Hardware and Software Liability

by Tim Tompkins

submitted to: Prof. Wes Huang   submitted for: Computer Ethics   6 December 2000

Often a piece of hardware or software will come with a license agreement that states that the creator is not liable for any damages that may result from the use of their product. However, do the programmer, publisher, designer, etc. of a piece of hardware or software have the right to say that they are not at all responsible? If a company was to produce a common household item, and it was found to be of poor quality and unnecessarily endangered the user with normal use, they would be held responsible in a court of law. Should the same law apply to computer products?

The ramifications are even more staggering when one considers that hardware and software can have uses in medical fields, where human lives are at stake, and in financial markets where large sums of money are concerned, and in educational institutions, where performance is measured.

Can we allow these companies to sell their products, and then shirk all liability, often with a single mouse click of the end user, especially as computer hardware and software is being used to perform more and more functions that we depend upon every day?

Several types of legal liability exist. Liability is defined, in a legal sense, as "almost any obligation, responsibility or duty that might arise from a cause in a statute, contract or tort" (Cardinali 258).

"Ordinary negligence applies when a software developer does not use the degree of care that a reasonably prudent person would have used when developing software" (Wahl 175). In court cases where negligence is concerned, it is either proven or disproven that the developer of a product exercised their "duty of care" in the manufacture and sale of said product. If it can be determined that there is something a software developer should have done, and would reasonably have been expected by him by all others involved in the use and distribution of the software, then he can be found guilty of negligence, and required to pay damages to the plaintiff.

"Malpractice is a failure to employ the higher standard of care that a member of a profession should employ" (Wahl 175). It has not been determined whether malpractice can always be applied to computer cases. This is simply because it has not been unequivocally concluded if computer professionals are, in fact, professionals. "Certainly computer professionals posses some of the appropriate characteristics" (Johnson 42). However, some computer professionals possess little autonomy, and there is no legally recognized professional organization (Johnson 42). In some cases the courts have "declined to invent such a tort (i.e. wrong) on the basis that simply because an activity is more technically complex and important does not mean the greater potential liability must be attached" (Cardinali 258). However, there are examples of cases in which a verdict was issued in favor of the plaintiff suing under "computer malpractice," when it was determined that a consulting firm did not "act reasonably in light of its superior knowledge and expertise in the area of computer systems" (Cardinali 258). We see the beginning of a trend here: information and decisions regarding computer liability cases are confusing and often in conflict.

The third type of liability is strict liability. "Manufacturers and sellers of defective products are held strictly liable, (that is, liable without fault) in tort (that is, independent of duties imposed by contract) for physical harms to person or property caused by [a] defect" (Samuelson 25). Strict liability is usually only applied in extreme cases, where a product defect is obvious. In general, strict liability cannot be applied carelessly, since, "…holding some manufacturers liable in all cases will cause the consumer to behave ineffectively" (Cardinali 259). Since the problems with software are often not clear or entirely understood by members of the legal system, rarely is strict liability used in computer-related cases.

Software and hardware developers almost always have checks, tests, and quality assurance departments in place to make sure that their products are not defective, and perform as claimed. So how does it happen that defective software is released? There are a number of factors that can be shown to contribute to this increasingly relevant problem.

"There are extreme pressures on software developers to bring new products to the marketplace at an ever increasing pace" (Wahl 175). As today's society moves faster and faster and newer products and technologies are quickly becoming available, it is challenging for software companies to keep up, given budget constraints. Under pressure from project leaders and upper management, programmers have to churn out code that does the job in less time than ever before, which can mean working long hours without sleep, and possibly making careless mistakes.

Another problem plaguing the industry is the lack of standards. Many professions and products have standards associated with them, either legally imposed, or assumed. Computer hardware and software have, so far, not been subject to any imposed standard, only those voluntarily accepted by the developers. Usually those standards that are adopted are for ease of use or compatibility, not for quality assurance or reliability. Imposing standards on the industry would undoubtedly be difficult, as there are such varied products and services offered. Also, it may have the undesirable effect of driving up what some consider to be prices that are already too high.

Increasingly, the world is becoming a global community and a global market. As hardware and software is developed, manufactured, bought, and sold internationally, ensuring compatibility, safety, and compliance with each country's or state's regulations (if any) becomes a formidable task.

The primary tool for assuring quality presently is testing. It is, however, impossible to test a piece of software for every possible use. As a simple example, imagine trying to test the input functionality of a program that accepts an integer value. Assume that this program may run on proprietary hardware, and not necessarily be running on top of an existing operating system. Can you be sure that if a user enters "1234" then backspaces over the "34" and types "26" and the backspaces three times and enters more numbers, etc., etc., that you've tested every possible keystroke combination? Of course not, and it is foolish to assume that every miniscule possibility can be thought of and tested. That would take, quite literally in some complex cases, and infinite amount of time. So testing can only be conducted up to a "reasonable" level. Defining reasonable can be quite a challenge, but more on that later.

All of the aforementioned problems contribute to a series of dire consequences for the companies that use, as well as the companies that produce, products that fail. Obvious consequences include the immediate loss of productivity and sales that can occur (Cardinali 257). When a system fails, many man-hours and profit dollars can be lost within a short period of time, usually the entire time the system is down. Such examples include revenue generating web sites and retail point-of-sale systems going down.

A second level of loss includes the cost of providing emergency service during the failure of the primary service. Should a product fail, a temporary fix often needs to be applied immediately to allow the customer to continue operation in a semi-normal manner, especially for mission-critical systems in hospital, military, and numerous other important applications. Also in this category is the cost of restoring data, should it be necessary.

The third level of loss is long-term. Should a company's systems fail, either due to their own hardware and software, or that which they have purchased or licensed from another company, customers may no longer be willing to trust that company with their business, so the company's long term reputation, and consequently profit, are diminished.

Numerous are the problems that exist regarding hardware and software liability. Let us see what can be done to address these issues, and perhaps provide some guidelines that can be applied by both hardware and software developers and users of said hardware and software, and could perhaps form the basis for legislation in this area, should it be deemed necessary and supported by the relevant groups of people and companies involved.

The best solution to dealing with unreliable software is, of course, simply to create only reliable software (Wahl 176). As we saw above in the discussion of testing, this is just not possible. Currently "…standards of care in software development do not exist" (Wahl 176). A reasonable standard of care must be established for each application. This is an extremely difficult chore, as it requires a good working knowledge of the type of system that the standards are being applied to, as well as a compromise between the developers and users of that system. It is not, however, unreasonable to define such standards to the extent that they are loosely defined for other products and services.

Today many license agreements for commercial software read as follows: This software is provided on an "As Is" basis without a warranty of any kind, including without limitation the warranties of merchantability, fitness for a particular purpose and non- infringement. The entire risk as to the quality and performance of the software is borne by you. Should the software prove defective, you and not the companies assume the entire cost of service and repair. Under no legal theory tort, or contract or otherwise will the company be liable to you or any other person for any indirect, special, incidental or consequential damages of any character including damages for loss of good will, work stoppage, computer failure or malfunction or any and all commercial damage loss. In no way will the company be liable for damages in excess of the list price for a license to the software even if the company will have been informed of such danger or for any claim by other parties (Cardinali 260).
License agreements such as this absolve the publisher of every possible legal responsibility. So far, "…this type of limitation has been upheld in court" (Wahl 176). This type of an agreement would be laughed at if it was put on an automobile or an electric drill, and yet, the software is, in many ways, no less dangerous than those items. Software is a tool, and people rely upon tools to do their work. There is no reason why software should not be subject more of the same laws that products are subject to. It should be understood that sometimes software will fail at a greater rate than more tangible products, and most people will accept that. It is simply that the current state of these agreements is too far over the line.

There are various suggestions that have been made to the industry that simply "make sense" and one would consider it reasonable to apply them, such as ceasing publication of software incompatible with newer operating systems/other software, and that fixing bugs should take priority over adding new features, and better customer support, since that can often prevent the losses that would be associated with a liability case (Turner).

Hardware and software liability is a new issue in society and law, and reactions to it are slow. It is unclear how to apply existing laws to these products and services, and what new laws need to be written. As these items become more ubiquitous in everyday life, our society will have to deal with this, and hopefully we will be prepared for it, rather than cobbling together some policies that do not effectively deal with the situation. Simply put, "…the goal of liability law is to induce proper behavior on the part of both the consumer and the producer" (Cardinali 259). "The real question on who should be held liable rests with 'Who has the power to prevent harm from occurring?'" (Cardinali 259). Those with that power must to be encouraged to exercise it.

Works Cited

Cardinali, Richard. "If the System Fails, Who Is Liable?" Logistics Information Management Volume 11, Number 4, 1998: 257–261.

Johnson, Deborah G. Computer Ethics, Second Edition. Upper Saddle River, NJ: Prentice Hall: 1994.

Samuelson, Pamela. "Liability for Defective Electronic Information." Communications of the ACM. January 1993: 21–26.

Turner, James. "Software Users of the World, Unite!" Christian Science Monitor. 25 February 1999: 16.

Wahl, Nancy J. "Responsibility For Unreliable Software." Ethics in Computer Age. Gatlinburg, TN: ACM, 1994: 175–177.