Legal Liability for Bad Software

Karen Mercedes Goertzel

Abstract. This article focuses on lawsuits as a recourse for purchasers of defective COTS software — particularly safety-critical COTS software and software-controlled systems, such as software used in commercial aircraft, motor vehicles, unmanned aerial vehicles, medical devices, physical security systems, automated teller machines, commercial robots and industrial control systems, a wide variety of COTS diagnostic and sensor systems, a nd the whole growing panoply of cyber-physical devices and systems that collectively comprise the "Internet of Things." [1]

Background: Why Sue Software Companies?

Most commercial software contains serious flaws and defects, some of which are exploitable as vulnerabilities. This has always been the case, because it has proven impossible to fully test software, and while flaws/defects can be substantially reduced by adopting software assurance processes, [2] software products are never entirely free of flaws, either overlooked during development or intentionally left uncorrected at time of shipment. Over the last four decades, advances there have been exponential advances in software technology associated with the ubiquity of computers (first mainframes, then personal computers, and now mobile devices) and computer networks, the digital automation of once-mechanical controls for physical systems and processes, growing concerns over Information Warfare, the rise of hacking, malware, and computer crime, the open source movement, cloud computing, and on the horizon, the Internet of Things and The Singularity. With each advance, software has increased in size, complexity, ubiquity, exposure, and criticality due to by now almost complete human dependence on software's correct, reliable operation.

1974	London Ambulance Service	Poorly designed computer-assisted dispatch software delayed multiple ambulance dispatches, resulting in up to 30 deaths.
1985-87	Therac-25 radiotherapy machine	Concurrent programming errors caused the machine to give patients massive radiation overdoses; in two cases, these were fatal.
1991	MIM-104 Patriot surface-to-air missile	A software miscalculation error prevented the missile from intercepting an Iraqi missile on its way to strike a military compound in Saudi Arabia; 28 American soldiers were killed.
1992	F-22 Raptor flight control system	A software error caused the $150,000,000 fighter jet to crash.
1994	Royal Air Force Chinook engine control system	A software failure was found to be the likely cause of the helicopter's crash, which killed 29 people.
1996	European Space Agency (ESA) Ariane 501	A buffer overflow caused a hardware exception, leading to the rocket's crash upon launch.
1999	National Space and Aeronatic Administration (NASA) Mars Polar Lander	The software's misinterpretation of sensor data led to the craft's crash into Mars's surface.
2003	General Electric power grid monitoring system	A race condition in the system's software left a local power outage undetected, which escalated into a 48-hour blackout that extended across eight U.S. states and a Canadian province.
2005	ESA CryoSat-1 rocket propulsion unit	The absence of a critical software command led to the satellite's crash upon launch.
2009	Toyota (multiple models) electronic throttle control systems	Software errors caused multiple incidents of sudden acceleration, resulting in crashes that killed 89 people.
2009	Toyota Prius anti-lock braking system	Software errors led to braking failures, resulting in at least three crashes and multiple injuries.

Table 1. Some noteworthy software disasters [2]

But software has fallen short of its ability to meet expectations. And some of its failures have resulted in catastrophes of a magnitude that for other sectors and industries—from meat packing to automobiles—have raised outrage so great that the government was compelled to establish national regulations (and in a few cases, international) with whole agencies to enforce them. [3]

Inexplicably, despite these costly and fatal software-related disasters, and despite the recent spate of software malfuction- and defect-related automobile recalls at Toyota, Jeep-Chrysler, Honda and Volvo, and despite the software-related fraud committed by Volkswagen, nothing has inspired a general or sustained outcry or driven the government to undertake regulation of the software industry, as it has when other industries have been involved in safety or security catastrophes.

Despite decades of improvement of software safety and quality, most commercial software is still shipped with serious flaws and defects, some of which are exploitable as vulnerabilities. This has, in fact, been going on so long, it has simply become expected and accepted by most software customers. This is in large part because the software industry continues to audaciously insist that errors, defects and vulnerabilities in software are not only unavoidable, but represent a perfectly acceptable standard for software. [3] Moreover, they argue that the alternative of adhering to strict software assurance standards would not only be so costly as to make them uncompetitive and drive them out of business (or at least reduce their profit margins), it would severely inhibit innovation and delay the release of new features, which would harm the consumer. Apparently, it is with the consumer's best interests in mind that the software industry persists in producing poor quality, vulnerable products.

The truism that it is impossible to test software fully is widely accepted. This has been interpreted by many in the industry as free licence to adopt ultra-rapid agile methods and DevOps practices that enable developers to produce new software releases at an amazing rate but allow little time to fix coding errors or patch critical vulnerabilities that are found during testing and allow even less time to rethink designs that contain larger defects or requirements that are deficient. [4] Issues that cannot be mitigated by simple bug fixes, and which can only be discovered through thorough reviews and testing — or revealed when the software fails or is successfully hacked after deployment — are unlikely to ever be addressed. [5]

These problems have been reduced somewhat in larger software firms, such as Microsoft [6], that have the budgets to implement software assurance programs that help their developers reduce the overall number of flaws, vulnerabilities and even design defects in their software. But most software vendors — despite the publication of numerous "quality" and "secure" software methodologies over the past two decades — still do little to improve the quality or security of their processes. As a result, they ship products that contain easily avoidable flaws that their developers actually discovered during testing but chose not to correct because doing so would have delayed the release by a few days or weeks. (Even Microsoft, under pressure to adopt rapid agile development practices, has reduced the robustness of their security development lifecycle for products they develop under their agile regime.) The almost universal philosophy in the commercial software industry is "ship now, patch later."

Liability	The state of being bound or obliged in law or justice to do, pay, or make good something; legal responsibility;
Privity	The existence of a direct relationship between the parties of a contract;
Theory	(as in Theory of Case) Facts upon which a lawsuit will be founded, and which form the basis of a right to sue;
Tort	A legal injury or wrong committed upon a person or property independent of contract.

Table 2. A few definitions [7]

Outside of specialist firms that focus on developing real-time embedded software for safety-critical or cryptographic systems, the software industry has never shown interest in self-regulation and has loudly resisted external regulation of its products' quality, safety or security. Governments have also been unwilling to regulate the software industry out of fear of stifling industry innovation and the significant economic growth the industry brings in countries where it is active. Even the European Commission, which is far more active in regulating standards for European Union industries than most other government bodies, has been reluctant to regulate the European software industry.

In the absence of regulation, any customer who falls victim to the catastrophic results of software faults and failures has only one recourse: the courts.

Software and the Courts

The ambiguities surrounding software liability have been widely discussed in academic law literature (see Bibliography). These questions include whether software is a good or a service under the law, how to deal with a product as intangible and highly technical as software, and how binding End User License Agreements (EULAs) are under contract law, particularly in cases alleging gross negligence by the vendor or significant injury to the customer. [8]

The laws governing tort lawsuits have not yet been adapted to a world in which software is so universally present and prevalent. The literature on software liability (as typified by the footnotes throughout this article) reveals that the majority of software liability legal precedents were set in the 1980s. Little has changed since then in how contract and tort liability law are interpreted in reference to software. Lawyers who litigate such suits in the 2010s must spend a great deal of time and intellectual energy interpreting laws designed to govern commerce in physical products or professional services such as health care or architecture to figure out whether and how they apply to software.

Suing for Breach of Warranty Under Contract Law

In many cases, recovery of damages resulting from software failures have relied on contract law, tort law or both. When software defects result in only economic losses (with a few extreme exceptions discussed in the section below on economic loss and tort liability), plaintiffs seeking restitution rely on the contract theory of "breach of warranty." Such suits are possible when software vendors include in their licence agreements express warranties about their product's performance, functional capabilities or attributes such as security and quality, or when warranties are implied by legal requirements imposed on all products, such as merchantability and fitness for a particular purpose. A software defect often represents a deviation from the software's express or implied warranties.

To successfully sue under breach of warranty, a plaintiff must satisfy the requirements of privity, which means the software contract (license) must have been made directly between the buyer and the software vendor. Privity does not exist for software that is licensed to a reseller, other equipment manufacturer (OEM) or integrator, then resold to an end user. [9]

In addition to needing privity, an admissible breach of warranty suit must have a software licence that:

1. Includes express warranties about the product or its performance, or includes implied warranties.
2. Excludes enforceable limitation of liability clauses that preclude the purchaser from recovering more than the original purchase price of the software licence.

Despite a growing body of breach of warranty case law for software products, it remains unclear whether the Uniform Commercial Code (UCC) — which defines the widely accepted interpretation of civil contract law in the USA — even applies to software licenses because they do not transfer ownership of the actual software product from vendor to purchaser, but only transfer the right to use the vendor-owned product. If this is the case, software contracts are not product sales contracts but contracts for a service. In this case, it is a loan service (loan of use of the software). By extension, the software is not a product but a part of service delivery. This distinction between good and service is pertinent to determining not only how breach of warranty liability can be applied under contract theory, but also whether the theory of strict products liability can be applied to software as a product.

COTS software and software embedded in COTS products are frequently accepted as goods because they are mass-produced and transferred to unknown buyers who must rely upon the representations of the software vendor — two key prerequisites of being considered a product under the UCC. It doesn't matter whether the software is "bundled" with services such as the vendor's standard technical support package or user training; the software itself remains a product. By contrast, when software is developed under contract for a specific customer(s), or bundled into a much larger set of third-party services like system integration services, the ability to label the delivered software as a good or product rather than a service is cast into doubt.

Civil Liability for Fraud

It may be possible to sue a software vendor for fraud if the plaintiff can prove that he or she accepted the defective software product only as a result of the vendor's willful misrepresentation of that product's performance. Courts have favourably ruled against software vendors for fraudulent misrepresentation when those vendors have misrepresented facts that are known exclusively to the vendor, such as undisclosed vulnerabilities, defects, malicious logic or failure to test the software. [10]

Tort Products Liability

Because warranty law limits damages to recovery of product cost, plaintiffs often turn to tort theories of product

liability because contractual limitations do not normally apply to them and they have no requirements for privity or foreseeability. Tort liability arises when:

1. A defect in a product results in a failure of that product to operate safely as reasonably expected. For software, it doesn't matter whether the failure was accidental or caused by a Denial of Service attack exploiting the defect; and
2. Personal injury, death or property damage results.

According to the American Law Institute's *Restatement (Third) of Intentional Torts: Products Liability*, "A product is defective in design when the foreseeable risks of harm posed by the product could have been reduced or avoided by the adoption of a reasonable alternative design … and the omission of the alternative design renders the product not reasonably safe." [11] Three tort theories of product liability are potentially applicable to COTS software: negligence, malpractice and strict products liability.

Under all three tort theories, the plaintiff can recover damages associated with the following:

—loss of valuable data. Data can be valuable due to security classification or regulated privacy;

—destruction of raw materials;

—destruction or loss of property other than the product itself.

Under a strictly limited set of extreme conditions, plaintiffs may also recover damages due to destruction of the product itself.

Another area of tort liability, intentional liability, is almost exclusively limited to acts of battery (as in "assault and battery"). However, some have questioned whether intentional liability might be applicable in cases when a product contains a defect, backdoor or malicious logic introduced through malicious developer or supply chain sabotage. Stay tuned for the first lawsuit to explore this possibility.

Tort Theory of Negligence

General tort theories of negligence apply to developers, integrators, testers, maintainers, technical support personnel, trainers and anyone else that is under contract to deliver the services of manufacturing, maintaining or supporting the software product or another service in which delivery of the software product plays a part. Tort negligence permits permit recovery if a software defect can be proven to result from the service provider's failure to apply due care when designing or implementing ("manufacturing") the software.

Under negligence, the service provider cannot be held responsible for every software product or service defect that causes loss to the customer or a third party; responsibility is limited to only harmful defects that could have been detected and corrected through "reasonable" software practices.

A buyer of COTS software products or software-controlled devices or systems will find it very difficult to prove vendor negligence because the buyer lacks visibility into the "black box" software and its development process. The complex and mysterious nature of software, which is not well understood by anyone but its developers, forces buyers to trust the vendors' representation of that software. As a result, anyone suing for negligence must be prepared to incur significant expenses associated with hiring technology-savvy lawyers, researchers and expert witnesses. [12]

Tort Theory of Malpractice

Tort theories of malpractice are a more rigorous, specialized form of tort negligence. They apply only to defendants in recognized, licensed professions, such as medicine, architecture and engineering. To sue under the tort theory of malpractice, the plaintiff must prove that the software's developer belongs to a recognized (ideally government-licensed) profession like software engineering and has failed to comply with the standards of that profession while engineering the defective software product.

A recognized system of professional licensure for software engineers has not yet been established nationwide, despite limited attempts at software engineering professionalization. For this reason, establishing tort liability of a software contractor — or a COTS software vendor — under tort theory of malpractice remains impractical. To date, no successful software malpractice suit has been undertaken in the U.S., though a few jurisdictions may have paved the way for successful litigation of computer malpractice claims. At least one court, the U.S. Court of Appeals for the 8th Circuit, agreed to hear an explicit computer malpractice case, Diversified Graphics, Ltd. v. Groves, 868 F.2d 293, in 1989. [13] By contrast, in Columbus McKinnon Corp. v. China Semiconductor Co., Ltd., 867 F. Supp. 1173, 1182-83 (1994), the Western District Court of New York refused to recognize software programmers as professionals and rejected the malpractice suit. [14]

The ability to sue software engineers for malpractice is one of several arguments put forth in favor of establishing a broadly recognized professional licensing scheme for software engineers, and such licenses are already being issued in some U.S. states and Canadian provinces [15]. Moreover, a growing number of independent software developers and software engineering firms now invest in software malpractice insurance in anticipation of an increase in liability and malpractice lawsuits and the eventual professionalization of software engineering. This is particularly common among those who serve industries in which the profession of engineering is well understood, such as the industries that produce avionic and space systems, medical devices or industrial control systems for nuclear plants. [16]

Tort Theory of Strict Products Liability

In most states, recovery is possible under strict liability tort theories regardless of any proven responsibility on the vendor's part. Neither negligence nor malpractice needs to be proven; strict liability can apply even if the vendor exercised reasonable care to avoid a defect and followed professional conduct standards. Proof that the defect was present and caused the plaintiff injury or property damage or loss is the only consideration.

Strict products liability for software is based on several premises:

1. The software is defective.
2. The software is a product, not a service. This tends to limit strict liability claims to COTS software and software embedded in COTS products.
3. The plaintiff used the software in the intended manner and did not introduce the defect through that usage. If the product requires user modification to operate, the vendor cannot be held liable for any injuries arising from defects introduced by the user's modifications.

According to *Restatement (Third) of Torts*, for a product to be subject to strict liability, it must be unreasonably dangerous to the ultimate consumer. Defects (including flaws/errors) in software and software-controlled products may be deemed unreasonably dangerous when:

1. They were present in the sofware code at time of product sale.
2. The design is defective, so that while it performs exactly as specified, the software is essentially designed.
3. The customer was not adequately warned by the vendor of the presence of known hazards, or of the limitations and parameters under which the software must operate.
4. The software was defectively designed, which means that while it performs exactly as the vendor intended, the intended performance is not reasonably safe.

The *Third Restatement* also suggests that if a manufacturer's knowledge of expected harms from defects can be proven, this will help support a claim of strict liability: "Because manufacturers invest in quality control at consciously chosen levels, their knowledge that a predictable number of flawed products will enter the marketplace entails an element of deliberation about the amount of injury that will result from their activity." [17] In other words, the sheer fact that manufacturers have quality control processes are admissions that they know their raw products are highly likely to be defective, and thus require careful scrutiny and correction before being released to the public.

Most courts accept that tort liability generally does not permit recovery of damages arising from the loss or destruction of the defective product itself. However, an exception can be made when defective software within a larger product is unreasonably dangerous and leads to the failure of the product, resulting in a calamitous event. This interpretation allows for recovery of product-destruction-related costs in most of the incidents in Table 1.

In such cases, the plaintiff doesn't need to prove that the defect and the associated product loss were clearly foreseeable due to the inherent nature of the product and its function, though one could clearly predict that defective flight control software could cause an airplane to crash or that failed controller software in an autonomous automobile could cause the vehicle to crash. In any case, such property losses would be recoverable under strict liability together with the personal injury damages recoverable under indisputable strict tort liability and the damage or destruction of whatever the airplane or automobile crashed into. [18]

Because strict liability and other tort lawsuits against software vendors are so expensive for plaintiffs to pursue, they have only proven practical when a class action lawsuit can be initiated.

The Federal Trade Commission as Watchdog and Plaintiff

The Federal Trade Commission (FTC) has expanded its purview from suing high technology companies for unfair competition to suing for unfair or deceptive trade practices such as "software misrepresentation." Most FTC cases are settled out of court with results that are binding on the defendant. In February 2013, the FTC settled one of the first liability suits involving inadequate software security against HTC America, Inc., manufacturer of Android-based smartphones. The FTC's complaint alleged "HTC's failure to employ reasonable security in the customization of its mobile devices," and stated that "had HTC implemented an adequate security program, it likely would have prevented, or at least timely resolved, many of the serious security vulnerabilities it introduced through the process of customizing its mobile devices." The FTC found that HTC had failed to follow commonly accepted secure programming practices and Android's documented secure customization practices, and that HTC also failed to adequately test the security of their products, to remove debug code before shipping, and to establish a vulnerability reporting mechanism.

In the settlement, HTC agreed to a number of remediation measures specified by FTC, including the establishment of a full-scale software security assurance program that would be independently validated every two years. At the time, the HTC case was heralded by many IT lawyers as one of the "most significant decisions in cybersecurity law." [19] In February 2016, the FTC reached a similar settlement with almost identical terms with router manufacturer ASUSTeK Computer, Inc., arising from multiple ASUS customer complaints to the FTC about vulnerabilities in ASUS' AiCloud and AiDisk software and firmware. As with HTC, the FTC confronted ASUS for systemic deficiencies in its software development and system engineering processes. With these two settlements, the FTC has shown not only its willingness but also its competence in confronting and altering the bad practices of producers of vulnerable software.

Whither Regulation?

In the U.S., there has been a long-running debate about whether the government should take a more active regulatory role to require vendors to produce secure computer software. One of their biggest concerns is that regulating the software industry could stifle innovation or drive software companies to other countries with less restrictive laws, as has happened in other U.S. industries. Based on historical precedent, there is also a case to be made that, absent strict regulation of commerce, exemplary punitive damage awards by courts and the fear of lawsuits they engender serve the same purpose as government regulation— they deter manufacturer misconduct and incentivize quality improvements. [20]

Proponents of regulation argue that continued reliance on the civil courts to improve the behaviors and products of the software industry unfairly favors the industry over injured consumers and is therefore unjust. Clearly agreeing with this sentiment, in 2009, the European Commission designated consumer protection for software buyers as a priority area for possible EU legislative action. [21]

In the current corporations-write-the-laws climate that exists in the U.S. (and, to a lesser extent, the U.K.), it is common to cynically assume that any regulations that did make it into law would be more likely to protect the software industry than the consumer — the exact opposite result proponents of regulation are seeking.

REFERENCES

1. This article does not discuss issues of intellectual property law with regards to software, nor does it explore the tantalizing question of criminal liability when a software-controlled autonomous machine such as a self-driving automobile or a robot/embodied artificial intelligence operates in a way that breaks the law. See Miller, C. C., (2014, May 13). When Driverless Cars Break the Law. The New York Times. Retrieved from http://www.nytimes.com/2014/05/14/upshot/when-driverless-cars-break-the-law.html See also: Paul, (2013, October 11). When Autonomous Vehicles Crash, Is the Software Liable? The Security Ledger. Retrieved from http://securityledger.com/2013/10/when-autonomous-vehicles-crash-is-the-software-liable/
2. List of Software Bugs. Wikipedia.org, Retrieved from http://en.wikipedia.org/wiki/List_of_software_bugs; London Ambulance Service. Wikipedia.org. Retrieved from http://en.wikipedia.org/wiki/London_Ambulance_Service#Computerisation
3. In his testimony to the House of Lords in 2006, Microsoft's Jerry Fishenden stated that the company was "making our platform as secure as we possibly can within the complex nature of software" [italics added]. See: House of Lords (U.K.) Science and Technology Committee. (2007, July 24). 5th Report of Session 2006–07, Personal Internet Security, Volume I: Report, Chapter 4: "Appliances and applications." HL Paper 165–I. Retrieved from http://www.publications.parliament.uk/pa/ld200607/ldselect/ldsctech/165/165i.pdf
4. For examples, see: U.S. Department of Agriculture Food Safety and Inspection Service, "FSIS History" Retrieved from http://www.fsis.usda.gov/wps/portal/informational/aboutfsis/history; (2011, March 25). After the Triangle Fire: State and National Workplace Safety Reforms. [Blog post]. Political Correction blog. Retrieved from http://politicalcorrection.org/factcheck/201103250003; Constitutional Rights Foundation. (Fall 2008). Upton Sinclair's The Jungle: Muckraking the Meat-Packing Industry. . Bill of Rights in Action, Vol. 24 (No. 1). Retrieved from http://www.crf-usa.org/bill-of-rights-in-action/bria-24-1-b-upton-sinclairs-the-jungle-muckraking-the-meat-packing-industry.html; U.S. Coast Guard. International Ice Patrol. Retrieved from http://www.uscg.mil/history/articles/iip_history.asp; The British Dams Society. About Dams–Safety: The Dolgarrog Disaster, 1925. Retrieved from http://britishdams.org/about_dams/safety.htm; Jensen, C. (2015, November 26). 50 Years Ago, 'Unsafe at Any Speed' Shook the Auto World. The New York Times. Retrieved from http://www.nytimes.com/2015/11/27/automobiles/50-years-ago-unsafe-at-any-speed-shook-the-auto-world.html;Piper Alpha. Wikipedia.org. Retrieved from http://en.wikipedia.org/wiki/Piper_Alpha
5. With the adoption of agile development processes, software firms have come to rely increasingly on public beta and even alpha testing of their products. The jury is out, however, on whether crowdsourcing of testing has been spurred by a desire to improve the quality of software or merely to shift the vendor's cost of testing to the consumer.
6. Lohr, S. (2003, October 6). Product liability lawsuits are new threat to Microsoft. The New York Times. Retrieved from http://www.nytimes.com/2003/10/06/business/product-liability-lawsuits-are-new-threat-to-microsoft.html
7. Adapted from Black's Law Dictionary Free Online Legal Dictionary, 2nd Edition. Retrieved from http://thelawdictionary.org
8. The last of these questions has been clearly addressed in Kingsway Hall Hotel Ltd. v. Red Sky IT (Hounslow) Ltd. (2010). The Court ruled that Red Sky IT, a software vendor, could be held liable for damages beyond the value of the software it sold. This overturned contract law precedent, which had previously limited such damages to the purchase price of the software, The ruling also held that the exculpatory terms of Red Sky's software license agreement were unreasonable, and thus invalid. See: England and Wales High Court (Technology and Construction Court) Decisions, Neutral Citation Number: [2010] EWHC 965 (TCC), Case No: HT-08-111. Retrieved from http://www.bailii.org/ew/cases/EWHC/TCC/2010/965.html. Note that EULAs are no longer fully binding in Australia, either thanks to a 2011 revision of Australian Consumer Law that eliminated the right of developers and importers of business software to limit their own liability for consequential loss arising from faults in their software. See: Westmoreland, R. (2011, November 8). Unlimited Liability for Software Companies. HWL Ebsworth blog. Retrieved from http://www.hwlebsworth.com.au/latest-news-a-publications/publications/intellectual-property-and-trade-marks/item/275-unlimited-liability-for-software-companies.html. For French law, see: Rambaud, S. (2004, June 7). French Supreme Court strikes out limitation of liability provision from an IT contract where the software publisher has breached a material obligation. Bird & Bird blog. Retrieved from http://www.twobirds.com/en/news/articles/2007/frenchsupremecourt-strikes-out-limitation-liability-provision
9. Increasingly, when no privity of contract exists between the plaintiff and defendant, and the plaintiff was financially injured through use of defective software licensed through a third party, courts are allowing tort actions for economic loss. Otherwise, tort actions exclude economic loss except in the extreme conditions described later in this article. See Abdullah, F., Jusoff, K., Mohamed, H., & Setia, R. (2009, November). Strict versus Negligence Software Product Liability. Computer and Information Science, Vol. 2 (No. 4). Retrieved from http://pdfs.semanticscholar.org/502a/604e4ff0e3b3028ff1ee9d82f63235055134.pdf
10. Kaner, C. (1997). Software Liability. Self-published whitepaper. Retrieved from http://kaner.com/pdfs/theories.pdf
11. Simons, K. W. (2006). A Restatement (Third) of Intentional Torts? Arizona Law Review, Vol. 48. Retrieved from http://www.bu.edu/lawlibrary/facultypublications/PDFs/Simons/RestatementThird.pdf. Restatement (Third) of Torts: Products Liability (1998) is one of a series of tort law-clarifying restatements published by the American Law Institute, a group of America's leading legal scholars. The U.S. legal profession considers the Restatements authoritative in their interpretations of tort theories, including strict product liability. The Restatement scholars' omission of the reasonable expectations of consumers as an independent governing standard, or test, for whether a product is defective is significant; this "consumer expectations test," which was present in the Second Restatement, was missing from the more recent Third Restatement.
12. By contrast, customers who contract service providers to develop custom software – either from scratch, or through significant modification of existing software – are much closer to, and have more visibility into, the software and the processes used to produce it. For this and other reasons, strict products liability rarely applies to custom-developed software. Tort negligence and, in some cases, tort malpractice may potentially apply, however. See McCullagh, D. (2003, August 26). A Legal Fix for Software Flaws. CNET News.com. Retrieved from http://news.com.com/2100-1002_3-5067873.html
13. Buchanan, I. & Rooney. (2001, June 1). PC, Computer Malpractice Actions: Are They the Wave of the Future?. Retrieved from http://www.bipc.com/Computer-Malpractice-Actions-Are-They-the-Wave-of-the-Future-06-01-2001/; Kaner, C. (1996). Computer Malpractice. Software QA Quarterly, Vol. 3 No. 4. Retrieved from http://kaner.com/pdfs/Malprac.pdf
14. Kornecki, & Cunningham, (2003). Software Safety–Ethics, Professionalism, and Legal Issues. Proceedings of the 21st International System Safety Conference. Retrieved from http://pages.erau.edu/~kornecka/papers/21SSC_ethics.pdf; Also Op. cit. Buchanan, Ingersoll & Rooney.
15. Kowalenko, K. (2012, February 3). Licensing Software Engineers Is in the Works. The Institute. Retrieved from http://theinstitute.ieee.org/career-and-education/career-guidance/licensing-software-engineers-is-in-the-works. Also, Peters, D. K. (2009, October 30). Licensure for Software Engineers. Class notes for Engineering 7893: Software Engineering (Memorial University of Newfoundland). Retrieved from http://www.engr.mun.ca/~dpeters/7893/Notes/presentations/SElicensing.pptx
16. O'Brien, H. M. (2015, February 2). The Internet of Things: The Inevitable Collision with Product Liability. Product Liability Advocate. Retrieved from http://www.productliabilityadvocate.com/2015/02/the-internet-of-things-the-inevitable-collision-with-product-liability/
17. Op. cit. Simons.
18. Brenneman, S. R. (1986). Computer Malfunctions–What Damages May Be Recovered in a Tort Product Liability Action. Santa Clara High Technology Law Journal, Vol. 2 Issue 2. Retrieved from http://digitalcommons.law.scu.edu/cgi/viewcontent.cgi?article=1025&context=chtlj
19. Rubens J. T. & Morse, E. A. (2013, November). Survey of the Law of Cyberspace: Introduction. The Business Lawyer, Vol. 69. Retrieved from http://apps.americanbar.org/dch/thedl.cfm?filename=/CL320000/sitesofinterest_files/Cyberspace_Law_Survey_2012.pdf
20. Vandall, F. J. (1981). Undermining Torts' Policies: Products Liability Legislation. The American University Law Review, Vol. 30. Retrieved from https://www.wcl.american.edu/journal/lawrev/30/vandall.pdf
21. Op. cit. Simons.

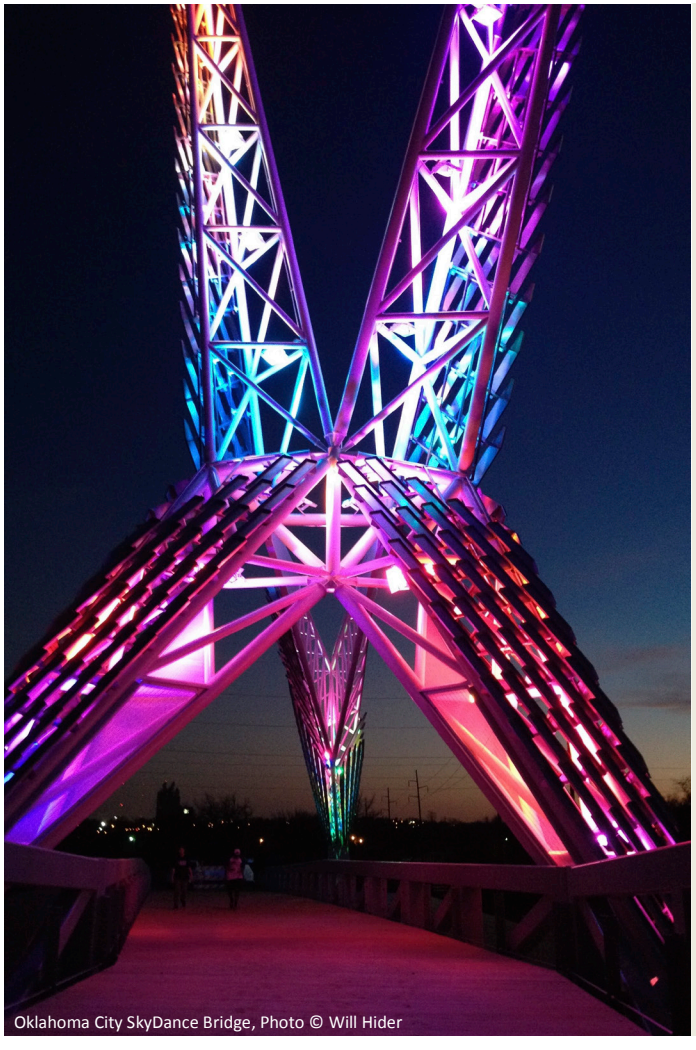
ADDITIONAL READING

Please contact the author to request a comprehensive list.

1. Bad Software Website. http://badsoftware.com/?page_id=62.
2. Armour, J. & Humphrey, W. S. (1993, August). Software Product Liability. Technical Report CMU/SEI-93-TR-13 ESC-TR-93-190. http://www.sei.cmu.edu/reports/93tr013.pdf3.
3. Beard, T. R., Ford, G. S., Koutsky, T. M. & Spiwak, L. J. (2010, July 7). Tort Liability for Software Developers: A Law and Economics Perspective. The Journal of Computer and Information Law, Vol. 27. http://www.phoenix-center.org/JCILTortLiability.pdf4.
4. Chong, J. Five part series in The New Republic.—(2013, October 3). Part 1, Bad Code: Should Software Makers Pay?. http://newrepublic.com/article/114973/bad-code-should-software-makers-pay-part-1—(2013, October 11). Part 2, Why Is Our Cybersecurity so Insecure?. http://newrepublic.com/article/115145/us-cybersecurity-why-software-so-insecure—(2013, October 22). Part 3, What You Don't Know About Internet Security Will Definitely Hurt You.. http://newrepublic.com/article/115281/what-you-dont-know-about-internet-security-will-definitely-hurt-you—(2013, October 30). Part 4, We Need Strict Laws if We Want More Secure Software. http://newrepublic.com/article/115402/sad-state-software-liability-law-bad-code-part-4—(2013, October 31). Part 5, The Security Burden Shouldn't Rest Solely on the Software User. http://newrepublic.com/article/115421/security-burden-shouldnt-rest-solely-software-user5.
5. Råman, J. (2006, May 26). Regulating Software Development. Juris Doctorate Dissertation, University of Lapland Faculty of Law. http://archive.nyu.edu/handle/2451/149936.
6. Scott, M. D. (2008). Tort Liability for Vendors of Insecure Software: Has the Time Finally Come? Maryland Law Review, Vol. 67 Issue 2. http://digitalcommons.law.umaryland.edu/mlr/vol67/iss2/5
7. Spruell, J. & Kamal, M. (2002). Defective Software and the Issues of Malpractice, Negligence, Fraud, and Misrepresetation. Proceedings of the International Association of Computer Investigative Specialists (IACIS) 2002 Conference (Fort Lauderdale, Florida). http://iacis.org/iis/2002/SpruellKamal.pdf8.
8. Turner, C. S. (1999). Software as Product: the Technical Challenges to Social Notions of Responsibility. Ph.D. Disertation, University of California, Irvine. http://users.csc.calpoly.edu/~csturner/fulltechreport.pdf9.
9. Vihul, L. (2014). The Liability of Software Manufacturers for Defective Products. The Tallinn Papers, Vol. 1 No. 2. http://ccdcoe.org/publications/TP_Vol1No2_Vihul.pdf

ABOUT THE AUTHOR

Karen Mercedes Goertzel is an internationally recognized cyber security, information assurance and software assurance expert with more than years of experience in research and analysis, technology strategy, solution specification and architecture, process definition and improvement, policy and guidance development, and technical communication. Her areas of subject matter expertise include system, software, and hardware assurance; application security (including security for web, mobile, Internet of Things, and cloud applications); information and communications technology; supply chain risk management; insider threat to information systems; and information protection, assured information sharing, and data loss prevention. Her passion is research, and she is the author or co-author of numerous published peer-reviewed articles and conference papers and of several book-length research studies.



Oklahoma City SkyDance Bridge, Photo © Will Hider