be deemed to have agreed. The court rejected this argument, saying that *If a seller truly doesn't want to be bound unless the buyer assents to the terms of the seller's agreement, the buyer can protect itself by refusing to deliver the product until the purchaser agrees to the agreement's terms.*

Results of a Bad Disclaimer

Remember the story of the child who tried to take all of the cookies out of the cookie jar, and therefore got none?

- ☒ False disclaimer of express warranties. (Worse, this most hurts customers who *trust* the seller.)
- ☒ Hidden, post-sale, disclaimer of implied warranty of merchantability.
- ☒ Refusal to take responsibility for statements made by seller's staff.

Potential Additional Costs:

- ☒ 4-year warranty
- ☒ No remedy limitations

Cem Kaner. Ph.D.. J.D. **P.O. Box 1200**
Attornev at Law **Santa Clara. CA 95052**
kaner@kaner.com **408-244-7000**

LIABILITY FOR DEFECTIVE DOCUMENTATION

Published in *Software QA*, Volume 2, #3, 1995, p. 8.

In October, 1985, W.H. Daughtrey bought a diamond bracelet as a Christmas gift for his wife from Sidney Ashe, a jeweler[1]. He paid $15,000. After Daughtrey agreed to buy the bracelet, Ashe filled out an appraisal form and put it in the box with the bracelet. The appraisal said that the diamonds were of *v.v.s. quality* (a high grade). Daughtrey didn't see the appraisal until later, probably not until the box was opened at Christmas.

In 1989, Daughtrey discovered that the diamonds were not of v.v.s. quality. Ashe offered a refund. Daughtrey refused, and demanded that the diamonds in the bracelet be replaced with diamonds that were of v.v.s. quality. Ashe refused. Daughtrey sued. He said that the statement that the diamonds were of v.v.s. quality was a description of the goods by the seller. According to the Uniform Commercial Code,

> 2-313(b) any description of the goods which is made part of the basis of the bargain creates an express warranty that the goods shall conform to the description.

Therefore, Daughtrey said, Ashe created a warranty that the diamonds were of v.v.s. quality, and breached it by selling a bracelet whose diamonds were of a lower grade. Ashe argued that this claim couldn't have been a warranty because he never called it a warranty and Daughtrey didn't read the claim until long after the sale. How could this description be part of the "basis of the bargain"?

Ashe won -- in the trial court. But the Supreme Court of Virginia overruled the trial court. Quoting the Official Comments to the Uniform Commercial Code, the Court said:

> The whole purpose of the law of warranty is to determine what it is that the seller has in essence agreed to sell[2].

and

> The precise time when words of description or affirmation are made . . . is not material. The sole question is whether the language is fairly to be regarded as part of the contract.[3]

The Court concluded that Ashe had agreed to sell v.v.s. quality diamonds, and therefore that he had breached the sales contract by selling inferior diamonds.

---

[1] *Daughtrey v. Ashe* (1992) South Eastern Reporter, Second Series, volume 413, p. 336 (Supreme Court of Virginia).
[2] *Daughtrey v. Ashe* (1992) p. 339.
[3] *Daughtrey v. Ashe* (1992) p. 339.

So what does this have to do with computer software?

When a customer buys a computer program at a store, there is probably a manual in the box. Just as there was an appraisal in the bracelet box. And the manual makes specific descriptive statements about the program. Just as the appraisal made a specific descriptive statement about t bracelet. You might not read the manual until long after you've bought the program. Just as Daughtrey didn't know anything about the appraisal until after the purchase. A judge in Virginia would probably treat statements in the manual as warranties[1]. The seller of the program would be liable for breach of warranty if the software didn't do what the manual stated.

Customer Care, Inc. publishes a survey every year, *the Customer Care Survey: Service & Support Practices in the Software Industry*[2]. In the 1994 survey, page V-29, 50% of the responding companies said that they don't put their manuals through Quality Assurance. I'm writing this article to tell you that if you're not testing your documentation, you're making a big mistake.

If your manual or your help or your packaging say false things about the program, your company is risking lawsuits for breach of warranty, for deceptive trade practices, and for misrepresentation[3].

Lawyers' Tricks

Your company's lawyers look for ways to protect your company from lawsuits. There are some thin that they can do to reduce your company's exposure. But if your company is at fault, they can't be expected to win every time.

For example, your lawyer can argue that your manual can't be interpreted as a warranty unle the customer read it and relied on its statements when making the decision to buy. That argument will work in a few States. Variations on it will work in most States[4]. But this won't work in other States, such as Virginia. And in most States, the argument will be very difficult if the customer dic flip through the manual before buying the program[5]. What will work, in every State, is making sure that the manual describes the program accurately. Your lawyer can't do that. But you can.

---

[1] Rather than relying on subsection (b) of Section 2-313 of the Uniform Commercial Code, whi says that a description of the goods is a warranty, the Court might use subsection (a) which says that "Any affirmation of fact or promise made by the seller to the buyer which relates to the goods and becomes part of the basis of the bargain creates an express warranty that the goods shall conform to the affirmation or promise." Manuals are large, detailed collections of statements (affirmations) of fact about the program that they document.

[2] Customer Care, Inc., 235 Martling Ave., Tarrytown, NY 10591.

[3] In the case of *Burroughs Corporation v. Hall Affiliates* (1992) Southern Reporter, Second Serie volume 423, p. 1348 (Supreme Court of Alabama), Burroughs was held liable for fraud on the basis of false statements from its sales representative to a customer. There was no claim that the salesperson *deliberately* misled the customer. Alabama and several other States allow customers to hold sellers liable for innocent and negligent misrepresentation, as well as for deliberate fraud.

[4] The laws vary across States. Good discussions are in Clark, B. & Smith, C. (1984, supplement 1994), *The Law of Product Warranties*, Warren Gorham & Lamont, and Rosmarin, Y.W. & Sheldon, J. (1989, supplemented 1994), *Sales of Goods and Services* (2nd Ed.), National Consume Law Center.

[5] If the customer looks through the manual before the sale, the court is likely to analyze the manual in the same way as it would analyze advertising materials, such as brochures. Statements of fact in brochures are generally treated as warranties. Published cases involving manuals are rare, but a customer did examine an operator's manual before buying a hay baler the case of *Schlenz v. John Deere Co* (1981), Uniform Commercial Code Reporting Service, volume 31, p. 1020 (United States District Court, District of Montana). Statements in the mar were held to have created an express warranty of safety.

The shrink-wrapped warranty disclaimer is another trick we're all familiar with. This is the piece of paper that some software companies put inside the box, that says that the program is sold "As Is" with no warranty. I'll probably write more about this in a later column[1] but the conclusion is a simple one. These disclaimers may convince people who don't know better that they have no rights, but they are not in accordance with the Uniform Commercial Code. American courts throw them out.[2] These disclaimers also fare badly outside of the United States.[3] Several software publishers now take a different approach. They give purchasers an honest warranty that the software will perform substantially in accordance with the packaging and documentation. Other software publishers provide a satisfaction guarantee and let dissatisfied purchasers return the product for a full refund within 30 days, 90 days or even a year.

In short, rather than relying on your lawyers to bail you out, you are better off making sure that your manual, help, and packaging match the program.

Bad Documentation and the Cost of Quality

The cost of quality associated with a product includes the cost of investments that your company makes in developing a high quality product, and the expenses it suffers in dealing with the failings of the product.[4] Examples of quality-related costs include money spent on code inspections, design reviews, defensive programming, fixing bugs, and answering customer complaints.

One of the key goals of quality engineering is reduction of the total cost of quality of a product. For example, it is often cost-effective to spend more money preventing bugs, thereby reducing the amount needed to support the product in the field. Successful problem prevention can reduce the total cost of quality.

It costs money to thoroughly test a manual. I'll explain what I do, to thoroughly test a manual, later. Here, the issue is money. My rule of thumb is that it takes about 15 tester-minutes per page of the manual. I haven't seen thorough tests go much faster than this. In one extreme case involving a particularly buggy program and a problematic manual, I spent about an hour per page on the manual.

You might test two or three drafts of a manual. It usually costs less to test a page for the second time, but I don't budget for less than 7.5 minutes per page.

Suppose that you spend a total of 100 tester-hours testing three drafts of a 200-page manual. Add 20% for administrative overhead (meetings, status reports, etc.) and suppose that your overhead+benefits+salary-weighted cost is $50 per hour per tester. Testing the manual cost $6000 over three tester-weeks. This is a significant quality-related expenditure. Some managers automatically say "too much, don't do it" in the face of this proposed expense.

---

[1] If you urgently need a detailed discussion from me now, I can send you a few pages from a draft of David Pels' and my forthcoming book, *Bad Software: Get Treated Fairly When You Buy Computer Software*

[2] *Step-Saver Data Systems, Inc. v. Wyse Technology and The Software Link, Inc.* (1991), Federal Reporter, Second Series, volume 939, p. 91 (United States Court of Appeals for the Third Circuit) and *Arizona Retail Systems, Inc. v. The Software Link* (1993) Federal Supplement, volume 831, p. 759 (United States District Court, District of Arizona).

[3] Lemley, M. (1995) "Intellectual Property and Shrinkwrap Licenses," *Southern California Law Review, volume 68*, page 1253, reviewed court cases and statutory law in 34 countries.

[4] Campanella, J. (Ed.) (1990) *Principles of Quality Costs: Principles, Implementation and Use* (2nd Ed.) American Society for Quality Control, Quality Press.

To decide whether it really costs too much to test the manual, do some quality engineering. Wh does your company save by making this three week, $6000 investment? Here is part of that analysis for a software publishing company. When you think carefully about your own company, you'll probably be able to add several additional internal benefits that a good manual provides in your company, and several additional risks that an error-prone manual poses for your company.

Within Product Development (including the Testing Group), the manual might serve several functions, such as the following:

It's a test plan. It takes you on a tour of the entire program. No matter how carefully you think you've tested, a thorough test of the manual against the program is likely to highlight problems that you missed. Apart from its effectiveness as a revealer of new bugs, the manual is a valuable test plan because of the credibility it provides for bug reports. Many bugs that you find initially look small, and are deferred. But if one of these bugs is exposed again by following the instructions or suggestions in the manual, you should re-open it, or re-report it. The bug will appear much more significant if you can run across it by doing simple things th any user would do (such as following the instructions in the manual.)

It's a training tool. New programmers and testers who join the project fairly late use the manual to learn about the program. Errors in the manual confuse them and can result in ne software errors or unproductive testing.

It's the external specification Many product development groups stop maintaining the external specification (if they ever maintained it) (assuming they ever wrote one), and rely o the manual to serve this purpose. Errors in the manual cause all of the problems that you ge from errors in the specification.

Here are some of the problems that an error-ridden manual causes the Marketing and Sales departments:

Delayed collaterals. Before the product ships, your company probably creates brochures, application notes, output samples, and several other materials that highlight what the program can do. The people who create these are probably not experts with the program. The probably rely on the most recent draft of the manual for guidance. Combine a few errors in this draft of the manual with a few bugs in this pre-release version of the program, and it ca take several days to produce output that should have been done in an hour.

Delayed packaging. The Marketing staff probably create sample output that will be photographed and put onto the program's box. As with the collaterals, bad documentation ca delay this work. Also, they'll probably use the manual as source material for box copy -- all those descriptions of what the product can do, what equipment it's compatible with, etc. Erro in the manual can result in errors on the box. If your company is lucky you'll discover these before the box goes to the printer, so the cost is just the cost of delay, not reprinting or stickering the box. These delays, however, can be very expensive. Your company can save money, if it is manufacturing thousands of boxes, by booking time with a printer well in advance. However, if you did this, but your company doesn't get your artwork to the printer time, it might have to pay for that reserved press time, along with having to pay extra for las minute printing arrangements. These expenses can dwarf that $6000 testing cost.

Erroneous ad copy. Claims in the manual can turn into claims in the advertising materials. Errors in the manual turn into errors in advertisements. Now you have to spend money on corrective advertising. And some people will accuse you of false advertising or will want to treat your ads as warranties. You will discover the hard way that $6000 buys more tester-hours than lawyer-hours.

Problems from pre-sale use of the manual. The manual might be available at trade shows, or sent to user groups or prospective customers on request. People who review your product for magazines will rely on the manual for information and instructions. In each of these cases, errors in the manual can be embarrassing and expensive, and dealing with the can waste the time of your senior Marketing or Sales staff.

Here are some of the problems that a weak manual will cause your Training staff.

Your customers will need more training If you provide training with your product, more of your customers will need the training if your manual is bad. If you provide "free" training your training costs rise. You train more people, and you probably have to train them for mo hours per student. If you charge for training, your customers will be aware of all the extra money they're spending to be able to use your product.

Your Training department might write its own manual I've seen this several times. The Training staff decide that they don't like the manual, so they write their own and give i to their students. Along with the expense of duplicated effort, this creates new testing requirements and new opportunities for providing incorrect claims to your customers.

Training sessions can become adversarial If the manual has errors, some students will quote them to the Trainer during the session. When the Trainer says X, the student says "B the manual says Y. Why does it say that?" This can be disruptive and embarrassing, especially if it happens several times. It reflects badly on the product and on your company, not just on the Trainer.

And finally, here are some of the problems that errors in the manual will cause your technical support and field support staff:

It takes longer and costs more to prepare answer books As part of preparation for release of a new product, a well-organized Support staff prepares books or a database of answers to the questions that they expect will be frequently asked, or that will be difficult t answer. Errors in the manual make this task harder and more prone to error.

It makes calls run longer. Sometimes the best way to handle a call is to alert the custome to the relevant section in the manual. This can often be done without insulting the custome and if that section provides a clear, detailed answer to a complex question, it might be just what the customer needs. A question that would take an hour of handholding and explanation over the phone becomes an easy two-minute call. On the other hand, if the manual is untrustworthy, Support can't push the customer back to the manual. Calls that should be short become long.

It results in more calls If the manual says that the program works a certain way, and the program doesn't work that way, people will call and ask what the problem is. Along with generating calls about the errors, errors in the manual teach some people that the program (or the manual) is untrustworthy. Therefore they feel more justified in calling more often ar more quickly for more help, and in demanding that the help be provided free of charge because, after all, the program is full of errors.

It results in more difficult calls Over the past two years, I've interviewed several Suppor mangers and staff as part of my research for a new book. One of the common threads was th difficulty of dealing with people who were misled by the manual. Imagine dealing with someone who followed the instructions but wasn't able to achieve the result that the manua promised, or worse, who followed the instructions but lost data or got into trouble. These ca are especially hard if you don't have an easy solution to the callers' problems. Calls like thes burn Support staff out. People quit and your company has to hire and train new staff. This i an outrageously expensive cost of a low quality manual.

As a Test Group Manager and as a Documentation Group Manager, I've run into management resistance to doing what was required to test the manuals thoroughly. I respond with a rough sketc of quality-related costs and risks. I lay out the near-term benefits of good testing and the near-term and long-term risks of poor testing of the manual. I present this to my manager, and to my manage manager's manager if necessary.

This is persuasive stuff. To people who care about quality, you are talking about customer satisfaction. To people who care about schedules, you are talking about efficiency -- a well-tested manual helps you find bugs sooner and train Development, Marketing and Support staff faster. To people who care about money, you are spending a little to save a lot. To people who are afraid of making decisions, you are providing a business case that makes this decision look safe and obvious. Finally, note that when you identify the in-house staff who will benefit from a thorough test of the manual, you create a list of people who have a stake in lobbying their management to make sure th you have the budget you need to do this task well. Go talk to them.

Doing the Testing

When you test the manual, you should use the program exactly as the manual says. Try every example. Verify every Note, every Caution, every Tip. Try every suggestion. Check every explanati of every error message. Check every definition in the manual's Glossary to be sure that they all ma sense in the context of this program.

Check every limit (such as a claim that the program can store 100,000 records) and ask whethe the program's behavior is reasonable at the limit. For example, if a program can store 100,000 records, don't just generate 100,000 records and see if it holds them all. How long does it take to retrieve one or to enter one as you approach this limit? It might be reasonable for the program to slow down as the database gets bigger, but if adding each new record takes 36 hours of computer time, you know that the manual should be setting readers' expectations at a lower number, or there will be significant dissatisfaction.

Limits aren't the only claims made about the program's capabilities. Check all the other claims too. Be skeptical. If the manual says that printed output is gorgeous on all supported printers, chec it with an old 9-pin dot matrix, or some other printer that is likely to provide less gorgeous output than your samples in the manual. Would the owner of that printer agree that you were getting as gorgeous a result as you could get from that printer?

Every time you see a mismatch between the program and the manual, note the mismatch on tl manuscript (for the writer) and file a bug report. The bug report says that the manual and the program differ, and asks which one is correct. Close the report as Fixed when either the program o the manual is changed to eliminate the mismatch.

When you see outdated information in the manual draft, give the writer your notes on the rece design changes to the program.

When it appears that the writer didn't understand the purpose of a feature, try to provide the writer with an explanation (if you can do it quickly).

If a section of the manual is confusing to read, test this area of the program. This is a sign that this part of the program's design is probably too complex. The area's bug count will probably be hig This is an example of an important theme -- let the manual suggest to you areas of the program tha will need more intense testing.

When you are testing Help, you have additional concerns, because the Help is coded with jum and branches, just like any other program. It can have bugs, maybe serious ones. You have to test these, along with the content. My experience is that moderately thorough Help testing can easily take twice as long as testing of the manual.

Along with testing Help and the manual, you should be testing box copy, brochure copy, and other information that the company prepares in order to send to many customers.

In Closing

Thoroughly testing your documentation might be the most cost-effective thing you can do to significantly reduce the probability that your company will be sued. It's also an effective way to find bugs in the program, a strong assistance to the writers, and a valuable source of information for you company's in-house users of the not-yet-released program. So many companies fail to take the accuracy of their documentation seriously that you may have to educate your company before they'll grant you the time you need to do this job properly. Think your case through in terms of quality-related costs, and schedule/efficiency-related benefits. A strong analysis and presentation along thes lines will probably get you the approval you need for the testing, and enhance your credibility as a business decision-maker at the same time.

Notes

Law of Software Quality

Section 9.

Contracts: Damage Control

Contracts: Damage Control

Limiting the Risks

In commercial contracts, buyers and sellers are free to reallocate risks associated with breach of contract. They can agree:

- ☒ To limit the liability of the breaching party
- ☒ To limit one side's legal expenses by requiring that lawsuits be brought in that party's home state, under its laws
- ☒ To limit the length of time in which the aggrieved party can sue the breaching party
- ☒ Etc.

The provisions are suspect in non-negotiated contracts, and are particularly suspect in contracts that cannot be seen before the sale.

UCC Article 2 allows all customers to reject post-sale material modifications to a contract.

Contracts: Damage Control

Restrict Remedies & Limit Liability Under the U.C.C.

- ☒Restrict remedy for defective product to repair, replacement or refund
- ☒Limit liability to direct damages (no consequential damages)
- ☒These are viable if customer is a business, but not in some consumer cases (e.g. personal injury).

Limitation can fail if court determines that seller provided absolutely no remedy (remedy in contract failed completely) or if the court determines that the result doesn't provide a "minimum adequate remedy."

Contracts: Damage Control

Allocating Risk of Tort Loss

U.C.C. 2-719 (3) Consequential damages may be limited or excluded unless the limitation or exclusion is unconscionable. Limitation of consequential damages for injury to the person in the case of consumer goods is prima facie unconscionable but limitation of damages where the loss is commercial is not.

Contracts: Damage Control

Uniform Computer Information Transactions Act

- ☒Will govern all contracts involving software and digitally stored information.
- ☒Opt-in clauses can bring in goods sold with software.
- ☒Was proposed as an amendment to the Uniform Commercial Code, known as Article 2B, but American Law Institute withdrew from the drafting project after calling for “fundamental revision”, knocking it out as a UCC amendment.

- ☒Current draft of UCITA at www.law.upenn.edu/bll/ulc/ulc.htm

Proponents of UCITA

Software publishers

Database publishers (West / Lexis / NASDAC)

CitiBank

Daimler Chrysler

National Conference of Commissioners on Uniform State Laws

Opponents of UCITA

See your notes for a long list. Some examples:

- ☒Consumers
- ☒Insurance companies
- ☒Librarians
- ☒Staff of the Federal Trade Commission
- ☒25 Attorneys General
- ☒American Intellectual Property Law Assoc and IP section of the NY City Bar Assoc
- ☒Motion Picture Assoc, Newspaper Assoc, Magazine Publishers
- ☒Software developers

Software developers *oppose UCITA?*

Software publishers might think that UCITA is peachy but UCITA is opposed by every software developers' association that has spoken on the matter, including:

- ☒American Society for Quality Software Division
- ☒Association for Computing Machinery
- ☒Independent Computer Consultants Association
- ☒Institute for Electrical & Electronic Engineers

UCITA threatens our professionalism, our drive to improve our craft and our products, and the satisfaction of our customers.

UCITA creates disincentives for quality

By cost-reducing consequences of shipping bad software

- ☒Cost of support is recoverable from customer
- ☒Litigation is almost impossible
- ☒Remedies are minimal

By helping publishers limit presale competitive information

- ☒Hiding terms limits presale comparison shopping
- ☒Use restrictions kill critical magazine reviews

By allowing publishers to discourage competition

- ☒Eliminate competition from used software
- ☒Ban mass-market reverse engineering except for DMCA exceptions
- ☒Limit competitive use of provided information

UCITA: No accountability for known defects

Most defects in mass-market software are known at time of release or soon thereafter, but many are left undisclosed (let alone, unfixed.)

- ☒Vendor can exclude incidentals and consequentials. Incidentals include cost of reporting the defect and returning the product.
- ☒Vendor can charge fees for support (e.g. $5 per minute). This is an incidental expense.
- ☒Vendor can ship a known defect, then charge for support call, agree to give a refund (because of the defect) but keep the support fee and not reimburse the shipping cost.

ACM / IEEE / ICCA / Nader alternative proposal: default to no consequentials UNLESS there is an undisclosed known defect, then make damages non-excludable.

UCITA: Eliminates Article 2 safeguards on remedy limitations.

Eliminates (see comment 6 to section 803) the Article 2 provision for a minimum adequate remedy.

Eliminates the doctrine of failure of essential purpose of a limited remedy by expressly permitting boilerplate to preserve exclusion of incidental and consequential damages even when an agreed exclusive remedy fails or is unconscionable.

UCITA: Permits elimination of the right to cancel

Section 803(a)(1) contains language not found in Article 2 permitting a limited remedy "precluding a party's right to cancel for breach of contract." This seems to permit boilerplate to eliminate the right to refuse a tender that does not conform to the contract, thus effectively undermining the perfect tender rule supposedly established for mass-market transactions in Section 704(b). See also Section 802(d), referring to terms prohibiting cancellation.

UCITA: Narrowly defines material breaches

UCITA 701 (b) A breach of contract is material if:

(1) the contract so provides;

(2) the breach is a substantial failure to perform a term that is an essential element of the agreement; or

(3) the circumstances, including the language of the agreement, the reasonable expectations of the parties, the standards and practices of the business, trade, or industry, or the character of the breach, indicate that:

(A) the breach caused or is likely to cause substantial harm to the aggrieved party; or

(B) the breach substantially deprived or is likely substantially to deprive the aggrieved party of a significant benefit it reasonably expected under the contract.

This is allegedly based on the Restatement of Contracts.

UCITA: Narrowly defines material breaches

Compare UCITA to the Restatement (Second) of Contracts § 241 (1981), which lists five factors:

1) the extent to which the injured party will be deprived of the benefit he or she reasonably expected;
2) the extent to which the injured party can be adequately compensated for the benefit of which the party will be deprived;
3) the extent to which the party failing to perform or to offer to perform will suffer forfeiture;
4) the likelihood that the party failing to perform or to offer to perform will cure the failure, taking into account all the circumstances, including any reasonable assurances; and
5) the extent to which the behavior of the party failing to perform or to offer to perform comports with standards of good faith and fair dealing.

UCITA creates disincentives for quality

By cost-reducing consequences of shipping bad software

By helping publishers limit presale competitive information

- ☒Hiding terms limits presale comparison shopping
- ☒Use restrictions kill critical magazine reviews

By allowing publishers to discourage competition

UCITA: Validates post-payment disclosure of terms

- Permits “pay first, see the contract terms later” **even for such key terms as warranty disclaimers and remedy limitations**: UCITA defines “opportunity to review” in such a way as to provide that a customer who doesn’t see terms until after he or she has paid and taken delivery of the information is deemed to have an opportunity to review those terms. Section 112(e)(3). There is no exception for terms required to be conspicuous, so that a term is “conspicuous” even when first disclosed after payment or delivery. See Section 406 for conspicuousness requirement as to warranty disclaimers.
- This flies in the face of nearly a century of jurisprudence on post-sale presentation of disclaimers of warranties. (Finding the disclaimers ineffective in industry after industry).

UCITA: Validates post-payment disclosure of terms

- This makes comparison shopping (one of the great potential benefits of on-line shopping to consumers) impractical. Delay of disclosure of terms until after a customer is psychologically committed to the deal is the approach used in UCITA for all terms—even important elements of the deal such as warranty disclaimers, remedy limitations, transfer restrictions, prohibitions on criticism of the product, and the key feature of a license—the restrictions on the number of users and the length of time that use is authorized.
- Post-payment disclosure also makes it hard for journalists to gather information about the best available deals and present comparative

UCITA: Validates post-payment disclosure of terms

UCITA contains no requirement that terms of the contract be made available before payment is accepted and delivery is made, even when it would be easy to do so.

Instead, UCITA gives the licensor (or the seller of a combination of goods and software) unfettered discretion to decide to provide the customer with the terms before or after the customer pays and receives shipment. Section 112(e)(3).

UCITA: Probably eliminates coverage of software by goods-related consumer protection laws

Statutes like the California Song-Beverly Act and the federal Magnuson-Moss Warranty Act apply specifically to sales of goods.

Courts typically treat mass-market software transactions as sales of goods. UCITA recharacterizes these transactions as "licenses" of "computer information." Section 102(a)(40) and (10). Licenses are intangibles, not goods, and therefore goods-specific laws will no longer apply.

Note Fred Miller's comment, in UCC Bulletin, that Mag-Moss was never intended to apply to software.

Therefore, the supposed preservation of consumer protection law in Section 105(c) is misleading.

UCITA should provide that consumer protection laws that apply to sales of goods also apply to licenses of software in the mass market.

UCITA: Right of return is illusory

- The right of return touted by UCITA's sponsors evaporates the moment the consumer double clicks on the "I agree" screen. Section 209(b). Many software companies have included this right of return in the absence of UCITA, and they know that a return right is rarely invoked by customers who have already paid or taken delivery, and who are anxious to get access to the product and must click to do so.
- Additionally, UCITA's right of return disappears if the licensee has any opportunity to review the license before becoming obligated to pay. Section 209(b).
- A genuine right of return would allow the consumer to install the product and evaluate it for a limited but reasonable time, with a right to return the product either because of obvious defects or because of (in the context of the perceived quality of the product) the contract terms.

UCITA: ALI withdrew from UCC 2B

The authors of a May 1998 ALI resolution (Braucher and Linzer) wrote in their supporting memo:

- "The Draft reflects a persistent bias in favor of those who draft standard forms, most commonly licensors. It would validate practices that involve post-purchase presentation of terms in both business and consumer transactions (using "shrink-wrap" and "clickwrap"), **undermining the development of competition in contingent terms**, such as warranties and remedies. It would also allow imposition of terms outside the range of reasonable expectations and permit routine contractual restrictions on uses of information traditionally protected by federal intellectual property law. **A fundamental change**

UCITA: Fundamental conflict with UDAP disclosure policies

- Another problem with UCITA is that it conflicts with the approach of statutes prohibiting unfair and deceptive practices. Many cases and regulations apply these statutes so as to require early disclosure of key elements of transactions—early and prominent disclosure of key terms is crucial to an efficient marketplace based on meaningful consumer choice.
- By specifically authorizing post-payment disclosure of terms, UCITA would have one of two effects: misleading producers into thinking that this approach is legally protected, or watering down anti- deception laws by influencing interpretation of them to permit delayed disclosure.

UCITA:Weakens Article 2 standard for warranty by demonstration.

- Permits a product to fail to fully conform to a sample, model, or demonstration even when that sample, model, or demonstration was part of the basis of the bargain and created an express warranty: Under Section 402(a)(3), the actual product need only “reasonably conform" to the sample, model, or demonstration
- Eliminates some express warranties created by a display or description of a portion of the information: Under UCITA, a display or description of a portion of information doesn’t create an express warranty if the purpose was: “to illustrate the aesthetics, market appeal, or the like, of informational content." In other words, the licensor can show or describe the information, but the information doesn't have to fully live up to that display or description. Section 402(b)(2). No similar restriction is found in UCC Article 2

UCITA: Current IP law

- Copyright Act is federal law. Supercedes state laws that try to govern copying or distribution of original works.
- Copyright Act balances rights of creators / publishers and buyers.
  - First sale doctrine
    - » Buyer of a copy may lend, resell, destroy, or mark up her copy. The seller's rights to that particular copy are exhausted in the sale.
  - Fair use rights: limited copying allowed for
    - » reviews, parody
    - » classroom use
    - » reverse engineering

UCITA: End of Fair Use -- Allows unreasonable use restrictions

102(a) (19) "Contractual use term" means an enforceable term that defines or limits the use, disclosure of, or access to licensed information or informational rights, including a term that defines the scope of a license.

307(b) If a license expressly limits use of the information or informational rights, use in any other manner is a breach of contract.

UCITA: Allows restrictions on public discussion of product flaws.

UCITA explicitly validates use terms, explicitly mentioning nondisclosure restrictions.

We already see software licenses that purport to ban publication of critical articles; at least one trade journal has stated that it decided not to risk being sued under these terms.

Even if the courts eventually ruled that such restrictions on mass market software are against public policy, this will take years to settle through repeated litigation and the effect in the meantime will be to chill public comment on bad products.

UCITA creates disincentives for quality

- By cost-reducing consequences of shipping bad software
- By helping publishers limit presale competitive information
- By allowing publishers to discourage competition
  - ☒Eliminate competition from used software
  - ☒Ban mass-market reverse engineering except for DMCA exceptions
  - ☒Limit competitive use of provided information

UCITA: Validates transfer restrictions in the mass market

UCITA Section 503(2) permits a license to prohibit transfer of software or other information, even if the licensee keeps no copy.

- ☒No more donations to libraries, churches, other charities
- ☒No more gifts of used software to friends and family
- ☒No more used software market
- ☒Consumer who wants to give away or sell a used computer with the operating system can be prohibited from doing so.

This is an example of a contract term that will be valid even though it conflicts with normal consumer expectations (UCITA has no reasonable expectations test for contract terms.)

UCITA creates other problems too

- Electronic commerce rules are unfair
  - ☒Message delivery
  - ☒Notice
  - ☒Writing requirements
  - ☒Security of electronic signatures
  - ☒"Self-help" disabling of customers' software and systems
- Choice of law, forum are wide open to the seller
- Compulsory arbitration fits in the framework.
- MANY other problems for small business.

UCITA: Makes messages effective under unreasonable circumstances

- Information is received (by definition) (102(a)(52)(B)) when it hits the consumer's ISP.
  - ☒If the message is lost, corrupted or filtered between receipt at ISP and failed delivery to consumer's computer, too bad.
- Information is received (by definition) even though it arrived at a system from which a customer cannot access it, as long as the sender does not know that the customer cannot access the information. Section 102(a)(52)(B)(ii)(II).
- Message is effective even if alleged recipient is not aware of receipt. (215)
  - ☒Many consumers have e-mail accounts that they do not check regularly.

UCITA: Creates severe risks for consumers who filter out incoming junk e-mail

Many consumers use filters to screen out spam, such as advertisements of pornography. A reasonably configured filter may delete a message because of its originating ISP, or because of key phrases in the heading, even though that particular message is a legal notice.

Section 102(a)(52) makes it clear that these messages have been received and so are effective even though the consumer will never have seen them.

UCITA: Choice of law and forum in mass-market transactions

- Allows vendor to choose any US forum (and possibly a foreign one) for its convenience. Will deprive many consumers of a forum they can afford by requiring suits to be brought in a remote location
- The boilerplate restriction is enforceable even if the specified forum is unjust or (exclusive or) unreasonable. Section 110.
- Comment 3 provides that a choice of forum “is not invalid simply because it adversely effects one party, even in cases where bargaining power is unequal.”

UCITA: Compulsory consumer arbitration

Not a UCITA case, but applies UCITA reasoning

- Hill v. Gateway 2000, 105 F.3d 1147 (7th Cir. 1997).
  - Computers (not software, just goods).
  - Broadly approves enforcement of terms presented post-sale.
  - Allegations of consumer fraud, racketeering
  - Post-sale contract term (enforced) required arbitration of all disputes, under expensive (ICC not AAA) circumstances.
  - Why would this not apply to arbitration of products liability, if it applies to fraud?
- Boyd v. Homes of Legend, 981 F. Supp. 1423 (M.D. Ala. 1997) applies Gateway reasoning to Mobile Homes

UCITA: Requires balancing test when a contract term violates fundamental public policy:

- UCITA limits the common law discretion of a court to refuse to enforce a contract or a portion of a contract when the contract violates public policy. This doctrine is limited in UCITA to cases where the court finds that the public policy is "fundamental," and then only "to the extent that the interest in enforcement is clearly outweighed by a public policy against enforcement of the terms." Section 105(b).
- If a term violates a fundamental public policy, should the court also have to engage in a process of balancing the interest in enforcing that term against the public policy? The Restatement (Second) of Contracts, in Section 178, calls for balancing, but does not require that the public policy be "fundamental." A further objection to this provision is that it will require decades of litigation to find out what sort of license terms are unenforceable.

UCITA: References

Kaner & Pels, Bad Software: What To Do When Software Fails, John Wiley & Sons, 1998.

www.badsoftware.com (mainly my stuff)

www.2bguide.com (primarily a publisher's-side site, lots of docs from both sides) (but misses some interesting customer-side docs)

www.law.upenn.edu/bll/ulc/ulc.htm

www.nccusl.org

www.infoworld.com (Ed Foster, Gripeline)

Stay tuned for www.ucita.com.

WHY SOFTWARE QUALITY PROFESSIONALS SHOULD ACTIVELY OPPOSE THE UNIFORM COMPUTER INFORMATION TRANSACTIONS ACT

Draft, May 15, 1999

The Uniform Computer Information Transactions Act (UCITA) is a proposed new law that will govern all transactions in software, including contracts for sale, licensing, documentation, maintenance and support of computer software. It will also govern contracts involving electronic information (movies, music, text that you download o buy on a CD) and, at the vendor's option, can govern sales of computers and some other devices that are sold in conjunction with software.

Until recently, UCITA was proposed as an amendment to the Uniform Commercial Code, and was called Article 2B. However, the American Law Institute (ALI), one of the two organizations that must approve all changes to the UCC, recently withdrew from the Article 2B project. The other organization, the National Conference of Commissioners on Uniform State Laws (NCCUSL), decided to rename the bill to the Uniform Computer Information Transactions Act and go forward with it.

==========

SIDEBAR

The American Law Institute has declined to say why it withdrew from the Article 2B project. However, their withdrawal was not a surprise.

In its May 1998 Annual Meeting, the ALI passed the following resolution: “The current draft of proposed UCC Article 2B has not reached an acceptable balance in its provisions concerning assent to standard form records and should be returned to the Drafting Committee for fundamental revision of the several related sections governing assent.”

The authors of the ALI resolution (Braucher and Linzer) wrote in their supporting memo that “The Draft reflects a persistent bias in favor of those who draft standard forms, most commonly licensors." (Companies that publish or sell software are licensors under 2B.)

Additionally, in December 1998, an ALI Council ad hoc committee formed to review Article 2B submitted a memorandum to the full Council stating that it was unlikely that an acceptable draft could be prepared in time for the ALI Annual Meeting in May 1999. The memorandum raised questions about whether the project

is premature in light of rapidly changing technology and business practices. It also noted the lack of consensus about the need for Article 2B and the opposition to it from many affected interests. The memorandum described the drafting of key provisions as “opaque” and “difficult to comprehend.”

For more details, see Braucher, 1999.

==========

NCCUSL will vote on UCITA at its annual meeting, July 23-30, 1999 in Denver. There will be a final UCITA drafting committee meeting on July 22, 1999 in Denver. For details on the meeting, contact me at kaner@kaner.com or check www.nccusl.org.

If you read a copy of this article before July 22, 1999, I urge you to write a letter to the members of NCCUSL from your State, asking them to oppose UCITA. If you read it in the July 20-30 timeframe, you can send a letter to me and I will see that it reaches the NCCUSL members from your state. After July 30, send opposition letters to your state representatives.

Why We Should Actively Oppose UCITA

The simple and short answer is that UCITA will dramatically reduce a software publisher's external failure costs for defective software. It does this brilliantly, in a wide range of ways, reducing the costs of customer support, of lost sales due to competition, and of legal action.

As a result, UCITA changes the economics of software publishing.

When we reduce the risks (to the publisher) of selling defective software, we reduce the incentive to spend the money and time to prevent, search for, and fix defects. In turn, this tells me that we (the American software industry):

- will ship worse software.
- will invest less money in technology and process improvement needed to produce better software.

1. will make the American industry more vulnerable to foreign competition. Les Hatton, a well known author on software quality (see, for example *Safer C*), just finished his masters degree in law. He advises me (personal communication, May 13, 1999) that the trend in Europe is to hold software companies more accountable for defects and to provid greater protection for European consumers and small businesses. I believe that this will provide a greater incentive for companies that primarily trade in Europe to improve their products, rather than ship them with obvious defects. Eventually, international competition will take care of this divergence of standards. But as with the car industry, that eventuality might be devastating for some parts of the American economy (us — software workers — for example).

What We Can Do

For the last three and a half years, I've spent about one-third of my time, unpaid, explaining software quality issues to legislative drafting and regulatory bodies. I've provided input to drafting committees for Uniform Laws (Article 2B/UCITA, Article 2—UCC law of sales, and the Uniform Electronic Transactions Act), to people drafting laws and treaties to govern international electronic commerce (a State Department study group, and members of the American Bar Association), to the Federal Trade Commission, and to various consumer protection groups. Several other software quality advocates have shared in this work, including Watts Humphrey, James Bach, Doug Hoffman, Sharon Marsh Roberts, Melora Svoboda, Ken Pugh, Brian Lawrence, and Bob Johnson. Software-related professional societies, including the Association for Computing Machinery, the Institute for Electrical and Electronics Engineers, the Independent Computer Consultants Association, and the software-test-discuss mailing list (but not ASQ) have submitted letters criticizing UCITA/Article 2B.

Here are a few things that I've learned.

First, UCITA is just one of several legislative proposals involving software quality that will go to the state and federal governments over the next few years. It is currently the most important. I expect to also see proposals to:

- reduce legal liability of vendors and users of Y2K-defective software;
- license software testers, developers, and consultants;
- increase liability for defects in consumer or mass-market software (various proposed lemon laws);
- limit competition, via changes to the Copyright Act to (for example) ban or restrict reverse engineering of software products;
- increase competition, via changes to the antitrust laws (if Microsoft prevails in its antitrust case);
- increase / decrease / regulate / deregulate privacy protection on the Net;
- establish standards for reliability of internet services;
- establish standards for electronic signatures and other technical aspects of electronic commerce.

There will probably be plenty of other proposals.

Second, we are credible sources of information on these types of issues. We are industry insiders. We aren't embittered whistleblowers—we want the industry to succeed. We also have special insight—we know how products fail, we understand the difficulties of making perfect products and we also know how our defects affect customers.

Our input is valuable because most of the people who will have to evaluate these proposed laws are lawyers, and most of them are unsophisticated about software Many of the lawyers working on committees writing legislation to govern electronic commerce didn't even have e-mail accounts when they started work.

Many of the lawyers who will vote on these proposed laws still don't have e-mail, let alone more sophisticated e-commerce experience.

Even the ones with some experience as software users get a huge proportion of their education about software development and marketing from other lawyers who represent software publishers. This bias is pervasive. Legislative drafting committees dealing with software are visited or advised by many paid lobbyists fo software publishers and by very few, usually unpaid, consumer advocates (almost none of whom have software-related backgrounds). Additionally, courses and industry seminars on software law are typically taught by lawyers who represent software publishers / consultants. And speakers at conferences on software law are typically lawyers who represent publishers. There are hundreds of lawyers working for software sellers, but I can count the number of lawyers who publicly advocate for software quality on one hand.

If legal drafting bodies and legislatures are going to deal sensibly with the proposed laws to govern software quality, they need input from people like us.

Third, we can provide input—we are *welcome* to provide input—as individuals and as professional societies. People are hungry for our input. Non-lawyers can have a significant impact on laws by addressing technological issues and explaining the consequences of technology-related decisions. Software developers and testers haven't been all that well received in the UCITA/Article 2B drafting committee meetings, but they have had big effects elsewhere. For example, Bob Johnson is responsible for many significant improvements to the Uniform Electronic Transactions Act. And, even in the UCITA process, our comments have been effective in slowing the process down and convincing decision-makers to consider UCITA more carefully. ALI would probably have approved 2B/UCITA if it wasn't for our many comments.

In my experience, regulatory agencies, such as the Federal Trade Commission, are even more interested in our input than legislative drafting groups.

With particular reference to UCITA, we can do the following:

- Write letters to the head of NCCUSL, Gene Lebrun, <GLebrun@LYNNJACKSON.COM>. (Please send me a copy so that I can distribute them to the rest of NCCUSL at the annual meeting in July, 1999. It i unlikely that a letter to Gene will go further, and he strongly supports UCITA.)
- Write letters to the NCCUSL members in your state (contact me, kaner@kaner.com, for their names, etc. or check my website, www.badsoftware.com).
- Write letters to your state legislators and state governor. (This is a state law issue; members of Congress don't count.)

Write letters describing defects that were badly handled and examples of deceptiv or dishonest or unfair conduct by software publishers to Adam Cohn <ACOHN@FTC.GOV> of the Federal Trade Commission. Adam will of course be interested in fully detailed (names, dates, etc.) letters. You can also send him a letter that deliberately disguises the people/companies/products, that is sent to him only to educate him about the types of practices in the industry. Tell him

in these letters, that this is what you are doing and (if you want) tell him tha suggested that he would find these educational-purposes-only letters useful. These are valuable because the FTC is in an awkward position regarding UCITA. Federal agencies rarely comment on state law, but the authors of UCITA are making claims about the status and content of consumer protection law in the USA, and the FTC has significant expertise in this area. The FTC wrote one long letter commenting on Article 2B (see www.ftc.gov/be/v980032.htm) but has to decide whether to write another a which issues are important to address.

- Write letters and op-ed articles for your local newspaper. I can help you a bit with this.
- Encourage your professional societies (such as ASQ) to take a stand and to write some letters of their own.

==========

SOME ORGANIZATIONS THAT OPPOSE UCITA

- fifty intellectual property law professors (www.2BGuide.com/docs/1198ml.html)
- American Association of Law Libraries (www.arl.org/info/letters/libltr.html and www.arl.org/info/letters/Wright_ALI_letter.html)
- American Library Association (www.arl.org/info/letters/libltr.html and www.arl.org/info/letters/Wright_ALI_letter.html)
- American Society of Media Photographers (www.nwu.org/pic/uccasmp.htm)
- Association for Computing Machinery (www.acm.org/usacm/copyright/usacm-ucc2b-1098.html)
- Association of Research Libraries (www.arl.org/info/letters/libltr.html and www.arl.org/info/letters/Wright_ALI_letter.html)
- Consumer Federation of America (www.cptech.org/ucc/sign-on.html)
- Consumer Project on Technology (Ralph Nader) (www.cptech.org/ucc/sign-on.html)
- Consumers Union (www.2BGuide.com/docs/cu1098.html)
- Independent Computer Consultants Association (unpublished)
- Institute for Electrical & Electronics Engineers (IEEE) submitted specific criticisms of 2B but not final opposition. See www.ieee.org/usab/FORUM/POLICY/98feb23.html and www.ieee.org/usab/FORUM/POLICY/98oct09.html.
- Magazine Publishers of America (www.2BGuide.com/docs/v9-98.pdf)

- Motion Picture Association of America (www.2BGuide.com/docs/v9-98.pdf and www.2BGuide.com/docs/mpaa1198.html)
- National Association of Broadcasters (www.2BGuide.com/docs/v9-98.pdf)
- National Cable Television Association (www.2BGuide.com/docs/v9-98.pdf)
- National Consumer League (www.cptech.org/ucc/sign-on.html)
- National Music Publishers Association (unpublished)
- National Writers Union (www.nwu.org/pic/ucc1009a.htm)
- Newspaper Association of America (www.2BGuide.com/docs/v9-98.pdf)
- Recording Industry Association of America (www.2BGuide.com/docs/v9-98.pdf and www.2BGuide.com/docs/riaa1098.html)
- Sacramento Area Quality Association (unpublished)
- Society for Information Management (www.2BGuide.com/docs/simltr1098.html)
- software-test-discuss (this is the Net's largest e-mail discussion forum on software quality control)
- Special Libraries Association (www.arl.org/info/letters/libltr.html and www.arl.org/info/letters/Wright_ALI_letter.html)
- United States Public Interest Research Group (www.cptech.org/ucc/sign-on.html).

Most of these letters are brief. After consultation with some other consumer advocates, I submitted a detailed letter with a section-by-section call for consumer-side revisions (www.badsoftware.com/kanerncc.htm). The Society for Information Management's letter details the concerns of large software customers (www.2BGuide.com/docs/simltr1098.html).

==========

How UCITA Drives Down Failure Costs

The total quality cost for a product is the sum of:

- prevention costs (cost of preventing defects) plus
- appraisal costs (such as cost of testing)
- plus internal failure costs (such as cost of fixing defects)

plus external failure costs (costs caused by the defect after the product is release to customers).

The external failure costs in this model are the costs of the seller or manufacture not the costs of the customer. This model ignores the customer's costs (Kaner, 1996).

Normally, the best way to reduce external failure costs is to improve the product, especially by preventing defects or by finding them early in development. However, a company *can* reduce its external failure costs by handling them (e.g. customer complaints or lawsuits) more efficiently.

UCITA provides another approach—reduce external failure costs directly.

I classify external failure costs into three categories:

- Customer support costs
- Legal costs.
- Lost sales (especially sales lost to competitors).

Note that the publisher doesn't reduce its customer's losses by reducing these costs. In many cases, the publisher will save money by *increasing* its customer's losses under UCITA.

Customer Support Costs

Here are some of the ways that UCITA lets publishers reduce their technical support costs (without improving the product). Citations are to the July, 1999 draft of UCITA:

- The publisher gets to charge customers for support, even for known bugs. For example, if you buy a program for $50, the publisher might charge you $3 per minute for a support call. Suppose that you run into a (known) defect, call the publisher, talk for 30 minutes ($90), realize that you're not getting anywhere, and demand a refund. The publisher says OK, you send back the product (at your expense), the publisher sends you $50 and keeps your $90. It would hav been much cheaper to throw the defective product away. (Section 803(a)(1) 803(c).)
- When the buyer rejects a defective product because of obvious defects, the publisher can demand "a full and final statement in a record of all defects on which the refusing party proposes to rely." (Section 702(c)(2).) If there's a b that you don't find and report in response to this, you can't complain about it when it bites you later. (But not even the publisher's testing staff can find all the defects, so why should we expect a customer to be able to create a full a final statement of them?)
- A publisher's contract to support its software will not require it to fix all defects. (Section 612(a)(1)(B).)
- In a contract dispute, the publisher can sometimes use “self-help” to shut dow the operation of the program without court approval. (Section 816.)

The publisher will have the same or (probably) greater power to restrict custome right to maintain the publisher's software or to contract for 3rd party support for the software. . (This is achieved via contractual use restrictions on modification o 3rd party use of the product, see Section 102(a)(20) and 701(a).)

Legal Costs

Here are some of the ways that UCITA lets the publisher reduce its risk of legal liability for defective products, without making the product less defective.

- All of the terms of the contract except the price can be hidden from the customer until after the sale. By "hidden", I mean that the customer has no w to obtain the terms until after paying for the product. (Section 112, 210-213
- The implied warranty of merchantability (which provides that products will be reasonably fit for ordinary use) will be trivially easy to disclaim (to refuse to provide), and the disclaimer can be hidden from the customer until after the sale. (Sections 112, 210-212)
- By defining software transactions as licenses (which are intangibles) instead o sales of copies of the software (Section 102(a)(42)), UCITA takes these transactions out of the scope of the Magnuson-Moss Warranty Improvement Act and of state consumer protection statutes that are based on sales of goo go away. Consumers thereby lose warranty rights.
- It is much harder under UCITA than under current law (UCC Article 2) to prove that a warranty was created by a demonstration of a product. (Section 402(a and 402(b)(2)). A publisher can more easily get away with making misleading product demonstrations at trade shows.
- Business customers lose their right to reject a product for obvious defects. (Section 704(b) restricts the centuries old "perfect tender rule" to mass mark contracts, repealing it for the rest.)
- Except for mass-market software in the first day or so of use, you cannot can the contract (and return the software) unless there is a "material breach" (a very serious defect or group of defects). (Section 601, 704(a), 802(a).) The definition of material breach (Section 701) is more seller-friendly than the current definition, as laid out in the *Restatement of Contracts*. And finally, the publisher can specify that you cannot cancel the contract even if there is a material breach. (Section 803(a)(1).)
- Even if it is proved that the product is defective and the seller has materially breached the contract, the seller is liable for almost no remedies (payments to the customer). For example, the publisher doesn't have to reimburse callers fo incidental expenses (such as the cost of phone calls to the publisher or of returning the product) or consequential losses (such as the cost of restoring lost data) caused by the product's defect. (Section 803(d).) UCITA eliminates principle of the "minimum adequate remedy" developed in UCC Article 2 (the current law of sales, which governs contracts for packaged software today -- See Reporter's Note 6 to Section 803: "This Act does not give a court the right to invalidate a remedy limitation because the court believes that the imitation does not afford a "minimum adequate remedy" for the aggrieved party.)".
- The publisher can easily set up a waiver of liability (you "agree" to not sue the publisher for defects that you have complained about) by including the waiver in the click-wrapped "license" that comes with a bug-fix upgrade that the publisher sends you. (Section 702(a).)

- The contract can specify what state or country's law will govern this transaction and what court (in what country, state, city) you have to go to in order to bring a suit against the publisher. Forget about bringing a case in sma claims court against a publisher who sells you a defective consumer product. Unless the software is very expensive, or the suit is brought as a class action, you probably won't be able to afford to bring such a lawsuit because of the added travel and legal research expenses. Additionally, the publisher will probably have written into the contract the state whose laws and courts are most favorable to it. (Sections 109, 110, and many debates and resolutions during the 2B drafting meetings.)

Lost Sales (Especially Sales Lost to Competitors)

- UCITA will let companies prohibit publication of criticisms of their product. For example, with VirusScan, you get the restrictions, “The customer shall not disclose the results of any benchmark test to any third party without McAfee's prior written approval” and “The customer will not publish reviews of the product without prior consent from McAfee." These are restrictions on disclosure. Section 102(b)(20) defines a “contractual use restriction” as "an enforceable restriction created by contract, which restriction concerns the use or disclosure of, or access to licensed information or informational rights, including a limitation on scope or manner of use." Therefore, on their face, these terms are enforceable. Section 105 provides some limitations on these clauses. A clause can be thrown out if federal law specifies that it can be thrown out or if the customer can jump through a remarkable set of litigation hoops to prove that the clause violates a fundamental public policy or if (Section 111) a judge rules that the clause is unconscionable. These decisions are made on a case by case basis, so it will probably take many years, many trials, and many appeals before we have a clear understanding of which restrictions are enforceable and which are not. In the meantime, publishers of defective products will be able to intimidate people who can't afford to be sue by quoting their nondisclosure terms in their contracts and threatening to sue if the person publishes a critical article. (Such threats *have* been made.)
- UCITA will let publishers ban reverse engineering. This is just another contractual use restriction (102(a)(20)). See the discussion of these in the pc above. There are many legitimate reasons for doing reverse engineering, such as figuring out how to make one product compatible with another, figuring out how to work around the defects in a product, and figuring out how to fix the product's defects. (See Kaner, 1998, for discussion and a list of other example
- Finally, by making it easy for publishers to hide the terms of their contracts until after the sale is made, UCITA makes it hard for customers to compare several types of terms. For example, when you buy a program, do you know whether that publisher's warranty is better than its competitors'? What does the publisher charge for support, compared to its competitors? What are your rights to arrange for 3^{d} party maintenance of the product? Can you write critical reviews of it? These terms might all affect your buying decision, but nc if you can't find them until after you've paid for the product.

In Closing

I've focused this paper on UCITA's impact on software quality, but UCITA has many other serious problems involving electronic commerce and intellectual property rights. If you want details, or if you can offer some help, please write me (kaner@kaner.com).

References

Braucher, J. (1999) "Why UCITA, Like UCC Article 2B, is Premature and Unsound" forthcoming in the *UCC Bulletin*, July 1999. Available at http://www.2BGuide.com/docs/0499jb.html.

Kaner, C. (1996), "Quality cost analysis: Benefits and risks," *Software QA*, Volume 3, #1, p. 23, www.kaner.com/qualcost.htm.

Kaner, C.(1998), "Article 2B and reverse engineering," *UCC Bulletin*, November, p. 1 ww.badsoftware.com/reversea.htm. See also "The problem of reverse engineering *Software QA,* www.badsoftware.com/reveng.htm.

Notes

Law of Software Quality

Section 10.

Misrepresentation

Misrepresentation

- ☒ False representation by the seller
- ☒ of a material (important) fact
- ☒ that the plaintiff justifiably relies on
- ☒ and as a result, the plaintiff is damaged.

Misrepresentation can be:

- ☒ Fraudulent
- ☒ Negligent
- ☒ Innocent

A misrepresentation is fraudulent if the maker

- ☒ knows or believes that the matter is not as he represents it to be, or
- ☒ does not have the confidence in the accuracy of his representation that he states or implies, or
- ☒ knows that he does not have the basis for his representation that he states or implies

Misrepresentation

Post-sale misrepresentation

Post-sale fraud is actionable if it causes a person to forego or refrain from asserting an existing right or to change position in some other way.

Mention this to support staff who think they're supposed to lie if necessary to keep a customer from returning a product.

Ritchie Enterprises v. Honeywell Bull, Inc. (1990)

Misrepresentation

Negligent Misrepresentation

- ☒ The *duty* is to exercise the care or competence of a reasonable person who is communicating information.
- ☒ Not all misrepresenters will be held liable. Many states require a special relationship between the victim and misrepresenter, such as a position of trust.
- ☒ States vary in the degree to which they allow a negligent misrespresentation suit, in the face of an integration clause and no misrepresentation in the body of the contract.

Misrepresentation

Negligent Misrepresentation

Burroughs Corporation v. Hall Affiliates, 423 So. 2d 1348 (Supreme Court of Alabama, 1982). Hall imports artificial flowers, and bought a Burroughs B80-40 computer in 1977 to handle its accounting and inventory functions. The system didn't work and Hall sued claiming that Burrough's salespeoples' representations about the system were fraudulent. The court listed 4 representations:

1. the machine would do inventory and accounting simultaneously
2. the machine was capable of multiprogramming
3. the machine was capable of operating a terminal display unit in a data communications environment
4. the machine and all of its component parts were new

The jury found that Burroughs had committed fraud and awarded $500,000. Burroughs challenged the finding to the Supreme Court of Alabama. The Court upheld the verdict, explaining that in Alabama, all that was required for a finding of fraud was

(a) a false representation by the seller

(b) the representation must concern a material existing fact

(c) the plaintiff must rely upon the false representation

(d) the plaintiff must be damaged as a proximate result.

Intent is not part of the basic definition of fraud in Alabama. What we call negligent misrepresentation in California is fraud in Alabama. In Alabama, the proof of intent is still relevant, in order to make a case for punitive damages.

Publisher Liability

Winter v. G.P. Putnam's Sons, 938 F.2d 1033, (9th Circuit) 1991. Winter became seriously ill from picking and eating mushrooms after relying on ***The Encyclopedia of Mushrooms***, published by Putnam. Putnam did not verify the material in the book and did not intentionally include the error.

- ☒ Putnam was not liable for errors in the book.
- ☒ Noted that Jeppesen cases have consistently held publisher liable for information error when information published to be used as a tool.
- ☒ However, software publisher might be liable for program that does not work as intended.

ALM v. Van Nostrand Reinhold 480 NE.2d 1263, 1985. ***The Making of Tools.*** Plaintiff used it, and a tool shattered, causing injury. VNR not liable, ***but author of the book might be.***

Cem Kaner. Ph.D.. J.D. **P.O. Box 1200**
Law Office of Cem Kaner **Santa Clara. CA 95052**
kaner@kaner.com **408-244-7000**

Liability for Defective Content

With all the storage space on CD-ROMs, many products come with loads of added content, such as books, articles, maps, audio clips, video clips, and clip art. Much of this material is bought cheaply and then copied onto the disk without any testing beyond verification that it installs correctly.

What happens if information provided on these disks is incorrect? Can a software publisher be sued for informational errors? From a legal liability point of view, should you *insist* on testing all of the factual material in the product?

The short answer is, *No*. The longer answer is, *Well, not usually but there are some important exceptions*. I've heard people overgeneralize the general rule. It can be a mistake to conclude that publishers never have liability for informational errors. In this brief article, I want to familiarize you with the general rule and then point out some of the risks.

The General Rule: No Liability

In the case of *Alm v. Van Nostrand Reinhold*[1], Alm bought a how-to book, *The Making of Tools*. He was injured when a tool shattered while he was (allegedly) following the book's instructions for making that tool. He sued the publisher and the author. The Court refused to allow the case to proceed against the publisher. It cited a long series of decisions that newspapers and magazines could publish material written by a third party without fear of being sued for that writer's mistakes. The Court concluded,

> *"Plaintiff's theory, if adopted, would place upon publishers the duty of scrutinizing and even testing all procedures contained in any of their publications."*[2]

Test all the procedures? Oh No! No! Not that!

There *are* serious problems with requiring a publisher to check all of the facts in books or articles submitted to it. For example[3], there is the First Amendment problem. The First Amendment to the Constitution of the United States prohibits laws "abridging the freedom of speech, or of the press." How long would it delay the news if the newspaper had to independently check every fact in every article? How much would it add to the cost of newspapers and books?

[1] *North Eastern Reporter, Second Series*, Volume 480, p. 1263 (Appellate Court of Illinois, 1985)

[2] *Alm*, p. 1267.

[3] The First Amendment problem isn't the only problem. The *Alm v. Van Nostrand Reinhold* Court was more concerned about the potentially huge but unpredictable size of the class of potential plaintiffs. This is an important problem, but too specialized for this paper. If you're intrigued by it, I recommend Jay Feinman's book, *Economic Negligence*, Little Brown, 1995, Section 1.3.2, "The threat of indeterminate liability."

How much would it interfere with the publication of unpopular ideas? How much harder would it be for a new writer or a controversial writer to get published? Could a political or religious group harass and eventually bankrupt a publisher with mistake-alleging lawsuits? In the face of such considerations, American courts have consistently held that a publisher is not liable for errors – not even for errors that cause personal injuries or deaths – that were made by an independent author.

The case of *Winter v. G.P. Putnam's Sons*[1] provides a second instructive example. G.P. Putnam's Sons bought 10,000 copies of a book that was originally written and published in Britain. Putnam distributed the book in the United States after putting its label on the book, plus some material on the flyleaf that said that the book contained "strongly practical, wide-ranging reference sections" and that the book's reader would be able to identify and classify particular mushrooms "at a glance." Winter and a friend picked and ate mushrooms, relying on the Encyclopedia to distinguish safe from unsafe. Unfortunately, one of the mushrooms they ate was the *amanita phalloides* (a.k.a. *Death Cap*). They became very ill, required liver transplants, and incurred about $400,000 in medical expenses. They sued.

The Court stated that ideas and expression are "governed by copyright laws; the laws of libel, to the extent consistent with the First Amendment; and the laws of misrepresentation, negligent misrepresentation, negligence, and mistake."[2]

After considering various arguments for imposing liability, the Court stated that "Guided by the First Amendment and the values embodied therein . . . we conclude that the defendants have no duty to investigate the accuracy of the contents of the books it publishes. A publisher may of course assume such a burden, but there there is nothing inherent in the role of publisher . . . to suggest that such a duty should be imposed."[3]

Exceptions

Here are several important exceptions to keep in mind:

your company might also be the author

your content might be about your own product

the content might be defamatory

your company might have created a warranty of accuracy or safety

your company might have a special relationship with the reader or user of the content

your content might be intended to be the sole source of information available to the user, and errors expose the user to great risk.

Each of these issues deserves an article much longer than this one (and you may see some of them in this magazine in the future).

[1] *Federal Reporter, Second Series*, Volume 938, p. 1033 (United States Court of Appeals, 9th Circuit, 1991). Some of the details of my description of this case are from Brett Lee Myers' article, "Read at your own risk: Publisher liability for defective how-to books", *Arkansas Law Review*, Volume 45, p. 699, 1992.

[2] *Winter*, p. 1036.

[3] *Winter*, p. 1036-37.

Author Liability

In *Alm v. Van Nostrand Reinhold*, the judge threw out the case against Van Nostrand, but let it go forward against the book's author. The same thing happened in *Jones v. J.B. Lippincott Co*.[1] In *Birmingham v. Fodor's Travel Publications, Inc*.,[2] the Court said "No jurisdiction has held a publisher liable in negligence for personal injury suffered in reliance upon information contained in the publication unless the publisher authored or guaranteed the information."[3]

"The balance might well come out differently, however, if the publisher contributed some of the content of the book. The burden of determining whether the content was accurate"[4] would be more reasonable to assign to a publisher that writes what it publishes.

This doesn't mean that your company will be liable for *every* mistake (or even most of them). The rules that govern author liability are complex.

But the point to keep in mind is that your *legal* duty to test content for accuracy is greater if you create it than if you buy it from someone else.

Of course, your customer satisfaction risks are probably the same in both cases – there is more to conducting honest and honorable business than meeting the minimum requirements of the law.

Your Own Product

Statements of fact about your own product can be taken as warranties that the product works as you've described it. See my article, "Liability for Defective Documentation" in Volume 2, #3 of *Software QA*.

Defamation

People and businesses can sue over false statements that damage their reputation. Laws of defamation (libel, slander, etc.) trade off free speech rights against peoples' needs to prevent the spreading of damaging lies about them. These are complex laws, especially complex if you sell the product in several countries.

If you think that some of the content your company is publishing might be defamatory, talk to your company's lawyer.

Warranty of Safety or Accuracy

If you promise that your material is safe or accurate, and if it is to your commercial advantage if people rely on your material, and if you invite people to rely on it, then it would be wise to be right. The classic case is *Hanberry v. Hearst Corp*.[5] The plaintiff bought shoes that bore the Good Housekeeping Seal of Approval and was injured. She sued Hearst (Good Housekeeping) alleging that the shoes were dangerously slippery and that Hearst had guaranteed the shoes when it published its approval of them.

[1] *Federal Supplement*, Volume 694, p. 1216 (United States District Court, District of Maryland, 1988). The plaintiff suffered an injury from treating herself after consulting the *Textbook for Medical and Surgical Nursing*.

[2] *Pacific Reporter, Second Series*, Volume 833, p. 70 (Supreme Court of Hawaii, 1992).

[3] *Jones*, p. 75.

[4] *Lewin v. McCreight, Federal Supplement*, Volume 655, p. 282, 284 (United States District Court, Eastern District of Michigan, 1987).

[5] *California Appellate Reports, Second Series*, Volume 276, p. 680 (California Court of Appeal, 1969).

Courts are cautious to avoid extending warranties beyond those intended by the publisher. For example, in *Yanase v. Automobile Club of Southern California*, the Court read a AAA Tourbook as rating the cleanliness and comfort of a motel, but not the safety of the motel's neighborhood.[1] And in *First Equity v. Standard & Poor's Corp*.[2], the Court carefully noted Standard & Poor's caution that it was compiling information from third parties and could not guarantee accuracy.

How does this apply to software? Software publishers sometimes make exaggerated claims, and these might result in a duty to test all of the content on the disk. Here's a purely hypothetical example, suggested to me by a colleague. Several programs on the market display city maps and offer directions from one place to another. Suppose that one such program was marketed with the claim written on the box that the maps are "up to date" and with the promise that the program will provide safe routes through strange cities. And suppose that, in fact, the publisher was using 17-year old maps because they were available very inexpensively. Could someone be injured or mugged as a result of inaccuracies in the map? If so, maybe they could sue.

Your company isn't obliged to make any promises about what it sells, but it is bound by any promises that it makes.

Special Relationship

You might have a special duty to provide accurate information to someone because of a contract or a professional relationship. For example, an accountant is liable to her client for errors in reports (such as audits) that she submits. The rules governing professional liability for misinformation are beyond the scope of this article. Feinman[3] is a useful starting place.

Sole Source of Special Information

If your company publishes navigation charts that will be used in the air by pilots, it will be held liable for errors that cause crashes.[4] If your company publishes warning labels, it will be held liable for accidents that occur because a label that should have said "FLAMMABLE" didn't.[5]

If you know that people will count on your product to provide them with accurate information, and that your product will be the primary (and maybe only) source of information that is available to these people, and that errors could result in deaths or injury, then you have to make sure that the information is accurate.

[1] *California Appellate Reports, Third Series*, Volume 212, p. 468 (California Court of Appeal, 1989).

[2] *Federal Supplement*, Volume 670, p. 115 (United States District Court, Southern District of New York, 1987).

[3] *Economic Negligence*, Little Brown, 1995.

[4] For example, see *Halstead v. United States, Federal Supplement*, Volume 535, p. 783 (United States District Court, District of Connecticut, 1982).

[5] *Firkin v. U.S. Polychemical Corporation, Federal Supplement*, Volume 835, p. 1048 (United States District Court, Northern District of Illinois, 1993).

Notes

Law of Software Quality

Section 11.

Consumer Protection

Consumer Protection

Deceptive Trade Practices

Unfair Competition

Additional Warranty Rules

Additional Leasing Rules

Additional Negative Option Rules

False Claims Act

Uniform Deceptive Trade Practices Act

> A person engages in deceptive trade practices when s/he represents that goods or services have sponsorship, approval, characteristics, ingredients, uses, benefits, or quantities that they do not have.

Deceptive Trade Practices

California Civil Code 1770:

The following unfair methods of competition and unfair or deceptive acts or practices undertaken by any person in a transaction intended to result or which results in the sale or lease of goods or services to any consumer are unlawful:

(a) Passing off goods or services as those of another.

(b) Misrepresenting the source, sponsorship, approval, or certification of goods or services.

(c) Misrepresenting the affiliation, connection, or association with, or certification by, another.

(d) Using deceptive representations or designations of geographic origin in connection with goods or services.

(e) Representing that goods or services have sponsorship, approval, characteristics, ingredients, uses, benefits, or quantities which they do not have or that a person has a sponsorship, approval, status, affiliation, or connection which he or she does not have.

(f) Representing that goods are original or new if they have deterioriated unreasonably or are altered, reconditioned, reclaimed, used or secondhand.

(g) Representing that goods or services are of a particular standard, quality, or grade, or that goods are of a particular style or model, if they are of another.

(h) Disparaging the goods, services, or business of another by false or misleading representation of fact.

Consumer Protection

False Advertising

California Business & Professions Code 17500

Applies to all advertising, not just consumer goods.

Bars making statements of fact that are known or should be known to be false -- negligent misrepresentations are unlawful. (see People v. Superior Court, Orange County, 96 Cal.3d 181, 1979, Cert. denied 446 U.S. 935.)

California Business & Professions Code 17200

As used in this chapter, unfair competition shall mean and include any unlawful, unfair or fraudulent business act or practice and unfair, deceptive, untrue or misleading advertising and any act prohibited by . . . Section 17500

17204. Actions for any relief pursuant to this chapter shall be prosecuted . . . by any person acting for the interests of itself, its members, or the general public.

Attorney fees -- see Civil Code 1021.5.

Consumer Protection

Unfair Competition

Princeton Graphics v. NEC 732 F. Supp. 1258 (S.D.N.Y. 1990). NEC held liable for advertising that the original Multisynch monitor was VGA compatible.

Compaq Computer Corp. v. Packard Bell Electronics, Civil Action 95-222, U.S. District Court, D. Delaware, 1995. Compaq alleges that Packard Bell recycles returned, used components into “new” computers.

Consumer Protection

FTC Actions

Go to www.ftc.gov for copies of complaints and settlement agreements involving such companies as Apple, Dell, Gateway 2000, Iomega, and others. Issues include:

- ☒Failure to provide tech support for life after promising to do so
- ☒Failure to disclose key terms of the contract
- ☒Deceptive practices of various kinds

Cem Kaner, Ph.D., J.D.
Law Office of Cem Kaner **kaner@kaner.com**
http://www.badsoftware.com **408-244-7000 (v)**
P.O. Box 1200, Santa Clara, CA 95052 **408-244-2181 (f)**

LIABILITY FOR PRODUCT INCOMPATIBILITY

In Press, Software QA Magazine, September 1998.

What does it mean to claim that one product is compatible with another?

There are at least three meanings:

(1) ***The product works with (is interoperable with) another product.*** For example, a word-processing program might be advertised as MS-DOS compatible if it runs under MS-DOS. It might be advertised as LOTUS 1-2-3 compatible if it can read and write files in Lotus's native file format.

(2) ***The product works just like (emulates) another product.*** For example, many printers were advertised as LaserJet II-compatible. Many mice were allegedly Microsoft-compatible. One program might be said to emulate another if the same commands yield the same results in both cases (think of the spreadsheet war between Borland and Lotus).

(3) ***The product lives up to a well-specified standard that is independent of any particular product.*** For example, a printer might be PostScript compatible. A modem might be V.34 compatible. A graphics program might be JPEG-compatible (meaning that it reads and writes JPEG files).

I'll treat these three types of claims separately later, but I also just use the word "compatible" in several places in this paper. In those places, I am including all three types in my discussion.

Suppose that Brand X Inc. ships a product, Product X, that it advertises as compatible with RealWare Corp.'s product, RealThing. And Suppose that Product X is not compatible with RealThing. Lots of people stand to lose time and money because of this, for example:

> A company that competes with Brand X loses market share because people buy Product X instead of the competitor's product because they think that Product X is RealThing-compatible. This is the situation that we see claimed in *Princeton Graphics v. NEC Home Electronics* (1990, discussed below).

RealWare loses market share because people buy Brand X thinking that Product X is RealThing-compatible. Additionally, if Product X has problems and customers think that those problems are typical of RealThing-compatible products, then RealWare loses customer goodwill for problems that don't exist in its products. This is the situation that we see claimed in the case of *Creative Labs, Inc. v. Cyrix Corp.* (1997a, 1997b, affirmed 1998) and in the case of *Compaq Computer, Corp. v. Procom Technology* (1995).

The reseller (retailer or VAR) who sold the product faces complaints, troubleshooting and other technical support costs, refunds, and potentially lawsuits from dissatisfied customers. This is the situation that we see claimed in the case of *Step-Saver Data Systems, Inc. v. Wyse Technology and The Software Link, Inc.*, (1991, discussed below).

The end customer buys Product X and now has to waste time troubleshooting the difficulties between Product X and the rest of her (software and/or hardware) system. This is often expensive. For example, it takes in-house help desks 3 to 18 times as long to resolve a multi-vendor problem as a problem that can be pinned to a specific piece of software or hardware (Oxton, 1997; Schreiber, 1997). Private individuals, who don't have help desks with experienced staff, often give up and throw products away instead of trying to make them work with their other products.

These cases, and some statutes on deceptive trade practices, suggest some standards that we should keep in mind when developing or marketing products that are allegedly "compatible." You'll note that three of the four cases here are hardware cases (or, more precisely, embedded software cases). I think that's just the luck of the draw. My bet is that the same standards will be applied to all other types of software--we just haven't had the right lawsuits yet.

Princeton Graphics v. NEC

When IBM came out with the PS/2 computer, it established a new graphics standard, VGA. Princeton Graphics and NEC both manufactured EGA-compatible monitors. EGA involves a resolution of 640 pixels across by 350 scan lines vertically, refreshing (repainting) the screen 60 times per second (60 Hz), while VGA involves a resolution of 640 by 480, refreshing the screen at 70 Hz. Another standard in use at that time, PGA, involves 640 by 480 at a refresh rate of 60 Hz.

In 1987, NEC issued a press release claiming that afer "extensive testing" the MultiSync monitor had been determined to be "fully compatible" with VGA (*Princeton Graphics v. NEC*, p. 1260). But if you switched between EGA and VGA resolution, the screen would roll, apparently because the NEC MultiSync misinterpreted the VGA signals as PGA (wrong vertical refresh rate). You could stop the roll when you switched into VGA mode by adjusting the vertical hold knob on the monitor, but when you switched back to EGA, you had to adjust the vertical hold back. (Some programs were designed on the assumption that the display was EGA while later programs supported or required VGA. Back in 1987-89, depending on your system's software mix, you might switch back and forth from VGA to EGA several times in a day.)

The court ruled that (p. 1262):

> "When a clearly defined standard, like IBM's VGA standard, exists and is widely accepted within the industry, a 'compatible' product must meet that standard or at least perform in a manner equivalent to the standard's requirements."

However (p. 1262), the court said that when there is no well-known standard in the industry, "'compatible' essentially is understood to mean 'works with' or the ability of one device 'to function with' another."

The court ruled that NEC had falsely advertised its compatibility and held NEC liable to Princeton Graphics.

Compaq v. Procom

Compaq made the Proliant line of network servers and a line of hot-pluggable hard drives for use with Proliant servers. Compaq sold the Compaq Insight Manager (CIM) program, which monitors (among other things) the performance of the server's hard drives. The CIM could generate 'prefailure warnings' to the user, indicating that a hard drive will soon fail. If the performance of the drive was worse than certain settings determined by Compaq, Compaq would replace the drive (if it was under warranty) even though it had not yet failed. The threshold values (performance criterion values) differed for different hard drives. Compaq stored the threshold values on a special partition on the disk.

Procom made Proliant-compatible drives and bought them from Seagate, the same supplier as Compaq.It also stored threshold values on a special partition that the CIM program could find. However Procom's threshold values differed from Compaq's, for the same drives.

Compaq sued Procom alleging (among other things) that Procom was falsely advertising its drives as fully compatible with Compaq's drives. The claim was false because Procom's prefailure warnings would be given to customers at different times than Compaq's for comparable drives. Thus, their performance was not identical.

The court relied on the determination in *Princeton Graphics v. NEC*, that when there is no well-known standard in the industry, a looser definition ("works with") is in order. Here, there was no standard for prefailure values. Compaq had blended its assessment of technical risk and business risk in its determination of parameters for prefailure warnings. (Remember, when a warranty-holding customer gets a prefailure warning, he is entitled to a new drive. This is expensive. It creates a tradeoff between the value of warning customers early and the cost of replacing drives prematurely.) Procom came up with its own prefailure values, to be used in the same way as Compaq's. The numbers were different from Compaq's but in the absence of a standard, the court didn't know whether Procom or Compaq was using the better set of numbers or whether the differences would matter in practice. Therefore the court ruled that the drives were not incompatible.

Creative Labs v. Cyrix

Creative Labs makes SoundBlaster sound cards. Cyrix makes microprocessors, including the Media GX, which is capable of producing sounds without the assistance of a sound card. The audio component of the Media GX is called XpressAUDIO. Cyrix advertised XpressAudio as "compatible with Sound Blaster."

Creative Labs tested a Compaq Presario 2100 computer (which uses the Media GX) and found that it was sound-incompatible with 16 of 200 games tested (8%). Creative Labs then filed a false advertising suit against Cyrix. As did Princeton Graphics and Compaq in their cases (above), Creative Labs filed suit under the Lanham Act (Section 43(a); United States Code, Title 15, Section 1125(a)). The court said (1997a, p. 1874)

> "The elements of a Lanham Act false advertising claim are: (1) a false statement of fact by the defendant in a commercial advertisement about its own or another's product; (2) the statement actually deceived or has the tendency to deceive a substantial segment of its audience; (3) the deception is material, in that it is likely to influence the purchasing decision; (4) the defendant caused its false statement to enter interstate commerce; and (5) the plaintiff has been or is likely to be injured as a result of the false statement, either by direct diversion of sales from itself to defendant or by a lessening of the goodwill associated with its products."

Cyrix retested these games and said that its product failed with only 10 of them. It worked with the other 6 when the computer was "properly configured" (p. 1875). Of these 200 games, then, the failure rate was 5%. Cyrix claimed that it had other data indicating a failure rate of only 2%.

The court responded (p. 1875) that "Even if the failure rate of games placed on computers with XpressAUDIO is closer to 2% than 8%, the evidence indicates that some games that function with Sound Blaster do not function with XpressAUDIO." Therefore, they are (evidently) not compatible.

The three Creative Labs decisions that have issued so far have involved motions for injunctions—court orders that Cyrix should (among other things) quit advertising compatibility until and unless Creative Labs loses its case at trial. Therefore, the court's reasoning and conclusions do not (they can't, at this point in the litigation) constitute a legally binding determination that the Cyrix processor is incompatible with the Sound Blaster. However, they are clear signals on how the court will rule when it is given the chance. This court's summary of the law came directly from *Princeton Graphics v. NEC*: when a standard has been defined (here, Sound Blaster compatible), the allegedly compatible product must meet that standard (*see* p. 1874-1875).

Step-Saver Data Systems v. Wyse and The Software Link

Step-Saver was a value added reseller who put together systems for dentists and doctors and lawyers. From November, 1986 to March, 1987, it sold 142 systems that treated the IBM AT computer as a multi-user machine. The software that enabled multi-user capability was The Software Link's (TSL's) Multilink Advanced, an operating system that was advertised as MS-DOS compatible. Additionally, "Step-Saver requested information from TSL concerning this new version of the program, and allegedly was assured by sales representatives that the new version was compatible with ninety percent of the programs available "off-the-shelf" for computers using MS-DOS." (*Step-Saver* p. 95)

Note that The Software Link didn't claim 100% compatibility, and therefore the strict*Princeton Graphics v. NEC* standard could not apply. Instead, this court defined compatibility ("practical compatibility" as opposed to "complete compatibility") as (p. 106):

> "Two products are compatible, within the standards of the computer industry, if they work together almost every time in almost every possible situation."

The *Step-Saver* case is famous for a different ruling. Step-Saver bought IBM AT computers, loaded them with the Multilink Advanced operating system, some Wyse terminals, and a bunch of other software. When customers returned the system as defective (allegedly because of compatibility problems when used as a multi-user machine), Step-Saver didn't just have a returned operating system. It also had to deal with a now-used set of software and hardware, and it wanted reimbursement from The Software Link (TSL) for all of its losses, not just refunds for the returned copies of the operating system. To achieve this result, it sued TSL for breach of warranty and for misrepresentation. TSL responded that a license had come with each copy of the product, that the license disclaimed all warranties, express and implied and had limited remedies to a refund of the cost of the product, that this license was a binding contract between Step-Saver and TSL and therefore, TSL said, Step-Saver wasn't entitled to repayment for its losses. This license was essentially the same as the "license agreements" that you find inside the box when you buy software or that appear on the screen when you install software (having already paid for it).

The court threw out the post-sale license, saying that you just can't disclaim warranties after the sale is complete. Step-Saver was entitled to the benefit of whatever warranties were made and whatever losses it suffered.

The attempt to add Article 2B to the Uniform Commercial Code is partially a reaction to this decision and the cases that have followed it. For more information on that aspect of Step-Saver, and more on the history of post-sale licenses and post-sale disclaimers, see Kaner & Pels (1998) Chapter 7 and the Appendix. Under current law, no court has approved a post-sale (hidden in the box until after the sale) disclaimer of warranties, not for software and not for any other products. These disclaimers look pretty and legalistic and impressive on the paper, but they are not likely to hold up in court.

End User Class Action Suits

Customers can't sue under the Lanham Act but they *can* sue for breach of contract, for fraud, and for deceptive trade practices (DTP). I'm not aware of DTP lawsuits that have focused specifically on compatibility, but there is an increasing amount of DTP litigation against software companies. I am aware of small groups of lawyers that are training each other in the details of how to handle DTP cases against software and computer hardware companies. Consumers can bring class action suits under DTP statutes, and they can typically get their attorneys' fees from the defending publisher *if* they win their case.

The details of the statutes vary from state to state, but the banned acts are similar. Here are some examples from the California Civil Code Section 1770(a). The following are unlawful:

1770(a)(1)	Passing off goods or services as those of another.
1770(a)(2)	Misrepresenting the source, sponsorship, approval, or certification of goods or services.
1770(a)(5)	Representing that goods or services have sponsorship, approval, characteristics, ingredients, uses, benefits, or quantities which they do not have or that a person has a sponsorship, approval, status, affiliation, or connection which he or she does not have.
1770(a)(7)	Representing that goods or services are of a particular standard, quality, or grade, or that goods are of a particular style or model, if they are of another.
1770(a)(8)	Disparaging the goods, services, or business of another by false or misleading representation of fact.

Consumers (or any large group of customers) can also sue as a class for breach of contract or fraud (intentional deception).

In Sum

The court cases that we've seen so far don't cover every possible situation. There haven't been enough of them yet, and the issues are somewhat complex. But I think that the courts have laid out a continuum that we should pay attention to.

> If we are emulating or interoperating with a product that is poorly defined, the court will probably cut us some slack when we advertise "compatible."
>
> If we are emulating or interoperating with a product that is well defined, or if we are advertising that we meet a well defined industry standard, then we will probably be held to it. When the product is well defined, "compatible" means compatible. Not "sort of" compatible. Not "take two workarounds and call me in the morning" compatible. Just plain old genuine compatibility.

Failures of compatibility have plagued software and hardware buyers. They're tired of it. The recent successes in the Creative Labs v. Cyrix case and the recent successes in several deceptive practices cases will spur interest among more lawyers. We are likely to see more successful legal actions against the publishers, manufacturers and retailers of not-quite-compatible products.

By carefully testing your company's claims of compatibility against the products that you're supposed to be compatible with, you can help your company avoid slipping into an expensive, unanticipated compatibility-related lawsuit.

References

Creative Labs, Inc. v. Cyrix Corp. (1997a) United States Patents Quarterly, 2nd Series, *volume 42,* p. 1872 (United States District Court, Northern District of California).

Creative Labs, Inc. v. Cyrix Corp. (1997b) United States Patents Quarterly, 2nd Series, *volume 43,* p. 1778 (United States District Court, Northern District of California).

Creative Labs, Inc. v. Cyrix Corp. (1998) 1998 U.S. Appellate Library LEXIS #6470 (United States Court of Appeals for the Ninth Circuit).

Kaner, C. & D. Pels (1998) *Bad Software: What To Do When Software Fails.* John Wiley & Sons.

Oxton (1997, March) "Multivendor support challenges", *Software Services Conference East, Nashville,* TN.

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

Schreiber, R. (1997, March), "How the Internet Changes (Almost) Everything," presented at the Association for Support Professionals' *Internet Support Forum,* San Jose, CA.

Step-Saver Data Systems, Inc. v. Wyse Technology and The Software Link, Inc., (1991) Federal Reporter, 2nd Series, *volume 939,* p. 91 (United States Court of Appeals, 3rd Circuit).

Notes

Law of Software Quality

Section 12.

Negligence

Negligence

The Language of Negligence

Elements of a negligence case:

☒ Duty

» *products must not create an unreasonable risk of injury or property damage*

» *professionals must provide services at a level that would be provided by a reasonable member of the profession in this community*

☒ Breach

☒ Causation

☒ Damages

Examples:

☒ **Duty**: *Bryant v. Glastetter*, 95 CDOS 1426 (Cal. 4th App. Dist., Super. Ct. No. 215438, 2/23/1995). Court ruled that drunk driver had no duty to tow truck operator who was injured while towing drunk's car – there was a negligent breach of a duty, but not of a duty to protect THIS plaintiff from THIS harm. No liability.

☒ **Causation**: Reckless drivers can't be sued until they cause an accident.

☒ **Damages**: For normal bad driving, a driver pays only compensatory damages. For reckless driving, a driver might also have to pay punitive damages

The Negligence Formula

"Negligence" involves a tradeoff -- conduct must be , not just harmful.

How do we decide that conduct is unreasonable?

Judge Learned Hand presented the tradeoff as a formula, in the famous case of the United States v. Carroll Towing Co.:

- ☒Let be the burden (expense) of preventing a potential accident
- ☒Let be the severity of the loss if the accident occurs
- ☒Let be the probability of the accident

Failure to attempt to prevent a potential accident is unreasonable if x

Negligence

The Incremental Cost of Finding a Bug

To calculate the "Burden" of additional care, we must calculate what it would have cost to change the process to prevent or find this bug:

- ☒ the cost of strengthening the testing so that line 7000's bug is found *in the normal course of testing*.
- ☒ the cost of changing the design and programming practices in a way that would have prevented this bug and others like it in the first place.

Negligence

Factors for Evaluating Negligent Conduct

Factors in the Defendant's conduct that a court would be likely to consider:

1. Actual knowledge of the problem
2. Expertise of the staff
3. Standard methodology
4. Industry standards
5. Safety committee & hazard analysis
6. Design for error handling
7. Handling of customer complaints
8. Bug tracking methodology
9. Quality control plan
10. Testing intensity
11. Testing coverage

For additional discussion,
see Chapter 14 of Testing Computer Software

Negligence

Actual Knowledge Can Result in Increased Liability

General Motors v. Johnston, 592 So.2d 1054, 1992 (Supreme Court of Alabama). Johnson purchased a new Chevrolet 2500 pickup in 1988, drove it for less than 200 miles, over two days. With his 7 year old grandson in the truck, he pulled up to a stop sign, started the truck again, and it stalled. A larger truck collided with his truck, injuring Johnson and killing the grandson.

GM had received reports of stalling problems in vehicles like this one, and a dealer service bulletin advised dealers that "rolling, hunting or surging idles" could be fixed by replacing the PROM. The software on the modied PROM was changed from the original software. This PROM controled the fuel injector. GM chose to replace the PROM for complaining customers, but not to announce a recall and make the change widely available.

The jury awarded $15 million and the Alabama Supreme Court upheld it at $7.5 million.

Complaints put you on notice of problems that you may not have known about before you released the product.

Negligence

Industry Standards

- In a negligence suit, failure to follow a standard is relevant if but only if plaintiff can show that this failure was a causal factor in the injury.
- We must distinguish between product and process standards.
- To what extent *should* industry standards determine a standard of care?
- Are standards that are suitable for Mil Spec also suitable for shrinkwrap product development?

Negligence

Safety Committee, Hazard Analysis

- ☒The wrong answer is, “Safety committee? What safety committee”
- ☒Use consultants if you have to; make sure that the committee has the expertise to identify hazards.
- ☒Try to identify all potential hazards arising out of the product. Are there special (higher risk) uses or users?
- ☒What are the forseeable misuses of the product?

Negligence

Error Handling

- ☒Doctrine of forseeable misuse
- ☒90% of industrial accidents are caused by “user errors”
- ☒Therac-25
- ☒A320 airplane

Negligence

Customer Complaints

How do you handle complaints?

- ☒ This is your last chance to improve customer goodwill.
  - » Response time - 90% abandonment rate?
  - » What do you do to follow up with customers?
  - » What's your process for getting information into and out of Product Development?
- ☒ Evidence of good faith.
  - » Remember that punitives are available for despicable conduct.
  - » Your process is an indicator of your willingness to be negligent.

Jurors more likely to be people who call for support than people who answer the calls.

- ☒ Baldwin v. Alabama Insurance Brokers, 599 So.2d 1196 1992 (Ct. of Civil Appeals, AL)

Negligence

Bug Handling Process

The goal and objectives of a bug tracking system

- ☒Is the process reasonable?
- ☒Is the process thorough?
- ☒Is the process accountable?
- ☒Are customer interests carefully considered?
- ☒Are potential consequences carefully considered?

Negligence

Quality Control Plan

This is a broad area. There will be narrow factual questions, and a broader impression that the system is in control or it is not.

- ☒Design/code reviews, inspections
- ☒IEEE 829 test plans
- ☒ISO 9000-3
- ☒Auditing of the QC plan

Speaking from the perspective of an expert witless, it is seriously problematic for the defendant when the defendant writes up a detailed development process and then doesn't follow it.

Don't write what you aren't going to do.

Negligence

Test Documentation

There are many different notions of what a good set of test documentation would include. Before spending a substantial amount of time and resources, it's worth asking what documentation should be developed (and why?)

What questions should we ask in order to determine our requirements?

The list that follows comes from the Los Altos Workshop on Software Testing (I thank Chris Agruss, James Bach, Karla Fisher, David Gelperin, Kenneth Groder, Elisabeth Hendrickson, Doug Hoffman, III, Bob Johnson, Cem Kaner, Brian Lawrence, Brian Marick, Thanga Meenakshi, Noel Nyman, Jeffery E. Payne, Bret Pettichord, Johanna Rothman, Jane Stepak, Melora Svoboda, Jeremy White, and Rodney Wilson for their insights.)

I don't think it is necessary to ask all of these questions every time. Pick the few that will be most informative for you and your company.

Test Plan Requirements: Contrasting Objectives

- ☒ Is test documentation a product or tool?
- ☒ Is software quality driven by legal issues or by market forces?
- ☒ How quickly is the design changing?
- ☒ How quickly does the specification change to reflect design change?
- ☒ How much traceability do you need? What docs are you tracing back to and who controls them?
- ☒ Is testing approach oriented toward proving conformance to specs or nonconformance with customer expectations?
- ☒ Does your testing style rely more on already-defined tests or on exploration?
- ☒ Should test docs focus on what to test (objectives) or on how to test for it (procedures)?
- ☒ Should control of the project by the test docs come early, late, or never?
- ☒ To what extent should test docs support tracking and reporting of project status and testing progress?

Test Plan Requirements: Contrasting Objectives

- ☒ Who are the primary readers of these test documents and how important are they?
- ☒ How well should docs support delegation of work to new testers?
- ☒ What are your assumptions about the skills and knowledge of new testers?
- ☒ Is test doc set a process model, a product model, or a defect finder?
- ☒ A test suite should provide prevention, detection, and prediction. Which is the most important for this project?
- ☒ How maintainable are the test docs (and their test cases)? And, how well do they ensure that test changes will follow code changes?
- ☒ Will the test docs help us identify (and revise / restructure in face of) a permanent shift in the risk profile of the program?
- ☒ Are (should) docs (be) automatically created as a byproduct of the test automation code?

Software Negligence: Testing Intensity

Are these good test methods for detecting errors? How hard are you looking?

- ☒boundaries
- ☒stress tests
- ☒depth of exploration of key risk areas

Characteristics of an excellent test case

Why is this important? Because sometimes the testing looks jury-impressingly sloppy.

Negligence

The Problem of Coverage

Coverage measures of the amount of testing done of a certain type. Since testing is done to find bugs, coverage is a measure of your effort to detect a certain class of potential errors:

- *100% line coverage* means that you tested for every bug that can be revealed by simple execution of a line of code.
- *100% branch coverage* means you will find every error that can be revealed by testing each branch.
- *100% coverage* means that you tested for every possible error. This is obviously impossible.

So what kind(s) and level(s) of coverage should we consider appropriate? There is no magic answer.

Negligence

The Problem of Coverage

Several people seem to believe that complete statement and branch coverage means complete testing. (Or, at least, sufficient testing.)

Part of the rationale comes from IEEE Std. 982.1-1988, § 4.17, “Minimal Unit Test Case Determination”

IEEE Unit Testing Standard is

100% Statement Coverage

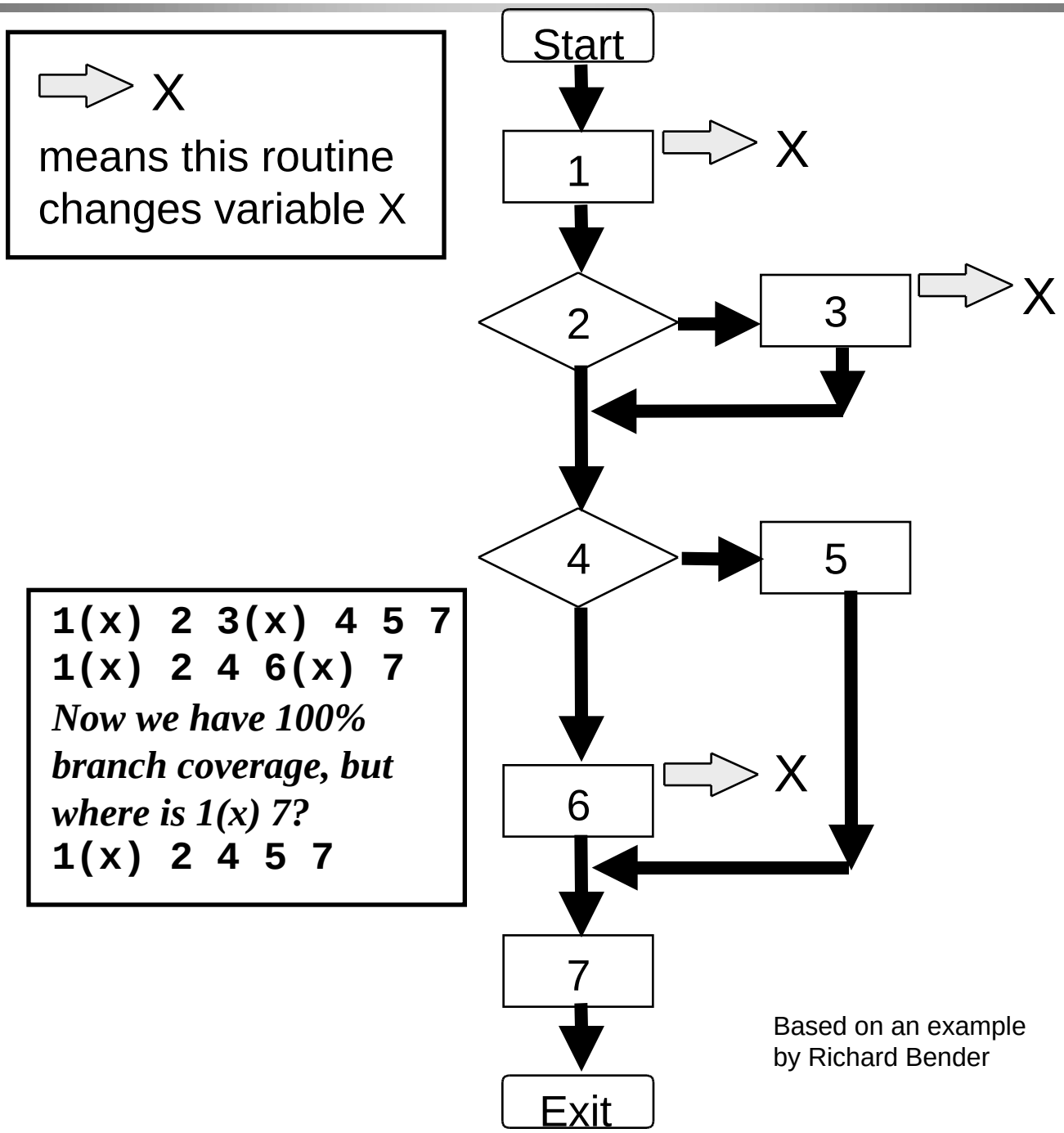
and 100% Branch Execution

Most companies don’t achieve this (though they might achieve 100% of the code they actually write.)

Negligence

The Problem of Coverage

One Example: Data Flows



Negligence

The Problem of Coverage

There is no magic talisman for amount and/or type of testing. You have to do what is reasonable under the circumstances.

- ☒ 100% line and branch coverage is not sufficient testing
- ☒ 100% line and branch coverage may not be necessary for reasonable testing – you might better spend your time on some other goal.
- ☒ Some senior people in the field will use it against you in expert testimony against you if you don't achieve 100% line/branch coverage.
- ☒ Obviously (I hope that it is obviously), you have to scale the amount of testing to the amount and nature of risk. You have to do what is reasonable under the circumstances.

LAW OFFICE OF CEM KANER **Cem Kaner, Ph.D., J.D.** **kaner@kaner.com**
P.O. Box 1200 **408-244-7000 (Voice)**
Santa Clara, CA 95052 **408-244-2181 (FAX)**

The Impossibility of Complete Testing

Law of Software Quality **Column, Software QA magazine**

I'm writing this from a hotel room in Sacramento, California. The National Conference of Commissioners on Uniform State Laws (NCCUSL) is reviewing Article 2B of the Uniform Commercial Code (UCC)— a 340-page statute that will govern all contracts for the development, sale, licensing, and support of software. Next week, I'll meet with yet more lawyers about the same bill, at the annual meeting of the American Bar Association.

NCCUSL is the authoring organization for Uniform Laws (laws that are written to be identical in all states). Its expenses are mainly paid by the American state governments. It is effective at getting its bills passed. For example, over the past 12 months, 200 of NCCUSL's bills have been introduced in state legislatures and 95 have been passed so far.

This is a full meeting of NCCUSL. The 300 Commissioners (senior attorneys who serve as unpaid volunteers) are reviewing nine different draft Uniform Laws, involving such areas as child custody, guardianships, estate planning, sales of merchandise, and software. The discussion of 2B has been largely favorable. The expectation is that it will be approved for introduction in state legislatures in September, 1998.

For the next year, work on 2B will be done by a 16-member NCCUSL committee whose meetings are open to the public. A typical Article 2B meeting draws 40-80 people, mainly lawyers. The 2B Drafting Committee meetings have had more attendance and press coverage than any of NCCUSL's other Drafting Committees' meetings. I've been attending these meetings for the last 18 months and have been joined by other software quality advocates, including James Bach, Doug Hoffman, Brian Lawrence, Melora Svoboda, Clark Savage Turner, and Watts Humphrey (see Humphrey, 1997). The next meeting will be September 26-28 at the Sofitel Hotel in Minneapolis. The meeting after that will run November 21-23 in Memphis.

As I've said in more detail in previous columns, I think this is a dangerous law that will reduce software publishers' accountability to their customers. I've put together a website on this, www.badsoftware.com.

It's tempting to argue in these meetings that software products should be free of defects. Unfortunately, I don't think they can be, and therefore I don't think that the law should insist that they be. This complicates my position in these meetings, and so I spend a lot of time explaining simple things about software development to lawyers. One of those simple things is a lesson that we all know (or that I hope we all know) as testers***—it is impossible to fully test a program***.

Lawyers aren't the only people who don't understand this. People still sign contracts that promise delivery of a fully tested program. And I still hear executives and project managers tell testers that it is their responsibility to find all the bugs. But how can you find them all, if you can't fully test the program? You can't. As the testing expert in your company, you'll have to explain this to people who might not be technically sophisticated.

This paper explores three themes:

1. I think that I've figured out how to explain the impossibility of complete testing to managers and lawyers, with examples that they can understand. These are my notes.
2. A peculiar breed of snake-oil sellers reassure listeners that you achieve complete testing by using their coverage monitors. Wrong. Complete line and branch coverage is not complete testing. It will miss significant classes of bugs.
3. If we can't do complete testing, what should we do? It seems to me that at the technical level and at the legal level, we should be thinking about "good enough testing," done with as part of a strategy for achieving "good enough software."

Tutorial on Impossibility

You probably know this material, but you might find these notes useful for explaining the problem to others.

Complete testing is impossible for several reasons:

> We can't test all the inputs to the program.
>
> We can't test all the combinations of inputs to the program.
>
> We can't test all the paths through the program.
>
> We can't test for all of the other potential failures, such as those caused by user interface design errors or incomplete requirements analyses.

Let's look at these in turn.

Can't test all inputs

The number of inputs you can feed to a program is typically infinite. We use rules of thumb (heuristics) to select a small number of test cases that we will use to represent the full space of possible tests. That small sample might or might not reveal all of the input-related bugs.

Valid Inputs: Consider a function that will accept any integer value between 1 and 1000. It is *possible* to run all 1000 values but it will take a long time. To save time, we normally test at boundaries, testing perhaps at 0, 1, 1000, and 1001. Boundary analysis provides us with good rules of thumb about where we can shortcut testing, but the rules aren't perfect.

One problem is that many programs contain local optimizations. Even though every integer between 1 and 1000 might be valid, the program might actually process these numbers differently, in different sub-ranges.

For example, consider the Chi-Square function, a function popular among statisticians. One of the inputs to Chi-Square is a parameter called "degrees of freedom" or DF, for short. There is a second input, which

I'll call X. Let's write a program to calculate Chi-Square(X,DF). The shape of the Chi-Square function changes, depending on the value of DF.

> When DF = 1, the shape of Chi-Square (X,1) is exponential. Chi-Square(X,1) is large for tiny values of X, but drops rapidly toward 0 for larger values of X. A computer program can calculate the exact value of Chi-Square (X,1).
>
> When DF is larger than 1, its shape changes and we lose the ability to exactly calculate the value of Chi-Square. Abramowitz & Stegun's famous *Handbook of Mathematical Functions* (1964) provides a good formula for closely approximating values of Chi-Square.
>
> For DF of 100 or larger, Abramowitz and Stegun recommend using yet a third function that is faster and easier to compute.

If you don't know that the program uses three different algorithms to estimate the value of Chi-Square, you won't bother testing at intermediate values like 100.

There are lots of optimizations in code—special treatments for special cases or for ranges of special cases. Without detailed knowledge of the internals of the program, you won't know what to test.

Invalid Inputs: If the program will accept any integer between 1 and 1000, then to test it completely, you'd have to test every number below 1, above 1000, all the rational numbers, and all the non-numeric inputs.

Suppose you only tested –1, 0, and 1001 as your invalid numerical inputs. Some programs will fail if you enter 999999999999999—a number that has too many characters, rather than being too numerically large.

Edited Inputs: If the program accepts numbers from 1 to 1000, what about the following sequence of keystrokes: 1 <backspace> 1 <backspace> 1 <backspace> 1000 <Enter>? If you entered single digits and backspaced over them hundreds of times, could you overflow an input buffer in the program? There have been programs with such a bug. How many variations on editing would you have to test before you could be *absolutely certain* that you've caught all the editing-related bugs?

Timing Considerations: The program accepts any number between 1 and 9999999. You type 123, then pause for a minute, then type 4567. What result? If you were typing into a telephone, the system would have timed you out during your pause and would have discarded 4567. Timing-related issues are at the heart of the difference between traditional systems and client/server systems. How much testing do you have to do to check for timeouts and their effects? If timeouts do occur in your system, what happens with 1-pause-2-pause-3-pause where no one pause is quite long enough for a timeout, but the total pause time is? What if you type 1-pause-2 with a pause that is just barely enough to trigger a timeout? Can you confuse the system by typing the 2 just barely too late?

Can't test all combinations of inputs

A few years ago, printers and video cards both came out in higher resolution models. On a few programs, you could run the printer at high resolution (600 dpi) with no problem and you could run video at high resolution, but if you tried a print preview, crash. In retrospect, it's easy to understand that your video driver might interact with your printer driver, but the bug was a big surprise to us at the time. You probably know of interactions between mouse and video drivers. How many versions of mouse drivers should we test in combination with how many video drivers with how many printer drivers?

I Suppose a program lets you add two numbers. The program's design allows the first number to be 1 and 100 and the second number to be between 1 and 25. The total number of pairs you can test (not counting all of the pairs that use invalid inputs) is 100 x 25 (2500).

In general, if you have V variables, and N1 is the number of possible values of variable 1, N2 is the number of possible values of variable 2, and NV is the number of values of variable V, then the number of possible combinations of inputs is N1 x N2 x N3 x . . . x NV. (The number is smaller and the formulas are more complicated if the values available for one variable depend on the value chosen for a different variable.) It doesn't take many variables before the number of possible combinations is huge.

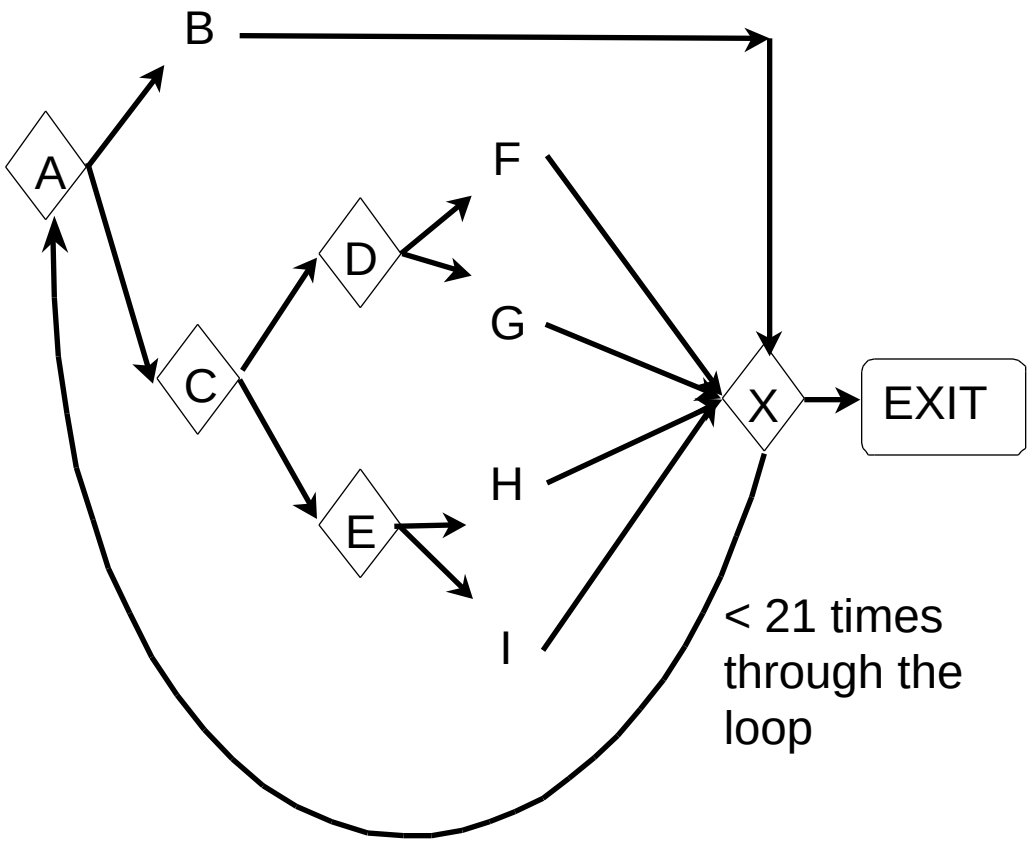
We can use heuristics (domain testing) to select a few "interesting" combinations of variables that will probably reveal most of the combination bugs, but we can't be sure that we'll catch all of them.

Can't test all the paths

A path through a program starts at the point that you enter the program, and includes all the steps you run through until you exit the program. There is a virtually infinite series of paths through any program that is even slightly complex.

I'll illustrate this point with an example from Glen Myers' (1979) excellent book. Some students of mine have had trouble with Myers' diagram, so Figure 1 present the example in a flowchart-like format.

Figure 1. The Paths Example, flowcharted



The program starts at point A and ends at Exit. You get to Exit from X.

When you get to X, you can either exit or loop back to A. You can't loop back to A more than 19 times. The twentieth time that you reach X, you must exit.

There are five paths from A to X. You can go A to B to X (ABX). Or you can go ACDFX or ACDGX or ACEHX or ACEIX. There are thus 5 ways to get from A to the exit, if you only pass through X once.

If you hit X the first time, and loop back from X to A, then there are five ways (the same five) to get from A back to X. There are thus 5 x 5 ways to get from A to X and then A to X again. There are 25 paths through the program that will take you through X twice before you get to the exit.

There are 5^3 ways to get to the exit after passing through X three times, and 5^{20} ways to get to the exit after passing through X twenty times.

In total, there are $5 + 5^2 + 5^3 + 5^4 + \ldots + 5^{20} = 10^{14}$ (100 trillion) paths through this program. If you could write, execute, and verify a test case every five minutes, you'd be done testing in a billion years.

Sometimes when I show this example to a non-programming executive, he doesn't appreciate it because he doesn't realize how simple a program this is. This program has just one loop. Most programs have many loops. This program has only five decision points. (For example, there's a decision point at A because you can go either from A to B or from A to C.) Most programs have many decisions. Students in their first semester of their first programming class will probably write more complex programs than this one.

Another path example

Some people dismiss the path testing problem by saying you can run all the tests you need with *sub-path testing*. A sub-path is just a series of steps that you take to get from one part of the program to another. In the above example, A to C is a sub-path. So are ACD, ACDF, and ACDFX. Once you've tested ACDFX, why should you test this sub-path again? Under this viewpoint, you'd probably test ABX, ACDFX, ACDGX, ACEHX and ACEIX once each plus one test that would run you through X the full 20 times. Think realistically, these people say. The program is not going to be so weirdly designed that it will pass a sequence like ACDFX one time and then fail it the next time. Sometimes they're wrong.

If you go through a sub-path with one set of data one time, and a different set of data the next, you can easily get different results. And the values of the data often depend on what you've done recently in some other part of the program.

Here's the example that I use to illustrate the point that simple sub-path testing can miss devastating bugs. As one of the system's programmers, I helped create this bug and then helped troubleshoot it after we started getting calls from beta testers in the field. I've simplified the description and I've probably misremembered some minor details, but the essence of the bug and the circumstances are captured here.

The application was a voice/data PBX (private telephone system) that used a custom station set (telephone, but with a built-in central processor, memory, and display). The station set had an LCD display. The top line showed information, the second line showed choices. For example, when you were talking to someone, the second line would show choices like "Transfer" and "Conference." Press the button below "Transfer" to transfer the call. If you had calls on hold, you could press a button to display the hold queue. For each call, you'd see a person's name or phone number, or a telephone line (we showed the best information we had). You could have up to ten calls on hold at the same time. The

display could show up to four calls at once, and you could scroll through the list. Press the button under the name to connect to the call.

Phone systems have conflicting design goals. For example, fast performance is very important. When you pick up a phone's handset, you expect to hear dial tone right away. But you also expect absolute reliability. Error checking and error recovery take time and can interfere with performance.

When a call was put on hold, we stored information about it on a stack. We could have up to ten calls on hold at once, so the stack had to be ten deep. We made it large enough to hold twenty calls, just in case we somehow forgot to take a call off of the stack when you connected to it or you or the caller hung up. We had no reason to expect stack problems, but the essence of fault tolerant programming is that you anticipate problems that the system *could* have and make sure that if those problems ever arise (perhaps when someone adds a new feature), they are going to be controlled.

We would have liked to check the integrity of every stack every time that a phone call changed state (such as going on or off of hold). Unfortunately, this took too much time. Instead, we defended against the possibility of an error by allowing stack space for 20 calls instead of 10. We also created a Stack Reset command. Any time your phone got to an idle state (no calls connected, on hold, or waiting), we knew that your stacks had to be empty. Therefore, we forced your stacks to *be* empty.

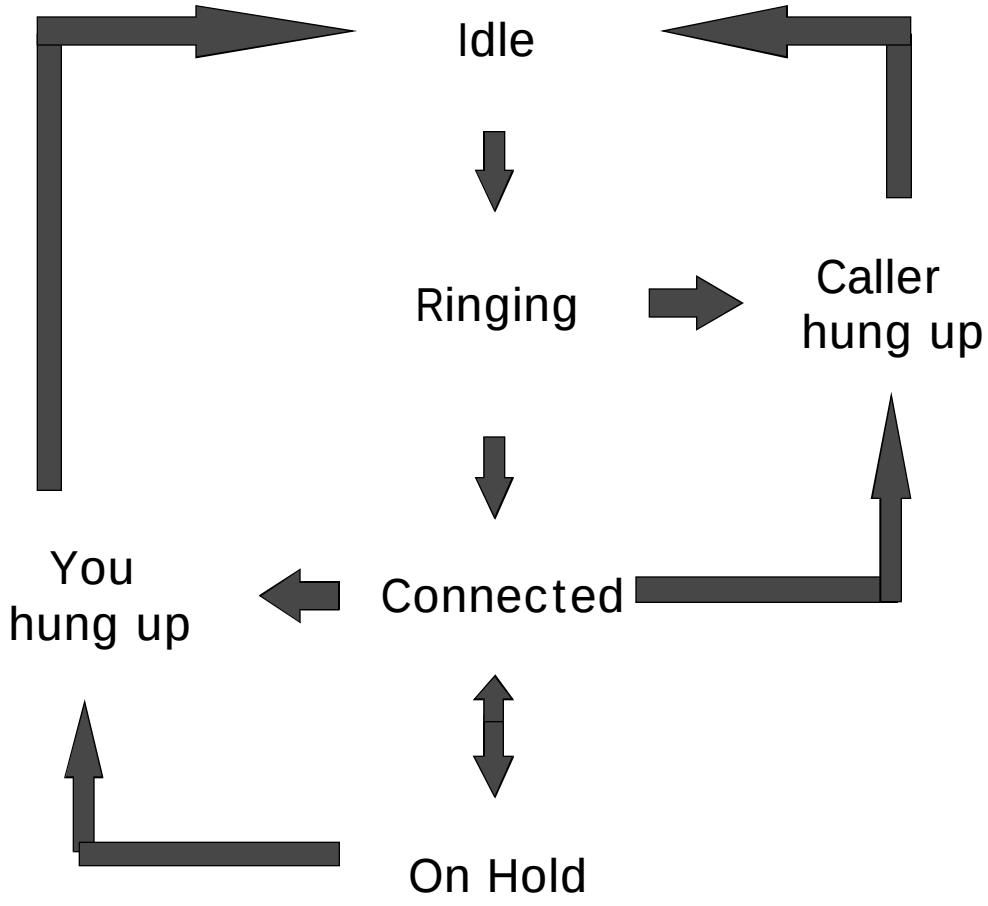


Figure 2. State Diagram of the Phone System

Take a look at the simplified state diagram, Figure 2.

Your station set starts in the Idle state. When someone calls you, the set moves into the "Ringing" state. Either you answer the phone ("Connected" state) or the caller gives up, hangs up, and you go back to Idle. While you're connected, you or the caller can hang up (back to Idle in either case) or you can put the caller on hold. When the caller is on hold, you can hang up on him. (This lets you get rid of a caller who is behaving inappropriately, without making it obvious that you are intentionally hanging up.)

Unfortunately, we missed a state transition. Sometimes, when you put a person on hold, she hangs up before you get back to her. See Figure 3. In retrospect, and in the way that I describe the state transitions, this is obvious. In retrospect, it is astonishing that we could have missed this case, given the ways that we drew our diagrams and discussed/reviewed our approaches. But we did. Mistakes happen, even when bright, careful people try to prevent them.

The effect of the mistake was quite subtle. When the caller hangs up while she waits on hold, her call is dropped from the hold queue. The call no longer appears on your display and you can still put up to ten calls on hold. Other system resources, such as time slices reserved for the call and phone lines used by the call, are freed up. It is impossible for the system to try to reconnect to the call; the only error is that a bit of space on the stack is wasted for a short time. Because the stack depth is 20 calls, and the design limit is 10 calls on hold, the system can tolerate an accumulation of 10 such errors before there is any performance effect. Furthermore, as soon as the phone returns to the idle state, the stacks are cleared, and the wasted stack space is recovered.

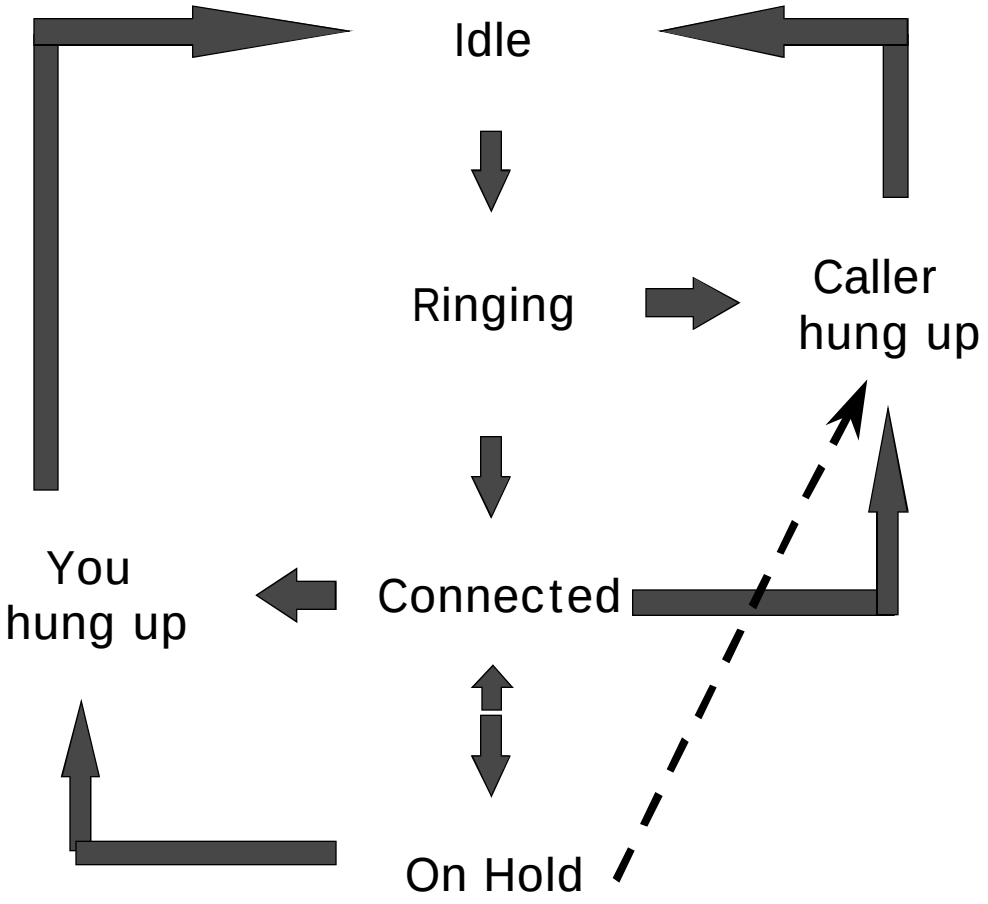


Figure 3. Corrected State Diagram of the Phone System.

If you did manage to run your phone so busy for so long that eleven of your callers hung up while on hold before you got the phone back to an idle state, and if you then tried to put a tenth call on hold, your phone would take itself out of service. It is bad for a telephone system to lose or drop calls, so calls on hold or waiting on the phone going out of service were automatically transferred to the hold and waiting queues of up to two other phones that you had designated as back-up systems. Once the calls were cleared off your phone, it rebooted itself, downloaded code and data from the main server (the PBX's main computer) and was back in service within two minutes.

Suppose that the phones that your calls were transferred to were as busy as yours. Your transferred calls would fill their hold queue, people would wait longer to be answered, people would hang up while they were waiting, and soon those phones would crash too. And as they crashed, they would send their calls back to your phone. This is a one-phone-at-a-time example of a "rotating outage."

We saw this problem at a beta site of stock brokers, the gymnasts of the telephone world. They loved our phone, except when it would mysteriously crash when a stockbroker was at her busiest. These crashes were expensive.

We were lucky that this was the worst consequence of the bug. Some communities' 9-1-1 (emergency response) administrators were interested in beta testing our system. We were reluctant to send buggy code to 9-1-1 so, fortunately, we never did set up such a beta site. Back in those days, some 9-1-1 systems practiced triage. If you called 9-1-1 and the dispatcher decided that your call wasn't urgent, then he would put you on hold and check the next waiting call. Once the genuine emergencies were handled, the non-emergency calls were taken off hold and dealt with. Under these circumstances, every phone's hold queue would often be full. Our bug would have been crashing 9-1-1 telephones, and that reduction in emergency response service would probably have resulted in unnecessary deaths. This was a serious bug.

Think about the paths through the program that you would have to test to find this bug in the laboratory. I'll define a "path" as something that starts when your phone is idle, goes through various steps, but ends up with the phone back at idle. A path might correspond to receiving an incoming call, answering it, receiving a second call, answering it (puts the first call on hold), switching between calls (putting each on hold), setting up a conference (all three talk together), receiving another call during the conference, putting the conference on hold, more switching back and forth, making an outgoing call with all of these people on hold, transferring the call, then reconnecting to the held calls and ending them, one at a time, until the phone was idle. That is one path.

As you design your set of tests, note that you have no expectation of finding stack bugs. Stack corruption issues had been cleared up about a year before. There were no known existing stack problems. Therefore you probably will not be taking special care to search for new stack failures. Furthermore, if you do run stack-related tests, you will discover that you can put 10 voice calls on hold, 10 data calls on hold, deal with 10 voice calls and 10 data calls waiting, and take calls on and off hold on any of those channels with no problem. If you hang calls up while they are holding, all visible indications are that that call is terminated correctly. And because the stack is fully reset as soon as the phone is allowed back to idle state (no calls connected, waiting. or on hold), you will not see a failure unless your test involves having 11 callers hang up while waiting on hold, and putting another 10 calls on hold together, before taking the phone back to idle. You can use a debugger to check the contents of the stack (we did glass box and black box testing), but if you run a simple test like putting the call on hold, letting the call disconnect itself, and then watching the phone reset its stack, the phone will appear to work correctly, unless you are very, very observant, because the system does a stack reset for your phone as soon as it hits idle, which

happens in this test only a few instructions after the caller hangs up. So the stack will be clear, as it should be, after the simple caller-hung-up-on-hold test.

If you were not specifically looking for a problem of this kind, how huge a set of path tests would you have to design before you could be reasonably confident that your set of tests would include one that would reveal this bug? I think the number of paths (from idle state to idle state) that you would have to explore before you stumbled across this bug would be huge.

Lots of other tests

A system can fail in the field because the software is defective, the hardware is malfunctioning, or the humans who work with the system make a mistake. If the system is designed in a way that makes it likely that humans will make important mistakes, the system is defective. Therefore, to find all defects in a system, you have to test for usability issues. And hardware/software compatibility issues. And requirements conformance issues, timing issues, etc. There are lots and lots of additional tests.

Dick Bender publishes a program called SoftTest that analyzes a well-written external specification and produces an optimal set of tests using a method called *cause-effect graphing* (see Myers, 1979 or Kit, 1995). This method helps you select a relatively small set of combinations of data and sub-paths that covers a wide range of potential problems. As part of the process, SoftTest estimates the number of unique test cases that could be run through the program. I was at his office when they analyzed a complex program. The computer plodded along for a while, then printed its estimate: 10^{100} tests. (SoftTest then suggested a set of about 3000 tests that covered the situations that cause-effect graphing would tell you were interesting.) 10^{100} tests is such a big number that you might not realize how big it is. So here's a comparison. I've been told that current estimates of the number of molecules in the universe is 10^{90}. No matter how many testers and how much automation you use, you aren't going to run 10^{100} test cases in a commercially reasonable time.

Coverage Monitors

Some testing experts call coverage monitors state-of-the-art tools and tell us that we aren't doing adequate testing unless we can demonstrate 100% coverage using a coverage monitor. I don't think the situation is that straightforward (Kaner, 1996b), but there's room for argument. But other people go farther than this, saying that if you achieve 100% coverage as measured by a coverage monitor, then you have achieved "complete" coverage and you have done complete testing. This is baloney.

A coverage monitor is a program that will measure how many tests you have run of a certain type, out of a population total of possible tests of that type. You achieve complete coverage when you finish running all of the tests of that type. For example, the monitor might check how many statements (or lines of code) you have executed in the program. A more sophisticated monitor recognizes that you can branch in many ways from one statement to others, and counts how many branches you have executed. You achieve complete statement-plus-branch coverage when you've tested every statement and every branch. Most discussions of coverage that I see are either of statement-plus-branch coverage or of some technique that adds a bit more sophistication to the same underlying approach.

Coverage-based testing can miss data flow-related bugs

Statement and branch coverage look at software in the same way that you would if you were preparing a detailed flowchart. In 1967, in my first programming class, I learned that the key to good program design

was to draw flowcharts. But the field has made progress since then. For example, it's often more important to look at the flow of data through the program, the ways that inputs are transformed into outputs, than to think about what instruction the processor will execute next.

More people know how to read flowcharts than data flow diagrams, so I'll present a data flow problem here in flowchart style. This is based on an example developed by Dick Bender in his course on Requirements-Based Testing. See Figure 4.

In Figure 4, we have a 7-statement program that uses one variable, X. This is a payroll program. It calculates your paycheck. X is the amount you are paid.

On line 1, the program initializes X. We'll give the program a bug, and say that line 1 initializes X to $1,000,000. (Of course, *you* might think of this as an *undocumented feature*.)

Line 2 calculates your base pay, setting a new value for X in line 3.

Line 4 determines whether the amount of your pay should be calculated using a different formula, perhaps because you are on vacation and so you get vacation pay instead of pay for hours worked. If so, X is reset in line 5.

At line 7, the program prints your check.

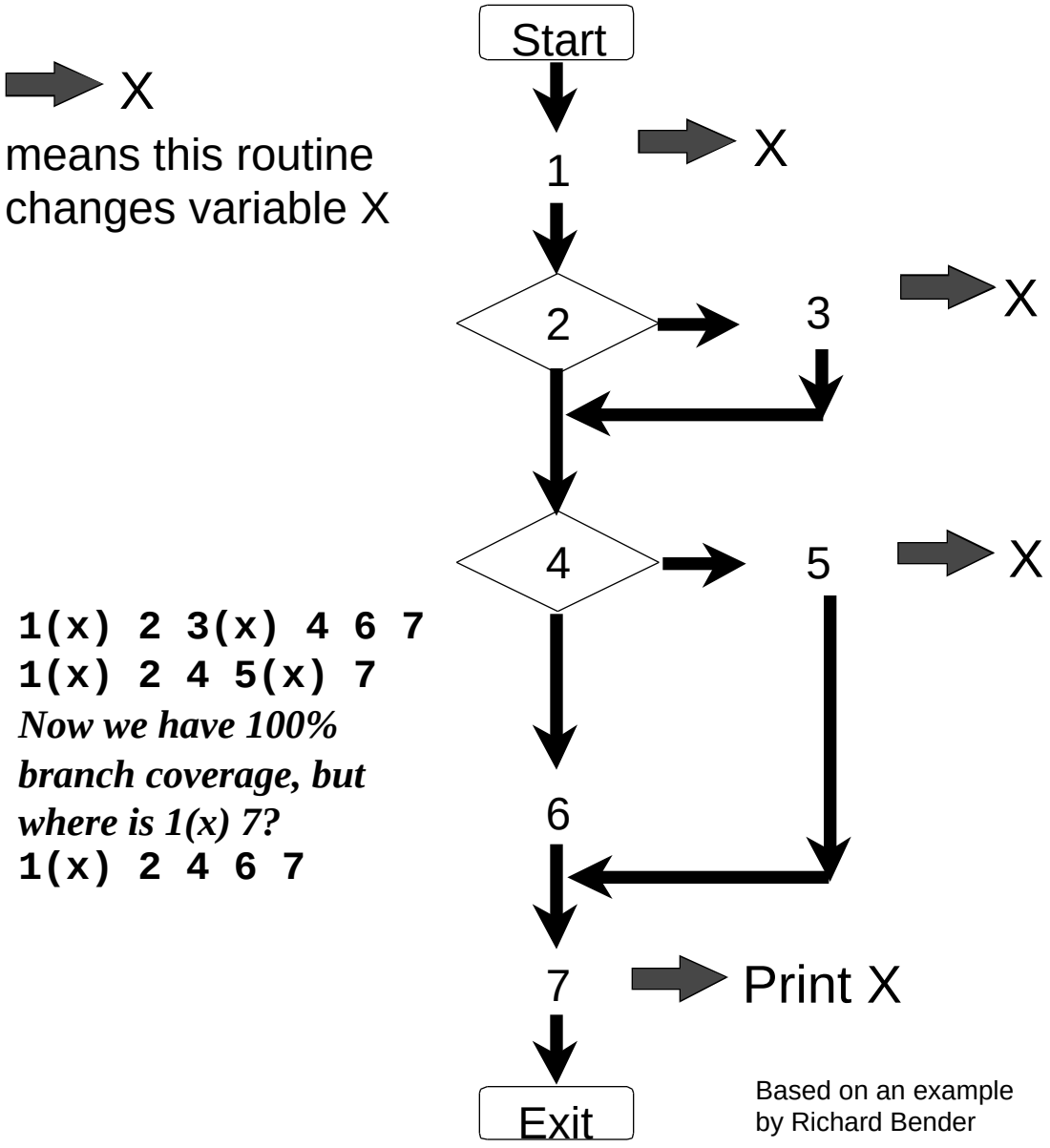


Figure 4. Complete Coverage Misses a Dataflow Bug.

There are three first-order data flows in this program. Think of a data flow as involving a path from the place or event that sets a variable's value to a place where that value is used. The flows are 1-to-7 (line 1 sets a value for X, and the value is used in line 7), 3-to-7, and 5-to-7.

Now let's do some coverage testing.

For our first test, we'll go through lines 1 (sets X), 2, 3 (sets X), 4, 6, and 7 (prints X). This covers all the lines except 5 and it covers the branches from 2 to 3 and from 4 to 6. To achieve complete line and branch coverage, we need a test that includes line 6 and the branches from 2 to 4 and from 4 to 5. Therefore our next test runs through lines 1 (sets X), 2, 4, 5 (sets X) and 7 (prints X).

We now have complete line and branch coverage. Have we tested the program completely? Absolutely not.

This "complete" pair of tests only covers two of the program's data flows, 3-to-7 and 5-to-7. If you stopped here, no one would ever know that you'll get paid $1,000,000 if the program ever runs the path through lines 1 (sets X), 2, 4, 6, 7 (prints X).

Interrupt-related bugs

Here's another example of the holes in flowchart-driven (line/branch) testing. While a computer is executing a main program, it might receive a signal, called an interrupt, telling it to switch focus to another routine called the interrupt handler. The computer will save certain working variables in a temporary storage spot, then handle the interrupt, then recover its working variables and return to the main routine as if nothing had happened. You might jump to the interrupt handler from (almost) any line in the main program. Coverage monitors don't count these implicit jumps from every line to the interrupt handler, for every possible type of interrupt. A program that deals with an interrupt can affect the main routine in many ways. For example, it might reset a variable that's in use or take a device out of service. Here's a specific example.

In the main program, we'll input values for A and B, then check whether B is zero, and if not, divide A by B. Suppose that right after we check B, but before we divide, there's an interrupt. The effect of the interrupt is to reset B to 0. Now we come back to the main routine. It knows that B is not zero because it just tested B. Therefore it divides A by B (which has just been reset to 0) and crashes.

This situation might seem far fetched, but it is not unusual in event-driven systems, in which many different events can happen at any time. Traditional programmers often write routines that input A, input B, check B for zero, do a bunch of stuff, then divide A by B. In an event-driven world, programmers learn the hard way that the more intervening processing between the time you check B's value and the time you use it, the greater the chance that B's value has changed while you weren't looking. Therefore, real-time programmers will often check for B=0 occur *just before* B is actually used as a divisor, *each time* B is used as a divisor. This appears to make no sense in the mainline code, but it makes great sense when you remember that at any instant the program might switch to some other routine that can affect the values of the program's variables.

Coverage monitors miss issues like these.

Missing code

The problem in the telephone example was that we had forgotten to write about 10 lines of code to handle one state transition. All of the lines of code that we did have worked. We tested all of the relevant lines of code and didn't find this bug.

If you divide A by B, and forget to check for B=0, the program doesn't crash because a bad line is present. It crashes because the line that was supposed to test for B=0 is not there. The program is broken, even though every line and every branch work properly.

Coverage monitors tell you whether you have tested every line and every branch. They don't tell you that you need more lines and branches.

Requirements-related errors

Suppose you're writing a program to compute taxes, and the government changed its tax rules such that income earned before May 5, 1997 is taxed at one rate and income earned after May 5 is taxed at a different rate. The programmer, having never read the tax laws, calculates using the current (post-May) rate without realizing that a different formula is needed before May 5.

A coverage monitor will show that every line is tested and working, but it will not help you discover that a system requirement (calculate taxes for January to May correctly) has not been met and will not help you discover that another requirement (calculate taxes for May 5 itself) is ambiguously specified because the law refers to dates before and after May 5.

Compatibility/configuration and other errors

Coverage-based testing won't tell you that the program works with one printer but not with another allegedly compatible one. It won't tell you that the documentation is error-ridden, that the program is unusably complex, that the program fails only on specific values, that the program fails when too many processes are running or too many computers are demanding attention at once.

Coverage-based testing tells you that you have traced though the flowchart. No more, no less. Flowcharts are good, but bugs know how to hide in them. Coverage-based testing is not complete testing unless we broaden our definition of coverage dramatically, in which case we discover 10^{100} possible tests and never achieve complete coverage.

What Does This Mean For Software Quality?

I wouldn't spend so much of my time writing about testing and teaching people to be better testers if I wasn't absolutely confident of the value of testing. But if you can never come close to completeness in testing, then you would be unwise to think of testing as "quality assurance." Testing groups that name themselves "QA" are misleading themselves and the company they work in.

Some testing groups add a few metrics-measurement, inspection, and standards-compliance functions. Do this make them "quality assurance" groups. No.

Think back to the telephone system described above. An important aspect of the reliability of that system was the management of time-related risks. Suppose that you were running a QA group and the programmers made a lot of time-related bugs. Could you send the programmers out for retraining? Could you require them to adopt the new practices? Could you give bonuses to the ones who made the most progress? If you don't have the authority to manage and train the programming staff, then you don't have the power to assure quality. If you do have that power, your title is probably Vice-President of Product Development, not Manager of Quality Assurance. Testing groups and testing-plus-other-measurements groups are doing Quality Assistance not assurance.

Quality is multidimensional. A stakeholder in a product will justifiably consider a product to be low quality if it doesn't satisfy his reasonable needs. Those needs differ widely across stakeholders. For example, even if the English language version of a product is reliable, a localization manager can reasonably call it unacceptable if her team has to recompile it in order to translate and adapt the program for another country. A support manager can reasonably call a reliable program unacceptable if it is impossible for customers to understand. A program that isn't written in a maintainable style won't stay reliable for long. And a customer might be happier with a program that occasionally crashes but is easy to use and that does the tasks that he needs, rather than a harder, less complete program whose failures can never be blamed on coding errors.

Software companies have to balance these different quality-related demands. To do it, they have to make tradeoffs during design and development. A drive to eliminate all coding errors from a program might not be the most effective drive to improve its overall quality.

Software publishers have to balance these demands because they have economic constraints. By spending a few dollars to prevent software failures, you *can* avoid wasting a huge amount of money on the consequences of those failures. For example, at an average cost of about $23 per technical support call, you can save $1 million by avoiding the need for about 43,500 calls. Several mass-market products have such high call volumes that they can save a fortune by investing in call volume reduction. But how much should a company spend if it doesn't know how many copies of the product it will sell? We make guesses about product volumes, but in my experience, many of those guesses are wrong. At what point do the direct cost (staff salaries), the sales lost due to delay, and the opportunity costs (you want to be building your next product) outweigh the benefit to be gained by finding and fixing more bugs? Must we run every test, in order to search for every bug? Are there no bugs that can be left unfixed? Quality is *sometimes* free (over the long term). Quality/cost analysis is rooted in tradeoffs. (Kaner, 1996a).

Remember too that there are significant costs of delayed time to market. As the best known example, a product that comes first to market will often sell much better than a technically superior product that comes out a few months later. What is the proper tradeoff between investment in quality and cost of delay in the marketplace?

The quality/cost tradeoffs are difficult. I don't think that it is possible today to release an affordable product that is error free and that fully meets the reasonable quality-related objectives of all of the stakeholders (including the customers). The goal of what is recently being called the "Good Enough Software" approach is to make sure that we make our tradeoffs consciously, that we look carefully at the real underlying requirements for the product and ask whether the product is good enough to meet them. Many of these requirements are not expressed clearly, and our initial statements of them might overstate or understate the actual needs. Our understanding of requirements shifts as we learn more about the product and the stakeholders. (Lawrence, 1997; Lawrence & Johnson, 1997).

Among the tradeoffs that we make are decisions about how extensively to look for bugs—if we can't do all possible test cases, we have to do less, and we will miss some. Our goal is "Good Enough Testing." (Bach, 1997a).

Another piece of the "good enough" approach is explicit recognition of the fact that software companies choose to not fix some known bugs. I was surprised recently to read that the "good enough" approach defines itself in terms of not fixing all the bugs, and then to read that no one really deliberately ships with known bugs. If you haven't read that claim yet, don't bother looking for it. But if you did, let me say first that every software company that I have ever personally worked with or consulted to has made conscious, deliberate decisions to release software with known bugs. This is not new—I wrote about this

practice of deferring bugs (we'll fix them later) as a normal practice in the industry in the first edition of my book (Kaner, 1988), extended the discussion of the bug deferral process in the second edition (Kaner, Falk & Nguyen, 1993), and no one has contacted me to tell me that this is unrealistic. I teach a course on testing at UC Berkeley Extension, at various hotels, at ASQC (now ASQ) San Francisco, and at many companies and the course includes extended discussion of risk analysis and persuasion techniques to minimize the number of inappropriately unfixed bugs. None of these students tell me that they fix all the bugs in their company. Many of them have shared with me the process they use for analyzing whether a particular bug must be fixed or not. As it applies to bug deferral decisions, which is only a small part of its scope, the goal of the Good Enough Software approach is to see that these bugs are analyzed thoughtfully and that decisions are made reasonably.

The Good Enough Software approach is difficult because we reject the easy formulations, like "Quality is Free" and "Test Everything." We live in a world of tradeoffs. We see imperfection as a fact of life. Our objective is to choose design, development, testing and deferral strategies that help us manage imperfection in the service of developing products that, despite their flaws, are excellent tools for their purposes. (Bach, 1997a,1997b).

Back to the Legal Stuff

What does it mean to accept that imperfections are inevitable, and that complete testing is impossible? Should we set a legal standard of perfection anyway and penalize manufacturers for all bugs, no matter how good the product is? I think not.

But there has to be a limit somewhere. It's one thing to release a product with a bug that you never found, despite reasonably good development (including testing) practices. It's a different matter when you release with a known bug that you didn't disclose to the customer. The customer uses the product in a normal way, runs into the bug and loses work and data, including hardware configuration data or data created by some other program. Recovering this will be expensive. If the software publisher knew of this problem, ran its cost/benefit analysis and chose to ship it anyway, why should the customer bear the cost of the bug?

And why should we protect publishers who choose to do virtually no testing? If they pass off their software as normal, finished product and it crashes all the time and does other damage, why should customers pay for the cost of that other damage?

The question of who pays for damage plays a big role in cost/benefit analyses (Kaner, 1996a; Kaner & Pels, 1997). If the publisher has minimal risk, it can afford to ship shoddier products.

Let me give you an example from the draft statute that is before NCCUSL. The draft addresses liability for viruses. The Reporter (senior author) of the draft wrote a cover memo to NCCUSL (Nimmer, 1997), in which he says that ***"Significant new consumer protections are proposed . . . The significant new protections include: creation of a non-disclaimable obligation of reasonable care to avoid viruses in the traditional mass market"*** (my italics). In this list of five significant new consumer protections, virus protection came first.

Look in the draft (NCCUSL, 1997) at Section 2B-313, Electronic Viruses. As advertised in the cover memo, 2B-313 does require software publishers to "exercise reasonable care to ensure that its performance or message when completed by it does not contain an undisclosed virus." This requirement applies to software products sold in stores, like Microsoft Works or a flight simulator game. For products sold over the internet or for products costing more than about $500 (there are other exceptions too), the

publisher is allowed to disclaim liability for viruses by burying a statement that "we didn't check for viruses" in the fine print of a license that you won't see until after you buy the product.

The section defines reasonable care as searching for "known viruses using any commercially reasonable virus checking software." I don't know any software publisher today who only looks for known viruses, and very few mass market publishers use only one virus checker. According to the reviews that I see on the web (PC World and CNET cover this from time to time), no virus checkers catch 100% of the known viruses. Further, virus creators sometimes design their virus to be undetectable by the current version of a specific virus checker. (See the discussion of Microsoft AV in Cobb, 1996.) Many publishers (I suspect that this is the large majority) use at least three virus checkers, at least in the Windows and Windows 95 markets. The statute is thus defining "reasonable care" as something less than the minimum that I've seen done by any even-slightly-concerned software publisher. I don't think that this is a drafter's accident—I've explained virus-related issues to the Drafting Committee at length, at two separate meetings.

If the publisher exercises what the statute defines as "reasonable care," you can't sue it for a virus on the disk.

Section 2B-313 also requires ***customers*** to exercise reasonable care to check for viruses, and defines reasonable care in the same way for customers as for publishers. If you don't, then under Section 2B-313(d) "A party is not liable if . . . the party injured by the virus failed to exercise reasonable care to prevent or avoid loss."

Let's see what happens if you buy a program at a store, that has a virus that trashes your hard disk. You pay money to a technician to recover (some of) your files and to get your system working again.

If the publisher checked for viruses by using only one virus checker, you (the customer) can't recover for your expenses and lost data because the publisher has exercised "reasonable care."

If the publisher didn't check for viruses, but you (or your child) trustingly installed the program on your computer without checking for a virus, you can't recover because you didn't exercise "reasonable care."

What if the publisher didn't check for a virus, but you did? Unfortunately, you missed it. You've exercised reasonable care and the publisher has not. Do you get reimbursed for recovery of your files and your system? Nope. After all, the publisher's duty is to check for a virus using one virus checker. Even though it didn't do that, you proved that if the publisher *had* checked for a virus using a commercially reasonably virus checker (yours), it wouldn't have found a virus. Therefore its failure to exercise reasonable care didn't cause the damage and therefore (this is how negligence law works, folks), the publisher is not liable.

OK, what if the publisher made it impossible to check its product for viruses until after you installed the program? For example, some manuals explicitly instruct you to turn off your virus checker during installation. You got the virus because you followed the instructions. *Now* do you get reimbursed for recovering your files and system? Nope. This is a consequential loss, and the publisher gets to disclaim all liability for consequential losses in the fine print of the you-can't-see-it-until-after-you-buy-the-product license.

When I was in law school, one of our professors defined the "blush test" by saying that an argument or a lawsuit is frivolous if you can't present it to the judge without blushing. I don't know how anyone could pass the blush test while saying that Section 313 provides a significant new consumer protection.

I Section 313 will probably be revised—it's just too obvious. But over the coming 18 months, many other claims will be made about the fairness and customer protection features of Article 2B. When you hear them, think about Section 313 and ask your friends and state legislators to look carefully before they believe that Article 2B will promote the marketing of good enough software or even provide basic incentives for honesty and fair dealing in the industry.

References

Abramowitz, M. & Stegun, I.E. (1964) *Handbook of Mathematical Functions.* Dover Publications.

Bach, J.S. (1997a) "Good enough testing for good enough software." *Proceedings of STAR 97 (Sixth International Conference on Software Testing, Analysis, and Review,* San Jose, CA., May 7, 1997, p. 659.

Bach, J.S. (1997b) *Is the Product Good Enough?* Unpublished manuscript, probably available at www.stlabs.com.

Cobb, S. (1996) *The NCSA Guide to PC and LAN Security.* 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 several websites, including www.badsoftware.com.

Kaner, C. (1988) *Testing Computer Software.* TAB Professional & Reference Books.

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

Kaner, C. (1996a) "Quality cost analysis: Benefits and risks." *Software QA*, vol. 3, #1, p. 23.

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, p. 313. An earlier version of this paper appeared in *Software QA Quarterly*, Volume 2, #2, 1995, p. 18.

Kaner, C. (1996c) "Negotiating testing resources: A collaborative approach." Presented at *Software Quality Week*, San Francisco, CA, May, 1996.

Kaner, C. & Pels, D. (1997) "Software customer dissatisfaction." *Software QA*, vol. 4, #3, p. 24.

Kit, E. (1995) *Software Testing in the Real World*. ACM Press: Addison-Wesley.

Lawrence, B. (1997, April) "Requirements happen." *American Programmer*, vol. 10, #4, p. 3.

Marick, B. (1997) "Classic testing mistakes." *Proceedings of STAR 97 (Sixth International Conference on Software Testing, Analysis, and Review,* San Jose, CA., May 7, 1997, p. 677.

Myers, G.J. (1979) *The Art of Software Testing.* Wiley.

National Conference of Commissioners on Uniform State Laws (1997) *Draft: Uniform Commercial Code Article 2B – Licenses: With Prefatory Note and Comments.* Available at www.law.upenn.edu/library/ulc/ulc.htm in the Article 2B section, under the name *1997 Annual Meeting*.

Nimmer, R. (1997) *Issues Paper: UCC Article 2B – Licenses.* Distributed to the National Conference of Commissioners on Uniform State Laws for their Annual Meeting, July 25 – August 1, 1997, Sacramento, CA.

1.
Cem Kaner. Ph.D.. J.D.
Law Office of Cem Kaner
kaner@kaner.com

P.O. Box 1200
Santa Clara. CA 95052
408-244-7000

SOFTWARE NEGLIGENCE AND TESTING COVERAGE[1]

Published in the Proceedings, STAR 96 (Fifth International Conference on Software Testing, Analysis, and Review), Orlando, Florida, May 16, 1996, p. 313.

Several months ago, a respected member of the software quality community posed the following argument to me:

> A program fails in the field, and someone dies. This leads to a trial. When the QA manager takes the stand, the plaintiff's lawyer brings out three facts:
>
> 1. The failure occurred when the program reached a specific line of code.
> 2. That line had never been tested, and that's why the bug had not been found before the product was released.
> 3. A ***coverage monitor*** was available. This is a tool that allows the tester to determine which lines of code have not yet been tested. Had the QA manager used the tool, and tested to a level of complete coverage (all lines tested), this bug would have been found, and the victim would be alive today.
>
> Therefore, the lawyer has proved that the company was negligent and the victim's family will win the lawsuit.

The question is, what's wrong with this argument? Anything? After all, the company had a well-understood tool readily available and if they had only used it, someone would not have died. *How could the company **not** be liable for negligence?*

OVERVIEW

This presentation explores the legal concept of *negligence* and the technical concept of *coverage*. The article advances several related propositions:

Coverage

1. The phrase, *complete coverage*, is misleading. This "completeness" is measured only relative to a specific population of possible test cases, such as lines of code, branches, n-length sub-paths, predicates, etc. Even if you achieve complete coverage for a given population of tests (such as, all lines of code tested), you have not done complete, or even adequate, testing.

[1] Portions of this paper were originally published in C. Kaner, "Software Negligence & Testing Coverage", ***Software QA Quarterly,*** Vol. 2, #2, p. 18, 1995.

2. We can and should expand the list of populations of possible test cases. We can measure coverage against each of these populations. The decision as to whether to try for 1%, 10%, 50% or 100% coverage against any given population is non-obvious. It involves tradeoffs base on thoughtful judgment.

Negligence

3. Negligence liability attaches when injury or loss is caused by a failure to satisfy a duty that was imposed by law, as a matter of public policy. So the question of whether or not it is *negligent* to fail to test every line of code turns into a question of whether the company had a public duty to test every line of code.
4. The nature of the duty involved depends on the role of the company in the development and release of the program.

   A company that publishes a software product has a duty to take reasonable measures to ensure that its products are safe.

   A company that sells software development services *might* have a duty to its client to deliver a program that is reasonably competently coded and tested.

   A company that acts as an independent test laboratory *might* owe a duty of competent testing and reporting to the client or to the public.
5. In any of these three cases, it is not obvious whether failure to achieve 100% line coverage is negligent. The plaintiff will have to *prove* that the tradeoff made by the software company was unreasonable.

WHAT IS NEGLIGENCE?

I'll start by considering the situation of the software developer/publisher. This provides the room to explore the coverage issues that are the focus of this paper. The situations of the service providers are of independent interest to the testing community and so will also be considered below.

Under negligence law, software development companies must not release products that pose an *unreasonable* risk of personal injury or property damage.[1] An injured customer can sue your company for negligence if your company did not take *reasonable measures* to ensure that the product was safe.

Reasonable measures are those measures that a reasonable, cautious company would take to protect the safety of its customers. How do we determine whether a company has taken reasonable measures? One traditional approach in law involves a simple cost-benefit analysis. This was expressed as a formula by Judge Learned Hand in the classic case of *United States v. Carroll Towing Co.*:[2]

[1] Note the restriction on negligence suits. Most lawsuits over defective software are for breach of contract or fraud, partially because they don't involve personal injury or property damage.

[2] ***Federal Reporter, Second Series,*** volume 159, page 169 (United States Court of Appeals, 2nd Circuit, 1947); for a more recent discussion see W. Landes and R. Posner, ***The Economic Structure of Tort Law,*** Harvard University Press, 1987.

Let **B** be the burden (expense) of preventing a potential accident.

Let **L** be the severity of the loss if the accident occurs.[1]

Let **P** be the probability of the accident.

Then *failure to attempt to prevent a potential accident is unreasonable if*

B < **P** x **L.**

For example, suppose that a software error will cause a total of $1,000,000 in damage to your customers. If you could prevent this by spending less than $1,000,000, but don't, you are negligent. If prevention would cost more than $1,000,000, and you don't spend the money, you are not negligent.

In retrospect, after an accident has occurred, now that we know that there is an error and what it is, it will almost always look cheaper to have fixed the bug and prevented the accident. But if the company didn't know about this bug when it released the program, our calculations should include the cost of finding the bug. *What would it have cost to make the testing process thorough enough that you would have found this bug during testing?*

For example, if a bug in line 7000 crashes the program, **B** would not be the cost of adding *one* test case that miraculously checks this line (plus the cost of fixing the line). **B** would be

> the cost of strengthening the testing so that line 7000's bug is found *in the normal course of testing,* or
>
> the cost of changing the design and programming practices in a way that would have prevented this bug (*and others like it*) in the first place.

Coming back to the coverage question, it seems clear that you can prevent the crash-on-line-7000 bug by making sure that you at least execute every line in the program. This is *line coverage*.

Line coverage measures the number / percentage of lines of code that have been executed. But some lines contain *branches*—the line tests a variable and does different things depending on the variable's value. To achieve complete *branch coverage,* you check each line, and each branch on multi-branch lines. To achieve complete *path coverage,* you must test every path through the program, an impossible task.[2]

The argument made at the start of this article would have us estimate **B** as the cost of achieving complete line coverage. *Is that the right estimate of what it would cost a reasonable software company to find this bug? I don't think so.*

Line coverage is just one narrow type of coverage of the program. Yes, complete line coverage would catch a syntax error on line 7000 that crashes the program, but what about all the other bugs that wouldn't show up under this simple testing? Suppose that it would cost an extra $50,000 to achieve complete line coverage. If you had an extra $50,000 to spend on testing, is line coverage what you would spend it on? *Probably not*.

Most traditional coverage measures look at the simplest building blocks of the program (lines of code) and the flow of control from one line to the next. These are easy and obvious measures to create, but they can miss important bugs.

[1] See C. Kaner "Quality Cost Analysis: Benefits and Risks" ***Software QA***, Vol. 3, No. 1, p. 23, 1996. The amount **P** x **L**, is an estimate of the External Failure Costs associated with this potential accident. Note that there are two different estimates of an external failure cost. You can estimate the amount it will cost your company if the failure occurs, or you can estimate how much it will cost your customers. Most cost-of-quality analyses will look at your company's costs. In negligence cases, judges and juries look at customers' losses.

[2] G. Myers, ***The Art of Software Testing*** Wiley, 1979.

A great risk of a measurement strategy is that it is too tempting to pick a few convenient measures and then ignore anything else that is more subtle or harder to measure. When people talk of *complete coverage* or *100% coverage*, they are using terribly misleading language. Many bugs will not be detected even if there is complete line coverage, complete branch coverage, or even if there were complete path coverage.

If you spend all of your extra money trying to achieve complete line coverage, you are spending none of your extra money looking for the many bugs that won't show up in the simple tests that can let you achieve line coverage quickly. Here are some examples:

> A key characteristic of object-oriented programming is that each object can deal with any type of data (integer, real, string, etc.) that you pass to it. Suppose that you pass data to an object that it wasn't designed to accept. The program might crash or corrupt memory when it tries to deal with it.. Note that you won't run into this problem by checking every line of code, because the failure is that the program doesn't expect this situation, therefore it supplies no relevant lines for you to tes
>
> There is an identifiable population of tests that can reveal this type of problem. If you pass every type of data to every object in your product, you will find every error that involves an object that doesn't properly handle a type of data that is passed to it. You can count the number of possible tests involved here, and you can track the number you've actually run. Therefore, we can make a coverage measure here.
>
> A Windows program might fail when printing[1]. You achieve complete coverage of printer compatibility tests (across printers) if you use the set of all Windows-supported printers, using all Windows printer drivers available for each of these printers. These drivers are part of the operatir system, not part of your program, but your program can fail or cause a system failure when working with them. The critical test case is not whether a particular line of code is tested, but whether it is tested in conjunction with a specific driver.
>
> Suppose that you test a desktop publishing program. One effective way to find bugs and usability failures is to use the program to create interesting documents. This approach is particularly effective if you use a stack of existing documents and try to make exact copies of them with your program. To create your stack, perhaps you'll use all the sample files and examples that come with PageMaker, Quark, FrameMaker, and one or two other desktop publishers. In this case, you achieve complete coverage if you recreate all of the samples provided by all available desktop publishers.

The Appendix to this article lists 101 measures of testing coverage. Line coverage is just one of many. There are too many possible tests for you to achieve complete coverage for every type of coverage in the list.

[1] I've been asked by software quality workers who are not familiar with mass-market issues why anyone would want to spend much effort on printer testing. Here is some relevant information. The technical support costs associated with printe incompatibilities are significant for several mass-market software companies. For example, at a presentation on "Wizards the ***OpCon West 96 Customer Service & Support Conference***, March 18, 1996, Keith Sturdivant reported that print/merge calls on Microsoft Word had been averaging over 50 support-minutes per caller until Microsoft developed a special print troubleshooting Wizard. In my experience at other companies, this is not a surprisingly high number. In "Benchmark Repo Technical Support Cost Ratios," ***Soft\ letter***, Vol. 10, #10, p. 1, September 21, 1993, Jeffrey Tarter reported an average tecl support call cost of $3 per minute. At this rate, the 50 minute printer call cost is $150 – for a product that is bundled free w a computer or sold at retail for as little as $99. The cost and aggravation to the customers is also very high.

I hope that the list helps you make priority decisions consciously and communicate them explicitly. The tradeoffs will differ across applications—in one case you might set an objective of 85% for line coverage,[1] 100% for data coverage, but only 5% for printer / driver compatibility coverage. For a different program whose primary benefit is beautiful output, you would assign printer coverage a much higher weight.

If you had an extra $50,000 to spend, would you focus your efforts on increasing line coverage or increasing some of the others? Surely, the answer should depend on the nature of your application, the types of risks involved in your application, and the probable effectiveness of the different types of tests. The most desirable strategy will be the one that is most likely to find the most bugs, or to find the most serious bugs.

The legal (negligence) test for the coverage tradeoffs that you make is *reasonability*. No matter what tradeoffs you make, and no matter how much money you spend on testing, you will miss some bugs.[2] Whether or not those bugs are products of negligence in the testing process depends on your reasonability, not on your luck in selecting just the right tests.

Your task is to prioritize among tests in the way that a reasonably careful company would—and to me that means to select the test strategy that you rationally believe is the most likely to find the most bugs or the most serious bugs.

There is no magic talisman in coverage that you can use blindly and be free of negligence liability. Being reasonable in your efforts to safeguard your customer requires careful thought and analysis. Achieving complete (line, branch, whatever) coverage will not insulate you. The plaintiff's attorney will just ask you why you spent all that money on line coverage, at the expense of, say, interrupt coverage. Try to assign your weights sensibly, in a way that you can explain and justify.

The same reasoning applies to customer satisfaction in general. If your approach will control the risks, you've done your job. But if you can identify gaps that leave an unreasonable degree of risk to customer safety or satisfaction, there is no reasonable alternative to addressing those risks.

As a final note, I hope that you'll take a moment to appreciate the richness, multidimensionality, and complexity of what we do as testers. Sometimes we hear that only programmers should be testers, or that all testing should be driven from a knowledge of the workings of the code. This list highlights the degree to which that view is mistaken. Programming skills and code knowledge are essential for glass box testing tasks, but as we explore the full range of black box testing approaches, we find that we also need skills and knowledge in:

the application itself (subject matter experts)

safety and hazard analysis

usability, task analysis, human error (human factors analysis)

hardware (modems, printers, etc.)

customer relations.

A person who has these skills but who can't program may be an invaluable member of a black box testing team.

[1] It can be ***very*** expensive and difficult to achieve 100% line or branch coverage. Grady reports that values of 80-85% are reasonably achievable. R.B. Grady, ***Practical Software Metrics for Project Management and Process Improvement***, PTR Prentice Hall (Hewlett-Packard Professional Books), 1992, p. 171.

[2] See Chapter 2 of C. Kaner, J. Falk, and H.Q. Nguyen, ***Testing Computer Software*** (2nd. Ed.), Van Nostrand Reinhold, 1993.

SOFTWARE MALPRACTICE

Malpractice? Can you be sued for malpractice? I've heard alarmist talk about software malpractice – the main point of this section is to say, "Calm down."

A person commits malpractice if she provides professional services that don't meet the level that would be provided by a reasonable member of the profession in this community. For example, if you hire a lawyer to draft a contract, and the lawyer makes mistakes that any reasonable lawyer would never make, then you can sue the lawyer for malpractice.

Many people in the software community sell services rather than products. We sell our programming or testing skills on a contract basis, doing work that is defined by our customer. What if you do a terrible job? In any contract for services, the service provider is under a duty to perform the services in a reasonable and workmanlike manner.[1] If you hand back badly written code that has never been tested, you might face claims for breach of contract, misrepresentation, negligence, or malpractice:

> You're probably liable for breach of contract if you promised working code and you delivered garbage.
>
> You're probably liable for misrepresentation if you said that you knew how to write this type of application, but you've never done one before.
>
> If the only harm caused by your product is economic (it cost your customer money), then there's an excellent chance that the courts will refuse to allow a negligence-based suit to go forward. If you do a lousy job of programming and testing, your customer should sue you for breach of contract.[2]
>
> If the suit does go forward as a negligence suit, then your customer may have to prove that no reasonable person would have done as bad a job as you. This population of "reasonable persons who sell software services" includes hobbyists, recent university graduates, high school students, and a variety of other odd characters. Proving negligence can be a big challenge.
>
> The malpractice case is much clearer and much more dangerous. The care you took in providing your services is compared to the level of care that is to be expected from a professional software service provider. If your work doesn't live up to that standard, and your customer loses money as a result, then it doesn't matter whether a hobbyist would have done the job more sloppily than you. You acted as a professional. You provided sub-professional services. You lose.

There's just a small problem in this malpractice case – what professional standards should your work be compared to?

It's relatively easy to compare lawyers' work to professional standards because there *are* professional standards. I belong to the California Bar. Can't be a lawyer in California without being a member of the State Bar. To be admitted to the Bar, I had to take a three-day exam on basic legal knowledge that every practicing lawyer is expected to have. And I had to take another tough exam on professional responsibility – my professional and ethical obligations to my clients and to the public. These exams illustrate a standard of knowledge that is accepted by the entire legal community. As an attorney, I am required by my profession's Code of Professional Responsibility to competently represent my clients. If I fail to follow this Code, I can be

[1] This is clearly discussed in R.T. Nimmer, ***The Law of Computer Technology: Rights, Licenses, Liabilities*** (2nd Ed.) Warren, Gorham & Lamont, 1992 (supplemented 1994). See Chapter 9 generally, especially section 9.16.

[2] This is a complex issue for lawyers, that probably shouldn't be a complex issue. The new Article 2B of the Uniform Commercial Code will clean up most or all of it by making this a clearly defined contract issue. For more on the drafting of Article 2B, see C. Kaner, "Uniform Commercial Code Article 2B: A New Law of Software Quality," in press, ***Software QA***, Vol. 3, No. 2, 1996. To read the latest draft of Article 2B, check the ***Uniform Commercial Code Article 2B Revision Home Page*** at **`http://www.law.uh.edu/ucc2b.`**

We have nothing comparable in the software community. Some groups have attempted to certify programmers but none has won general (let alone universal) acceptance. Similarly, there are no widely accepted standards for software testers.[1] There is also no enforceable code of ethics.

Someone who advertises their services in terms of an industry certification is inviting an evaluation against a higher standard. My letterhead advertises that I am an ASQC-Certified Quality Engineer (ASQC-CQE). This implicitly invites any (technical, rather than legal) client of mine to judge me against a standard of

[1] The American Society for Quality Control recently developed a Certification for Software Quality Engineers (CSQE). I contributed to this effort by working in the meeting that developed the final "body of knowledge" to be used in the certifica exam. As an ASQC-Certified Quality Engineer, I generally support the ASQC's certification efforts, and I think that there is genuine personal value to be gained from the study and effort required for a CSQE.

I don't want the following comments to be taken as an attack on the CSQE designation or process. They are not an attack. They are a statement of limitations, made by someone who was part of the process, and I am making them to help avoid the confusion that could result in someone trying to identify the CSQE Body of Knowledge as a legally enforceable community standard.

The CSQE Body of Knowledge questionnaire was given to a select group of people in the software community. It we to some ASQC members (I never received one, even though I am moderately active in ASQC, a member of ASQC's software division, and reasonably active in the software testing community. Nor did Hung Nguyen, even though he co-authored ***Testing Computer Software***, is a Senior Member of the ASQC, and is very active in the San Francisco ASQC chapter.) My understanding is that it was sent to a random sample of ASQC members. The questionnaire also went to several SPIN members and to attendees at one or two testing-related conferences. It missed several societies. For exam it never went to the Association for Computing Machinery membership, even though this is the largest computer-related professional society in the US. It never went to the Human Factors & Ergonomics Society, even though that organization includes many members who are directly involved in software quality (designing, testing and evaluating software user interfaces for safety, usability, etc.) I know several software testers who are not active in ASQC.

Survey results were contained in the working document ***The Profession of Software Quality Engineering: Results fron a Survey of Tasks and Knowledge Areas for the Software Quality Engineer – A Job Analysis Conducted on Behalf of the American Society for Quality Control*** by Scott Wesley, Ph.D. and Michael Rosenfeld, Ph.D. of Educational Testing Service This document notes than only 18.1% of the recipients of the survey responded, compared to 36%, 38%, and 48% respor rates for surveys for other ASQC designations.

The survey published several types of demographic information about respondents. It didn't analyze the responses i terms of software market segment, but my sense from looking at the other data was that relatively few came from the mas market software industry. It also appeared that relatively few people from the mass-market software industry contributed t the development of the questionnaire. I have limited expertise in some other areas, so I am less confident in identifying ot software markets as under-represented.

My sense is that software development and testing are not homogenous. I believe that we approach problems differently for mass-market consumer software than for life-critical diagnostic systems or for life-critical embedded system The differences are not merely in thoroughness or degree to which they are systematic. The approaches are qualitatively different. For example, some groups in Microsoft have what looks like a mature development and testing process, but it is an SEI-mature process. The ASQC-Certified Quality Engineer approach recognizes the diversity of the Quality Control community across industries. The ASQC-Certified Software Quality Engineer approach does not appear to recognize this diversity. The ASQC-CSQE Body of Knowledge appears to treat the field as homogenous.

The ASQC-CQE Body of Knowledge requires background knowledge in the history of the quality control movement. The ASQC-CSQE Body of Knowledge appears much less scholarly to me. Couple this with a lack of university training available for software quality, and I question the degree to which we can expect as a matter of law (for malpractice purpos that the typical software quality worker would have a thoughtful, historical insight into the techniques and approaches that she uses.

In sum, I believe that the CSQE Body of Knowledge represents an interesting body of knowledge to study. I believe that it would be good for an experienced member of the community to know the material covered by the CSQE. And I beli that a CSQE designation tells me a fair bit about the level of knowledge and commitment of a job candidate or coworker. I I do not believe that it is based on a representative study of the software quality community's practices and I do not believ that it can or should serve as a standard that is useful as evidence of community norms.

As noted in the main body, however, if someone says they are a CSQE in their advertising, they are representing themselves as knowledgeable of the CSQE Body of Knowledge and this invites comparison ***of that person's*** efforts to those one would expect from a CSQE.

knowledge that one would expect from an ASQC-CQE, which is a more precisely defined standard than that of a "reasonable person who does quality-related work." It therefore makes a claim of malpractice easier to argue (against me) in my case than in the general case. You can choose to expose yourself to that risk, or you can choose not to.

NEGLIGENT CERTIFICATION

An independent software testing company can get itself into all sorts of interesting trouble. Under the right circumstances, the lab can get itself into the same trouble as any other software service provider (see the discussions of product development negligence and malpractice above). In addition, let me mention two other well known court cases:

> *Hanberry v. Hearst Corporation*[1] A consumer sued *Good Housekeeping* magazine for negligent endorsement of a defectively designed shoe.
>
> *Hempstead v. General Fire Extinguisher Corporation*[2] A worker injured by the explosion of a fire extinguisher sued Underwriters' Laboratories for negligence in inspecting, testing, and approving the design of the extinguisher.

In both cases, the public was told that this was a product that had been evaluated and approved. The organization that had allegedly evaluated and approved the product was sued.

If the public is told that, *because it was you who tested the product*, the public should believe that a product that you tested is reliable and safe, then that product had better be reliable and safe. Otherwise, cranky and injured customers will come to your door too, saying that they bought the product because they trusted your recommendation.

APPENDIX: THE MANY TYPES OF TESTING COVERAGE

This appendix lists 101 coverage measures. *Coverage* measures the amount of testing done of a certain type. Because testing is done to find bugs, coverage is also a measure of your effort to detect a certain class of potential errors. For example, 100% line coverage doesn't just mean that you've executed every line of code; it also means that you've tested for every bug that can be revealed by simple execution of a line of code.

Please note that this list is far from complete. For example, it doesn't adequately cover safety issues.[3] Nor does it convey the richness of the tests and test strategies that you can derive from customer complaints and surveys and from tests involving controlled customer observation. And you will add measures as you analyze the application that you're testing.

1. ***Line coverage.*** Test every line of code (Or ***Statement coverage:*** test every statement).
2. ***Branch coverage.*** Test every line, and every branch on multi-branch lines.

3. N-length sub-path coverage. Test every sub-path through the program of length N. For example, in a 10,000 line program, test every possible 10-line sequence of execution.

[1] California Appellate Reports, 2nd Series, Vol. 276, p. 680 (California Court of Appeal, 1969).

[2] Federal Supplement, Vol. 269, p. 109 (United States District Court for the District of Delaware, applying Virginia law 1969).

[3] N. Leveson, ***Safeware: System Safety and Computers***, Addison-Wesley, 1995.

4. Path coverage. Test every path through the program, from entry to exit. The number of paths is impossibly large to test.[1]

5. Multicondition or predicate coverage[2] Force every logical operand to take every possible value. Two different conditions within the same test may result in the same branch, and so branch coverage would only require the testing of one of them.

6. Trigger every assertion check in the program. Use impossible data if necessary.

7. Loop coverage. "Detect bugs that exhibit themselves only when a loop is executed more than once."[3]

8. Every module, object, component, tool, subsystem, etc. This seems obvious until you realize that many programs rely on off-the-shelf components. The programming staff doesn't have the source code to these components, so measuring line coverage is impossible. At a minimum (which is what is measured here), you need a list of all these components and test cases that exercise each on at least once.

9. Fuzzy decision coverage. If the program makes heuristically-based or similarity-based decisions, and uses comparison rules or data sets that evolve over time, check every rule several times over the course of training.

10. Relational coverage. "Checks whether the subsystem has been exercised in a way that tends to detect off-by-one errors" such as errors caused by using < instead of <=[4]. This coverage includes:

> ***Every boundary on every input variable***[5]
>
> ***Every boundary on every output variable.***
>
> ***Every boundary on every variable used in intermediate calculations.***

11. Data coverage. At least one test case for each data item / variable / field in the program.

12. Constraints among variables: Let X and Y be two variables in the program. X and Y constrain each other if the value of one restricts the values the other can take. For example, if X is a transaction date and Y is the transaction's confirmation date, Y can't occur before X.

13. Each appearance of a variable. Suppose that you can enter a value for X on three different data entry screens, the value of X is displayed on another two screens, and it is printed in five reports. Change X at each data entry screen and check the effect everywhere else X appears.

14. Every type of data sent to every object. A key characteristic of object-oriented programming is that each object can handle any type of data (integer, real, string, etc.) that you pass to it. So, pass every conceivable type of data to every object.

15. Handling of every potential data conflict. For example, in an appointment calendaring program, what happens if the user tries to schedule two appointments at the same date and time?

[1] See G. Myers, ***The Art of Software Testing*** Wiley, 1979, and Chapter 2 of C. Kaner, J. Falk, and H.Q. Nguyen, ***Testing Computer Software*** (2nd. Ed.), Van Nostrand Reinhold, 1993.

[2] G. Myers, ***The Art of Software Testing*** Wiley, 1979 (multicondition coverage) and B. Beizer, ***Software Testing Techniques*** (2nd Ed.), Van Nostrand Reinhold, 1990.

[3] B. Marick, ***The Craft of Software Testing*** Prentice Hall, 1995, p. 146.

[4] B. Marick, ***The Craft of Software Testing*** Prentice Hall, 1995, p. 147.

[5] Boundaries are classically described in numeric terms, but any change-point in a program can be a boundary. If t program works one way on one side of the change-point and differently on the other side, what does it matter whet the change-point is a number, a state variable, an amount of disk space or available memory, or a change in a document from one typeface to another, etc.? See C. Kaner, J. Falk, and H.Q. Nguyen, ***Testing Computer Software*** (2nd. Ed.), Van Nostrand Reinhold, 1993, p. 399-401.

16. Handling of every error state. Put the program into the error state, check for effects on the stack, available memory, handling of keyboard input. Failure to handle user errors well is an important problem, partially because about 90% of industrial accidents are blamed on human error or risk-taking.[1] Under the legal doctrine of *foreseeable misuse*,[2] the manufacturer is liable in negligence if it fails to protect the customer from the consequences of a reasonably foreseeable misu of the product.

17. Every complexity / maintainability metric against every module, object, subsystem, etc. There are many such measures. Jones[3] lists 20 of them.[4] People sometimes ask whether any of these statistics are grounded in a theory of measurement or have practical value.[5] But it is clear that, in practice, some organizations find them an effective tool for highlighting code that needs further investigation and might need redesign.[6]

18. Conformity of every module, subsystem, etc. against every corporate coding standard. Several companies believe that it is useful to measure characteristics of the code, such as total lines per module, ratio of lines of comments to lines of code, frequency of occurrence of certain types of statements, etc. A module that doesn't fall within the "normal" range might be summarily rejected (bad idea) or re-examined to see if there's a better way to design this part of the program.

19. Table-driven code. The table is a list of addresses or pointers or names of modules. In a traditional CASE statement, the program branches to one of several places depending on the value an expression. In the table-driven equivalent, the program would branch to the place specified in, say, location 23 of the table. The table is probably in a separate data file that can vary from day to day or from installation to installation. By modifying the table, you can radically change the control flow of the program without recompiling or relinking the code. Some programs drive a great deal of their control flow this way, using several tables. Coverage measures? Some examples:

> check that every expression selects the correct table element
>
> check that the program correctly jumps or calls through every table element
>
> check that every address or pointer that is available to be loaded into these tables is vali (no jumps to impossible places in memory, or to a routine whose starting address has changed)
>
> check the validity of every table that is loaded at any customer site.

20. Every interrupt. An interrupt is a special signal that causes the computer to stop the program in progress and branch to an interrupt handling routine. Later, the program restarts from where it was interrupted. Interrupts might be triggered by hardware events (I/O or signals from the clock that a specified interval has elapsed) or software (such as error traps). Generate every type of interrupt in every way possible to trigger that interrupt.

[1] B.S. Dhillon, ***Human Reliability With Human Factors***, Pergamon Press, 1986, p. 153.

[2] This doctrine is cleanly explained in S. Brown (Ed.), ***The Product Liability Handbook: Prevention, Risk, Consequenc and Forensics of Product Failure***, Van Nostrand Reinhold, 1991, pp. 18-19.

[3] C. Jones, ***Applied Software Measurement***, McGraw-Hill, 1991, p. 238-341.

[4] B. Beizer, ***Software Testing Techniques*** (2nd Ed.), Van Nostrand Reinhold, 1990, provides a sympathetic introduction to these measures. R.L. Glass, ***Building Quality Software***, Prentice Hall, 1992, and R.B. Grady, D.L. Caswell, ***Software Metrics: Establishing a Company-Wide Program***, Prentice Hall, 1987, provide valuable perspective

[5] For example, C. Kaner, J. Falk, and H.Q. Nguyen, ***Testing Computer Software*** (2nd Ed.), Van Nostrand Reinhold, 1993, pp. 47-48; also R.L. Glass, ***Building Quality Software***, Prentice Hall, 1992, "Software metrics to date have not produced any software quality results which are useful in practice" p. 303.

[6] R.B. Grady, D.L. Caswell, ***Software Metrics: Establishing a Company-Wide Program***, Prentice Hall, 1987 and R.B. Grady, ***Practical Software Metrics for Project Management and Process Improvement***, PTR Prentice Hall (Hewlett-Packard Professional Books), 1992, p. 87-90.

21) ***Every interrupt at every task, module, object, or even every line.*** The interrupt handling routine might change state variables, load data, use or shut down a peripheral device, or affect memory in way that could be visible to the rest of the program. The interrupt can happen at any time—between any two lines, or when any module is being executed. The program may fail if the interrupt is handled at a specific time. (Example: what if the program branches to handle an interrupt while it's in the middle o writing to the disk drive?)

The number of test cases here is huge, but that doesn't mean you don't have to think about this type of testing. This is path testing through the eyes of the processor (which asks, "What instruction do I execute next?" and doesn't care whether the instruction comes from the mainline code or from an interrupt handler) rather than path testing through the eyes of the reader of the mainline code. Especially in programs that have global state variables, interrupts at unexpected times can lead to ver odd results.

22) ***Every anticipated or potential race.***[1] Imagine two events, **A** and **B**. Both will occur, but the program is designed under the assumption that **A** will always precede **B**. This sets up a race between **A** and **B**—if **B** ever precedes **A**, the program will probably fail. To achieve race coverage, you must identify every potential race condition and then find ways, using random data or systematic test case selection, to attempt to drive **B** to precede **A** in each case.

Races can be subtle. Suppose that you can enter a value for a data item on two different data entry screens. User 1 begins to edit a record, through the first screen. In the process, the program locks the record in Table 1. User 2 opens the second screen, which calls up a record in a different table, Table 2. The program is written to automatically update the corresponding record in the Table 1 when User 2 finishes data entry. Now, suppose that User 2 finishes before User 1. Table 2 has been updated, but th attempt to synchronize Table 1 and Table 2 fails. What happens at the time of failure, or later if the corresponding records in Table 1 and 2 stay out of synch?

23) ***Every time-slice setting.*** In some systems, you can control the grain of switching between tasks or processes. The size of the time quantum that you choose can make race bugs, time-outs, interrupt-related problems, and other time-related problems more or less likely. Of course, coverage is an difficu problem here because you aren't just varying time-slice settings through every possible value. You alsc have to decide which tests to run under each setting. Given a planned set of test cases per setting, the coverage measure looks at the number of settings you've covered.

24) ***Varied levels of background activity.*** In a multiprocessing system, tie up the processor with competing, irrelevant background tasks. Look for effects on races and interrupt handling. Similar to time-slices, your coverage analysis must specify

- categories of levels of background activity (figure out something that makes sense) and
- all timing-sensitive testing opportunities (races, interrupts, etc.).

25) ***Each processor type and speed.*** Which processor chips do you test under? What tests do you run under each processor? You are looking for:

- speed effects, like the ones you look for with background activity testing, and
- consequences of processors' different memory management rules, and
- floating point operations, and
- any processor-version-dependent problems that you can learn about.

26) ***Every opportunity for file / record / field locking.***

[1] Here as in many other areas, see Appendix 1 of C. Kaner, J. Falk, and H.Q. Nguyen, ***Testing Computer Software*** (2nd Ed.), Van Nostrand Reinhold, 1993 for a list and discussion of several hundred types of bugs, including interrupt-related, race-condition-related, etc.

27) ***Every dependency on the locked (or unlocked) state of a file, record or field.***

28) ***Every opportunity for contention for devices or resources.***

29) ***Performance of every module / task / object.*** Test the performance of a module then retest it during the next cycle of testing. If the performance has changed significantly, you are either lookin at the effect of a performance-significant redesign or at a symptom of a new bug.

30) ***Free memory / available resources / available stack space at every line or on entry into an exit out of every module or object.***

31) ***Execute every line (branch, etc.) under the debug version of the operating system.*** This shows illegal or problematic calls to the operating system.

32) ***Vary the location of every file.*** What happens if you install or move one of the program's component, control, initialization or data files to a different directory or drive or to another comput on the network?

33) ***Check the release disks for the presence of every file.*** It's amazing how often a file vanishes. If you ship the product on different media, check for all files on all media.

34) ***Every embedded string in the program.*** Use a utility to locate embedded strings. Then find a wa to make the program display each string.

Operation of every function / feature / data handling operation under:

35) ***Every program preference setting.***

36) ***Every character set, code page setting, or country code setting.***

37) ***The presence of every memory resident utility (inits, TSRs).***

38) ***Each operating system version.***

39) ***Each distinct level of multi-user operation.***

40) ***Each network type and version.***

41) ***Each level of available RAM.***

42) ***Each type / setting of virtual memory management.***

43) ***Compatibility with every previous version of the program.***

44) ***Ability to read every type of data available in every readable input file format.*** If a file format is subject to subtle variations (e.g. CGM) or has several sub-types (e.g. TIFF) or versions (e.g dBASE), ***test each one.***

45) ***Write every type of data to every available output file format.*** Again, beware of subtle variations in file formats—if you're writing a CGM file, full coverage would require you to test your program's output's readability by ***every one*** of the main programs that read CGM files.

46) ***Every typeface supplied with the product.*** Check all characters in all sizes and styles. If your program adds typefaces to a collection of fonts that are available to several other programs, check compatibility with the other programs (nonstandard typefaces will crash some programs).

47) ***Every type of typeface compatible with the program.*** For example, you might test the program with (many different) TrueType and Postscript typefaces, and fixed-sized bitmap fonts.

48) ***Every piece of clip art in the product.*** Test each with this program. Test each with other programs that should be able to read this type of art.

49. ***Every sound / animation provided with the product.*** Play them all under different device (e.g. sound) drivers / devices. Check compatibility with other programs that should be able to play this clip-content.
50. ***Every supplied (or constructible) script*** to drive other machines / software (e.g. macros) / BBS's and information services (communications scripts).
51. ***All commands available in a supplied communications protocol.***
52. ***Recognized characteristics.*** For example, every speaker's voice characteristics (for voice recognition software) or writer's handwriting characteristics (handwriting recognition software) or every typeface (OCR software).
53. ***Every type of keyboard and keyboard driver.***
54. ***Every type of pointing device and driver at every resolution level and ballistic setting.***
55. ***Every output feature with every sound card and associated drivers.***
56. ***Every output feature with every type of printer and associated drivers at every resolution level.***
57. ***Every output feature with every type of video card and associated drivers at every resolution level.***
58. ***Every output feature with every type of terminal and associated protocols.***
59. ***Every output feature with every type of video monitor and monitor-specific drivers at every resolution level.***
60. ***Every color shade displayed or printed to every color output device (video card / monitor / printer / etc.) and associated drivers at every resolution level.*** And check the conversion to grey scale or black and white.
61. ***Every color shade readable or scannable from each type of color input device at every resolution level.***
62. ***Every possible feature interaction between video card type and resolution, pointing device type and resolution, printer type and resolution, and memory level.*** This may seem excessively complex, but I've seen crash bugs that occur only under the pairing of specific printer and video drivers at a high resolution setting. Other crashes required pairing of a specific mouse and printer driver, pairing of mouse and video driver, and a combination of mouse driver plus video driver plus ballistic setting.
63. ***Every type of CD-ROM drive, connected to every type of port (serial / parallel / SCSI) and associated drivers.***
64. ***Every type of writable disk drive / port / associated driver.*** Don't forget the fun you can have with removable drives or disks.
65. ***Compatibility with every type of disk compression software.*** Check error handling for every type of disk error, such as full disk.
66. ***Every voltage level from analog input devices.***
67. ***Every voltage level to analog output devices.***
68. ***Every type of modem and associated drivers.***
69. ***Every FAX command (send and receive operations) for every type of FAX card under every protocol and driver.***

70. ***Every type of connection of the computer to the telephone line (direct, via PBX, etc.; digita vs. analog connection and signaling); test every phone control command under every telephone control driver.***
71. ***Tolerance of every type of telephone line noise and regional variation (including variations that are out of spec) in telephone signaling (intensity, frequency, timing, other characteristics of ring / busy / etc. tones).***
72. ***Every variation in telephone dialing plans.***
73. ***Every possible keyboard combination.*** Sometimes you'll find trap doors that the programmer used as hotkeys to call up debugging tools; these hotkeys may crash a debuggerless program. Other times, you'll discover an Easter Egg (an undocumented, probably unauthorized, and possibly embarrassing feature). ***The broader coverage measure is every possible keyboard combination at every error message and every data entry point.*** You'll often find different bugs when checking different keys in response to different error messages.
74. ***Recovery from every potential type of equipment failure.*** Full coverage includes each type of equipment, each driver, and each error state. For example, test the program's ability to recover fro full disk errors on writable disks. Include floppies, hard drives, cartridge drives, optical drives, etc. Include the various connections to the drive, such as IDE, SCSI, MFM, parallel port, and serial connections, because these will probably involve different drivers.
75. ***Function equivalence.*** For each mathematical function, check the output against a known good implementation of the function in a different program. Complete coverage involves equivalence testing of all testable functions across all possible input values.
76. ***Zero handling.*** For each mathematical function, test when every input value, intermediate variable, or output variable is zero or near-zero. Look for severe rounding errors or divide-by-zero errors.
77. ***Accuracy of every graph,*** across the full range of graphable values. Include values that force shift in the scale.
78. ***Accuracy of every report.*** Look at the correctness of every value, the formatting of every page, an the correctness of the selection of records used in each report.
79. ***Accuracy of every message***
80. ***Accuracy of every screen.***
81. ***Accuracy of every word and illustration in the manual.***
82. ***Accuracy of every fact or statement in every data file provided with the product.***
83. ***Accuracy of every word and illustration in the on-line help.***
84. ***Every jump, search term, or other means of navigation through the on-line help.***
85. ***Check for every type of virus / worm that could ship with the program.***
86. ***Every possible kind of security violation of the program, or of the system while using the program.***
87. ***Check for copyright permissions for every statement, picture, sound clip, or other creation provided with the program.***
88. ***Verification of the program against every program requirement and published specification.***

89. ***Verification of the program against user scenarios.*** Use the program to do real tasks that are challenging and well-specified. For example, create key reports, pictures, page layouts, or other documents events to match ones that have been featured by competitive programs as interesting output or applications.

90. ***Verification against every regulation (IRS, SEC, FDA, etc.) that applies to the data or procedures of the program.***

Usability tests of:

91. ***Every feature / function of the program.***

92. ***Every part of the manual.***

93. ***Every error message.***

94. ***Every on-line help topic.***

95. ***Every graph or report provided by the program.***

Localizability / localization tests:

96. ***Every string.*** Check program's ability to display and use this string if it is modified by changin the length, using high or low ASCII characters, different capitalization rules, etc.

97. ***Compatibility with text handling algorithms under other languages (sorting, spell checking, hyphenating, etc.)***

98. ***Every date, number and measure in the program.***

99. ***Hardware and drivers, operating system versions, and memory-resident programs tha are popular in other countries.***

100. ***Every input format, import format, output format, or export format that would be commonly used in programs that are popular in other countries.***

101. ***Cross-cultural appraisal of the meaning and propriety of every string and graphic shipped with the program.***

Notes

Law of Software Quality

Section 13.

Strict Products Liability

Strict Products Liability

A company is liable to a victim if it sells a product that is:

- ☒defective, and
- ☒unreasonably dangerous, and
- ☒the cause of personal injury or property damage

A design may be unreasonably dangerous if it fails to meet the safety expectations of a reasonable consumer.

Law of Software Quality

Section 14.

Malpractice

Malpractice

Malpractice is the failure to exercise the skill and knowledge normally possessed by members of a profession or trade.

☒ **Programmer malpractice?**

Getting licensed means that we will finally qualify to be sued in malpractice.

☒ **Professional advice malpractice?**

The more your advertising & docs make a customer think she can replace a professional with your product, the better her chances of success in a malpractice suit if your program gives bad advice.

Malpractice

Software Developers

When we develop custom software, as service providers, we are required to take reasonable care in the provision of our services.

Is our required level of care determined by professional standards (malpractice liability) or by the ordinary care of any reasonably prudent person?

(I think the answer is ordinary care, not malpractice.)

Malpractice

Independent Test Labs

- ☒Test labs, as service providers, are responsible to their clients (just like developers).
- ☒The additional question that I am posing is whether test labs might be subject to negligence liability to end customers?
- ☒There are traps here for the unwary.

Notes

Notes

Notes