Software Engineering Malpractice and Its Avoidance

Christopher J. Palermo, Esq.
Christie, Parker & Hale, P. O. Box 7068, Pasadena, California 91109-7068

Abstract: "Software engineering malpractice" is an emerging legal theory which will impose new legal liability on software engineers and software developers for errors in software or unreliable software. This paper summarizes software engineering malpractice court decisions, and discusses how a software engineering malpractice case can arise in a variety of contexts encountered in software development. The paper explores preventative measures that software engineers and developers can take to help insulate themselves from malpractice liability. The paper concludes that proper preventative measures should be an integral part of any software reliability effort, and that such practices can help the software engineering profession avoid becoming a target of plaintiffs' attorneys.

1: Overview

Shortly after takeoff from Los Angeles International Airport, over water, the thrust reversers on an Aircorp 676 SuperJet suddenly engage. The aircraft tumbles, crashing into the ocean. All lives are lost.

A legion of lawyers, representing families of deceased passengers, file lawsuits against Aircorp for negligence and wrongful death. During preparation for trial, the lawyers discover that the cause of the accident was a coding error in the aircraft’s control software. Further investigation reveals that the coding error was in a module prepared by MicroFinn, a software development company which prepared the control software under contract to Aircorp. Coding was done by MicroFinn's high-profile president, Bill Applegates.

Aircorp had an enormous operating loss last year, and as the lawsuit progresses the lawyers realize that Aircorp will never have enough assets to compensate the hundreds of victims. But MicroFinn is riding a wave of record profits. The lawyers change the course of the litigation, filing suit against MicroFinn and Bill Applegates for negligence and "software engineering malpractice."

An impossible scenario? Definitely not. Embedded software serves critical control functions in rail and air traffic control, automobiles, medical systems, financial applications, and military systems. When this software fails, human injury is risked. These risks are well documented. [1]

At least two court cases have involved a theory of malpractice liability for errors in software. Under current law, such a case could be won. Unfortunately, software engineers and their employers rarely take preventative measures to avoid legal liability.

This paper reviews the state of current law regarding malpractice liability of software engineers and software developers for errors in software. The paper then discusses how a software engineering malpractice case can arise in custom software development. Finally, the paper explores preventative measures that software engineers and developers can take to help insulate themselves from malpractice liability. The paper concludes that proper preventative practices are essential to any software reliability engineering effort, and can help the software engineering profession avoid becoming a target of plaintiffs' attorneys.

2: Software Engineering Malpractice: Nature and Status

2.1: Background

In general, persons can incur legal liability for acts relating to computers based on the criminal law, contract law, tort law, and other civil law theories. Tort law includes intentional torts (such as assault or battery) and negligent torts. "Software engineering malpractice" is a variety of negligence.

Torts may be viewed in a hierarchy organized by the amount of fault required for liability to arise.
Intentional tort liability requires intent to injure; strict liability, primarily imposed against manufacturers of dangerous products, requires no intent or fault. Negligence may be defined as conduct which falls below the standard of care established by law for the protection of others against unreasonable risk of harm [25].

The complexity of many software systems makes it difficult to establish a case of negligence. Confronted with a complex software system, a court may be unable to tell whether a programmer met the required standard of conduct. Most cases of computer negligence which reach the courts involve a contract consented to by the parties, and this relationship of consent may cause judges to refuse malpractice complaints. In addition, malpractice has traditionally protected against personal injury whereas most harm arising from software failure is economic rather than personal. The courts are also unfamiliar with software engineering malpractice cases since most computer negligence cases have involved hardware. Nevertheless, negligence actions against software developers have been gaining increasing attention, in part resulting from mass-media coverage of virus threats [17].

A plaintiff (the allegedly injured party) can state a negligence case by filing a lawsuit which alleges that (1) the defendant had a legal duty to protect the plaintiff against unreasonable harm; (2) the defendant breached that duty; (3) the breach was the cause of the harm done to plaintiff; and (4) the plaintiff did in fact suffer harm as a result of the breach. These four elements are often abbreviated by attorneys as "duty," "breach," "proximate cause," and "damages." Each is discussed below.

2.2: Duty

Negligence is conduct which fails to meet an established legal standard [25]. The standard is that of "a fictitious person ... the reasonable man of ordinary prudence" [26]. Thus, negligence arises from the defendant's failure to meet a duty of due care under the circumstances. The plaintiff will always try to establish the highest possible standard of conduct for an allegedly negligent defendant.

Traditionally, the law has held all professionals to a higher, "professional malpractice" standard of conduct. Such professionals as attorneys, accountants, physicians, dentists, and even clergy are expected to possess a standard minimum of special knowledge and ability. The defendant must have and use the knowledge, skill and care ordinarily possessed and used by members of the profession in good standing. The defendant will be liable if he does not have the expected knowledge and skill.

The standard of the profession can be established by law or through formal programs such as professional licensing [37]. Thus, professions have the legal advantage of establishing their own standards of contact, which laypersons cannot do; but a disadvantage is that once the standard is established, every member of the profession must meet it.

There are numerous cases applying a professional malpractice standard of care to architects and construction engineers [26]. But there is no agreement over whether software engineers, computer programmers, computer installers, etc. are "professionals" subject to a malpractice standard of conduct [27, 28]. The courts have failed to resolve the matter.

Malpractice has arisen in a computer sales setting, not involving negligence of software engineers, in at least three cases [20, 21, 22]. Different outcomes occurred based on differences in the facts.

In the first case, Invacare Corp. v. Sperry Corporation [20], the court was willing to apply general negligence law (not labeled malpractice) to computer salespeople.

The plaintiff, Invacare Corporation, alleged that Sperry Corporation was negligent in recommending its System 80 and that "it knew, or in the exercise of ordinary care, it should have known, that the systems were totally inadequate to provide a closed loop manufacturing system for plaintiff, in that the equipment delivered by defendant was incapable of operating the programs and related data processing products furnished by defendant and the programs and related data processing were inadequate". Invacare also alleged that Sperry was negligent in telling Invacare that the programs Sperry furnished would satisfy Invacare's needs, and that Invacare would not need a data processing consultant to operate the system.

Sperry referred to these allegations as "computer malpractice," but the Court held that the case could proceed on the basis of ordinary negligence:

... Invacare alleges that the personnel provided by Sperry failed to perform at a level of ordinary care. If machinists, electricians, carpenters, blacksmiths, and plumbers are held to the ordinary standard of care in their professions, the Court fails to see why personnel in the computer industry should be held any lower standard of care. Invacare simply alleges negligence in a
business setting. This does not give rise to a new tort of "computer malpractice." Negligence in a business setting is clearly actionable.

By labeling the case as one of ordinary negligence, the Court prevented Invacare from alleging that Sperry's personnel were expert professionals subject to a high duty of care, so that even a slight error in judgment would cause liability. Instead, Invacare would have to prove that Sperry's personnel failed to meet a standard of ordinary care for similar persons under the circumstances. An error, if reasonable for a similar person under the circumstances, would not result in liability for Sperry.

The difference in the standard of care was underscored in Invacare v. Sperry by a comparison with Chatlos Systems v. National Cash Register Corporation [21]. In Chatlos the plaintiff sought to apply "computer malpractice" to a company which sold and installed a complex computer system. The court observed that "[p]laintiff equates the sale and servicing of computer systems with established theories of professional malpractice. Simply because an activity is technically complex and important to the business community does not mean that greater potential liability must attach."

Relying on this holding, the Invacare Court pointed out: ... [Chatlos] was concerned with creating a new tort on a theory of elevated responsibility on the part of those who render computer sales and service. Such a concept is not before this Court. Rather, Invacare's claims allege a breach of the ordinary standard of care to which those in Sperry's industry are held. Such allegations do not involve a new tort of "computer malpractice."

Note that if software engineers increase the ordinary standard of care, by adopting voluntary standards or certification programs, even under Invacare regular negligence cases will become easier to prove.

No court has found that the supplier of a computer system owes the customer anything more than a duty of ordinary care. Suppliers have not been deemed experts or specialists so that even a slight error would result in liability. However, professional consultants who assist in preparing system requirements may be held to a high standard of professional care, especially when those standards have been promulgated by the national association of the profession [22].

For example, in the Diversified Graphics case [22], the plaintiff claimed that it hired an accounting firm to act as management consultants to locate a "turnkey" system "which would be fully operational without need of extensive employee training." The delivered system was unsatisfactory and Diversified Graphics sued for negligence and breach of contract. The jury found the accounting firm was negligent but that it had not breached the contract.

The accounting firm appealed only the negligence issue. Noting that plaintiff claimed to have retained the accounting firm as a consultant during the acquisition and installation of the system, the appeals court held: It is implicit in alleging the existence of an agreement that [Diversified] anticipated that [the accounting firm] possessed superior knowledge in this area; D.G. contracted for the benefit of [the accounting firm's] expertise. Based on D.G.'s allegations, [the accounting firm] was properly held to a professional standard of care.

Further, the appeals court found that Diversified had met the higher standard of proof required for breach of professional care. In concluding that the jury received sufficient evidence of this standard, the court noted that standards of the American Institute of Certified Public Accountants require that "due professional care be exercised when providing such services."

Diversified Graphics has profound implications for the software development profession. In the case, an accounting firm was held liable for malpractice because it held itself out as an expert management consulting firm in the area of systems design and analysis. It seems likely that a court could just as easily impose malpractice liability for minor errors committed by systems consultants, software developers, or VARs who hold themselves out as systems development experts.

In addition, under the logic of this case, professional standards developed for software reliability may well provide guidance for courts in ruling on software engineering malpractice cases. The accreditation standards currently under consideration by ACM and other organizations could, ironically, lead to exposure of software engineers to liability for failing to meet those standards. If Diversified Graphics had been a malpractice case involving software engineers, the court could have looked to professional computing standards and measured the defendant software engineer's conduct against those standards. The courts often take guidance from voluntarily imposed standards created by professional associations.

One possible escape route is the fact that current standards efforts have focused on what standards must be met by programs rather than by software engineers [2]. Of course, the law could find liability for failure to meet either kind of standard. A court could draw an analogy to an established body of tort cases involving civil engineering firms[31], in which the firms are held.
liable if the work fails to meet known professional standards.

For the software engineer, there are two particularly alarming cases, arising in Indiana [23] and Illinois [24]. The Indiana case could be considered the country's first example of a true software engineering malpractice case, although that term was not used, and the case involved liability of a developer for harm to a customer rather than to a third party.

In the Indiana case [23], Data Processing Services, Inc. ("DPS") agreed to provide accounting software for L.H. Smith Oil Corp.'s IBM System/32 and System/34 computers. After having paid several invoices, L.H. Smith refused to pay more, claiming that the job had not been done properly. The trial court found in favor of L.H. Smith, on the ground that DPS had breached an implied promise of having reasonable skill and ability to do the job. This finding was affirmed on appeal; the court stated:

Those who hold themselves out to the world as possessing skill and qualifications in their respective trades or professions impliedly represent they possess the skill and will exhibit the diligence ordinarily possessed by well informed members of the trade or profession ... We hold these principles apply with equal force to those who contract to develop computer programming.

Under the rule established in this case, liability for negligence will arise if a software developer (1) represents that it has expertise and training necessary to design and develop a system, (2) knows it lacks the requisite skills and expertise, (3) should have known that the customer was dependent on the developer's knowledge and abilities, and (4) should have foreseen that the customer would incur losses if the developer did not perform as agreed.

In the Illinois case [24], the software developer escaped liability based on legal technicalities. First, the court found that developing custom software legally constituted a "service" rather than sales of "goods." Second, the court refused to allow a malpractice claim, since under Illinois state law, malpractice suits are barred when "the consequences of the malpractice will be exhibited in the product." A different outcome is possible in the case of pre-packaged, mass-marketed "shrink-wrapped" software, which is legally considered a "product" in most states.

Even with regard to software engineers, not all courts have accepted the theory of "computer malpractice." In a 1984 Texas case involving a computer programmer [19], the Court refused to apply a malpractice standard, stating:

... Plaintiff seeks to generalize from the elements of the recognized forms of professional malpractice, specifically, a professional's representation that (1) they possess the requisite degree of learning and the skill necessary to the practice of their profession and which others similarly situated ordinarily possess, (2) they will exert their best judgment in the handling of matters entrusted to them, and (3) that they will exercise reasonable and ordinary care and diligence in handling matters entrusted to them ... Plaintiff's legal theory is creative, but without basis.

The court's reluctance to accept the plaintiff's theory is founded in traditional judicial conservatism. All "new torts" are considered by the courts on a case-by-case basis, and often many years elapse before a new theory is generally accepted. In the case of software malpractice, not all courts have been willing to embrace the theory, perhaps because of the few existing cases provide little guidance. But the existence of two cases speaking favorably on the concept provides a basis for other courts to change the course of the law. Since United States law is a derivation of English common law, these principles may also eventually apply to software sold in the U.K. [15].

2.3: Breach of duty

The second element of a negligence lawsuit is to establish that a person or institution acted or failed to prevent harm to another in accordance with his duty. Once a level of care is established, it is usually simple to prove non-conformance with that duty.

The doctrine of "respondeat superior", which literally means "the higher one must respond," renders an employer liable for negligence committed by an employee within the scope of his employment. The employee is rarely sued personally because of limited financial resources with which to satisfy a judgment. In the introductory hypothetical, high personal income and personal responsibility made Bill Applegates an attractive candidate for a lawsuit. Most software engineers are not so lucky, and will be shielded from personal liability by respondeat superior.

The respondeat superior doctrine suggests that management must vigilantly monitor the performance and work product of programmers and software engineers. If bugs are introduced into programs through lack of supervision or review, management
personnel will be asked to answer to corporate executives or shareholders.

When an employee is not originally named in the lawsuit, sometimes an employer will cross-sue its own employee to seek indemnification for the employee's negligence [28]. In situations where the employee's error is extreme, or where it violated company policy, procedures, or instructions, a court may hold the employee personally responsible.

In some situations, the courts allow a plaintiff a shortcut in proving breach, under the doctrine of "res ipsa loquitur." Under this doctrine, which literally means "the act speaks for itself," there are some cases where the fact that a particular harm occurred establishes a presumption that the duty of due care has been violated [28]. Such cases involve injuries which usually do not occur in the absence of negligence. The accident itself establishes negligence. The doctrine has been applied to aircraft and rail accidents.

Given the complex nature of software, one cannot say that errors usually do not occur in the absence of negligence. On the other hand, if programming for particular problems becomes so commonplace and straightforward that errors usually arise only in the case of negligence, then the doctrine might apply. Consider a simple task such as writing a keyboard input routine: such a routine might be considered so common that an error in the routine, resulting in harm, would automatically result in liability under the doctrine of "res ipsa loquitur."

2.4: Causation

In negligence cases, causation is analyzed in two steps: actual cause and proximate cause.

A software engineer is not the actual cause of an injury unless the injury would not have happened but for an error in the software written by the software engineer. This is the "but for" standard of actual causation [26]. The software engineer is not liable, even if negligence is proven, if the actual cause of harm is erroneous input ("garbage in, garbage out"), system operator error, or a hardware failure [28]. Note, however, that the "but for" test can be satisfied if a court determines that the software engineer should have recognized the likelihood of the actual cause and taken steps to protect against it. In such a case, the software engineer can viewed as having the "last clear chance" to avoid harm. If so, a court can rule that the software engineer is the cause in fact of the harm.

Proximate cause is related to the question of foreseeability. Proof of negligence requires that the resulting harm was reasonably foreseeable at the time of the negligent conduct. For example, a computer negligence case may fail because it was not foreseeable to the software engineer that third persons would rely on erroneous output. This rule was applied by the New York Supreme Court to prevent recovery by unforeseen third parties who relied on an accountants' negligently prepared opinion [29]. The scope of foreseeable is an important limit on malpractice litigation. The limits in the context of software engineering malpractice are not yet known, but would at least include all anticipated end users.

Note that two concepts of foreseeability are used in negligence cases: foreseeability of error (actual causation) and foreseeability of harm (proximate causation).

2.5: Damages

Harm or damages is a required element in negligence and malpractice cases. In computer negligence cases reported to date, the amount of damages was based on a written contract. For example, if a software developer delivers a non-functioning system after receiving payment, the customer may be entitled to win back the entire purchase price of the system plus an additional amount for lost profits or loss of use of the system. One limiting factor is that damages must be provable to be recoverable. Many cases founder because the damage claimed is merely speculative or relies on no hard evidence or documents.

But the primary impact of software engineering malpractice is that it will allow recovery of punitive damages and non-economically based damages for personal injury. These types of awards are based on punishing the defendant, and compensating the plaintiff for pain and suffering. Consequently, they are highly speculative and subjective. In medical malpractice cases, they often result in damages in the millions of dollars. Awards of these damages represent a true threat to the software engineering industry.

Legislation can establish statutory damages for computer-related negligence. A recent proposal suggested fines on common carriers for $10,000 per minute of disrupted telephone service, including outages caused by software faults [38].

3: Software Engineering Liability in Other Settings

Since software engineering malpractice is a relatively new legal theory, its proponents may attempt to allege it as an alternate theory, in addition to the
theories of liability discussed below. In addition, the recognition of software engineering malpractice by the courts may cause injured parties to seek redress under a malpractice theory rather than the theories below.

Liability for breach of contract can arise when a custom software developer fails to deliver software on time. Contract liability can also arise when the software does not meet the needs of the user, does not meet the specifications set forth by the user, functions too slowly, takes too much memory, or lacks proper documentation. Proper drafting of computer procurement contracts can limit a software engineer's liability for these types of faults [7]. If no contract limitations are used, failure of a software system to conform to requirements or specifications in the contract is legally considered non-performance, and can lead to recovery by customer of the purchase price under the contract [11].

Unfortunately, a negligence or malpractice theory may enable asserting errors or omissions which are not covered by the contract or the customer's specifications [30]. For example, if a software engineer produces a program on time which meets specifications, but simply operates too slowly, the customer may allege that the engineer prepared the program negligently. In several cases, tort theories have been used as an alternative to contract theories [13, 14, 22, 34].

A software customer can allege that the software engineer breached an express warranty (one written into the procurement contract) or an implied warranty (one implied by the circumstances or created by law). Claims of express warranties can also arise when a software developer promotes or delivers a software product with an oral or written guarantee of accreditation or reliability. Breach of implied warranty claims can arise from failure of software to comply with advertised standards or verification procedures in a written contract. When a contract states such standards and procedures, it can be argued to contain an implied guarantee that the software will meet the standards and procedures.

When the contract does not contain such standards, an injured plaintiff may bring a negligence case based on established accreditation standards. Or a plaintiff could claim that a software developer familiar with the standards made an implied warranty that the software would conform to the standards.

Civil fraud can arise when a software developer knowingly misrepresents a material fact relating to software, and induces reasonable reliance on the misrepresentation by the customer. In most states, a customer must prove that the developer had actual intent to defraud, which is difficult to prove unless a "smoking gun" document exists.

Civil fraud often arises from "the demo" -- when a software developer demonstrates an incomplete software system and misrepresents its capabilities [8]. If the system later fails to meet its promoted or agreed-upon specifications, the sale can be considered induced by fraud. Fraud has been found in situations involving "mere puffery" or promotional statements by sales staff. For example, one Federal court found that a statement indicating that software could be modified "quickly and cheaply" to meet a customer's requirements was found to be a misrepresentation which could support a lawsuit based on fraud [9].

In a more extreme case, fraud will be found when a software developer internally recognizes a defect in software, such as slow operation, and then designs a demo system which covers up the defect [13].

If a plaintiff is faced with fraud, but cannot prove intent or the exact contents of a misrepresentation, the plaintiff may turn to malpractice as an alternative remedy, on the theory that the failure of the program to meet the customer's expectations arises from the software engineer's failure to comply with a professional standard of conduct.

Strict products liability can also arise in cases of personal injury. Providers of professional services have liability based upon fault concepts such as malpractice. Sellers of products have no-fault liability flowing from a bad result caused by a defective product. First applied to automobile defects [32], this branch of the law has adapted itself to everything from defective chicken soup to defective aircraft. Software could be next. If strict liability is applied to software, and the jury decides that a fault is a "dangerous defect," then the original software vendor and any resellers will be liable regardless of how the defect was created. Software which runs but produces faulty data or faulty analysis would be subject to products liability.

The first requirement for strict products liability is sale of a product. There is no agreement that software is legally a product [33], even pre-packaged mass-produced software. In addition, such software is often sold with aggressive "shrink-wrap licenses" which disclaim all liability for anything except defects in media. As a result, purchasers of defective software who are unable to use the strict liability theory, available for most defective products, may choose to sue under a malpractice theory.

4: Preventative Measures

Software engineering malpractice arises from the failure of a software engineer to conform to a meet a
duty of reasonable care. It is axiomatic that to avoid malpractice, software engineers must conform to that duty. The following practices may help ensure conformance to the duty imposed by the law.

4.1: Know all current professional standards

Software engineers cannot conform to a professional standard unless they know what the standards are. The law will impose malpractice liability based on a breach of a professional standard regardless of whether the professional actually knows the standard; for professionals, ignorance is no excuse. Therefore, it is imperative for software engineers to participate in current standards-drafting activities, and to know the standards agreed upon by the profession.

Standards are also legally created in case law and through professional licenses [37]. Following case reports and state licensing efforts is important.

4.2: Use formal reliability engineering procedures

The law of professional malpractice presumes that professionals will comply with all commonly accepted practices of the profession. In software engineering malpractice cases, the courts are likely to question whether a particular software engineer has conformed to commonly accepted software engineering practices, including practices not formally adopted as standards by professional societies. This question would likely be resolved by expert testimony.

Consequently, commonly accepted techniques for reliability checks, formal verification, etc. must be built into the software development process as a discrete milestone. Once formal verification procedures are in place, test reports and quality reports must be used to provide feedback and correction of software. Legal liability will not be avoided by instituting procedures which are ignored.

Ordinarily, professionals cannot escape liability by claiming that they relied on someone else for information [36]. Although use of a "second set of eyes" will not insulate the software developer from liability, it creates an opportunity for the developer to obtain indemnification if an outsider performs formal analysis of code or specifications. Therefore, it is advisable for developers to use an analyst from outside the company to accomplish verification.

Group development of software may help insulate individual programmers from liability. For example, if a software product is developed by a team including designers, test specification writers, testers, and requirements analyzers, then it may be impossible for a court or jury to ascribe liability to a single individual programmer.

Reliance on faulty CASE tools could also lead to liability for the programmer and the CASE tool publisher. Output from CASE tools and program generators should be subjected to verification. Software developers should ask whether CASE tool vendors subjected the tools to formal verification. In mission-critical applications, it would not be unreasonable to require formal verification of the tools. Indeed, standards should be set as to the conditions under which any commercial software package will be used in program development.

At a minimum, the software engineer who uses CASE tools or other programs for analysis should be able to show that (1) the engineer understands the design theory of the program; (2) the program has been tested for reliability; (3) the data input is consistent with input into other modules responsible for computer analysis of other parts of the project; and (4) the results of the computerized analysis have been reviewed by someone familiar with the type of project [35].

To ensure that all foreseeable harm is trapped, an analyst or software engineer should always evaluate the software's interaction with the external environment and any other foreseeable external influence. For example, when programs interact with databases, the databases may represent a separate area of exposure to liability. If the database is to be filled with data by the customer after delivery, then liability for errors in its content should be disclaimed in a contract. To evaluate the risk of liability to third parties, the developer should ask to evaluate sample data in the database. If the database is not robust, so that the developer could be liable for errors produced in reliance on the database, the developer should consider whether to demand that the developer correct the database or supply its own.

If a program has an EDI or telecom capability, in which data will be acquired in real time from an uncontrolled outside source, including instruments, weapons, or other systems, then the developer must provide for distribution of liability for imported data.

After formal verification and human review are complete, the reliability engineering process should include preparation of a formal, written analysis document. Program analysis and preparation of analysis documentation is notoriously difficult [4]. The existence of such a document is legally important and can help negate any inference that the software
Reliability engineering result data must be embodied in a form understandable to the layman. Reliance only on data flow analysis or thick stacks of incomprehensible charts is not advisable because a judge or jury won't understand them when it comes time to explain what a software developer did.

4.3: Write liability limits into contracts

The discussion above noted the possibility of a cross-claim for indemnification by a company against its employee. If the employee-programmer is liable, the amount of liability can be high. Thus, the software engineer may wish to bargain with his employer at the outset of employment or assignment to a specific task for a limitation on his obligation to indemnify his employer for damages resulting from errors in the programs which he writes [28].

In most states, the programmer cannot completely insulate himself from liability to foreseeable third parties injured as a proximate result of the programmer's negligence. The only way to limit third party liability is to try to meet the duty of care of a professional, by constantly applying preventative measures like those discussed here, and by verifying compliance with those measures.

Whenever software is prepared on a custom basis, the procurement contract provides an important opportunity to limit liability as well as the amount and nature of recoverable assets. Such contracts are complex, and certain types of limitations are legally unenforceable. As a result, consulting a computer contracts treatise [7] or attorney is advisable.

Limitations on liability can be accomplished using waiver clauses and disclaimers which exclude certain types of damage from the contract. The exclusions are limited by state laws making some disclaimers unenforceable. For example, in some jurisdictions contract clauses cannot be used to escape liability for fraud [14].

The contract can limit recovery by imposing a narrow definition of actionable "program error" or "program defect," or by excluding liability for such errors or defects. One large computer company defines "error" and "defect" as "deviations between software and the documentation furnished by the developer to the customer" [7]. This provides an opportunity to avoid liability by changing documentation when an error is discovered during pre-delivery testing, and provides early limits on liability for fault and any required redesign and testing. But in some foreign countries even these limitations are unenforceable if software development was not performed in accordance with industry standards, known as "the rules of the art" [18].

Unfortunately, most liability limitations in contracts apply only to the parties to the contract. In general, contract limitations cannot be used to prevent tort liability (including malpractice) to "foreseeable third parties." The quoted phrase refers to every person that the contracting parties are likely to encounter as a result of the subject matter of the contract.

For example, in the hypothetical situation used in the introduction to this paper, a court would be likely to find that a hypothetical software engineer could reasonably foresee that aircraft passengers would be harmed by a defect in the aircraft control software. Therefore, the court would not enforce any contract provision designed to avoid liability of Bill Applegate to such passengers. What about persons living on the ground in the path of the stricken plane? Again, nearly all courts would find injury to such persons foreseeable.

Suppose further that the Aircorp plane crashes into a river, spilling fuel into the water and contaminating it. A child swimming downstream is injured by burning fuel. Most courts would find that the injury to the child is too remotely linked to the software error, so that it is not foreseeable to the software engineer. Therefore, Bill Applegate and Microfirm would not be liable for injury to the child.

Independent contractors or consultants are dangerous sources of liability, because they are not directly controlled by the software developer. Contracts with such consultants should place the risk of liability to the consultant, and should require the consultant to warrant the correctness of software and compliance with all the other measures discussed in this section. Unfortunately, the effectiveness of such contracts is limited by the financial resources of the consultant.
4.4: Establish internal control of personnel

Hiring interviews and first-day orientations provide an opportunity to advise employees of their professional obligations to customers and third parties. New software engineers must be reminded of the potential for liability and encouraged to stay abreast of developments in the field and take all possible measures to ensure that programs are correct and operate properly.

Corporate software developers should establish programs to ensure that software engineering staff continue to be professionally competent and aware of industry standards. New engineers, no matter how intelligent and credentialed, must be monitored to ensure that work product conforms to professional standards. In addition, hiring personnel should confirm that new hires have acceptable education and prior work experience.

4.5: Consider a legal audit

To evaluate risks of current software, current development programs, and the insulation provided by form or custom contracts, it may be advisable to ask an attorney to review company policies and procedures. Alternatively, many companies can conduct a comprehensive in-house review of legal exposure. Often this process is called a "legal audit." The software developer should not expect total insulation: in most states, you cannot contract away personal tort liability or product liability. In some states, remedies can be limited.

The legal audit should consider liability in every jurisdiction in which the software is sold. Different laws apply in different states, and the law of software liability is also well developed in foreign countries such as the U.K. [15] and France [16].

5: Conclusions

This paper has reviewed the state of malpractice liability of software engineers and software developers for errors in software. Two court cases are of particular interest, including one in which a software developer was found liable for negligence in preparing software. The courts have not yet expressly decided a reported case based on a theory of software engineering malpractice, but there is case support for the theory. In the future, software engineering malpractice cases may arise in a variety of contexts, especially in custom software development. Preventative measures can help limit or reduce the impact of malpractice liability.

REFERENCES

[16] A. Bertrand, "Legal Aspects of Bad Quality in Software," Software Protection vol. 7, no. 6-7, p. 10 (Nov.-Dec. 1988) (see p. 12, which states that in France, "[w]ith regard to third parties, the [tort] liability of the developer will be unlimited").


