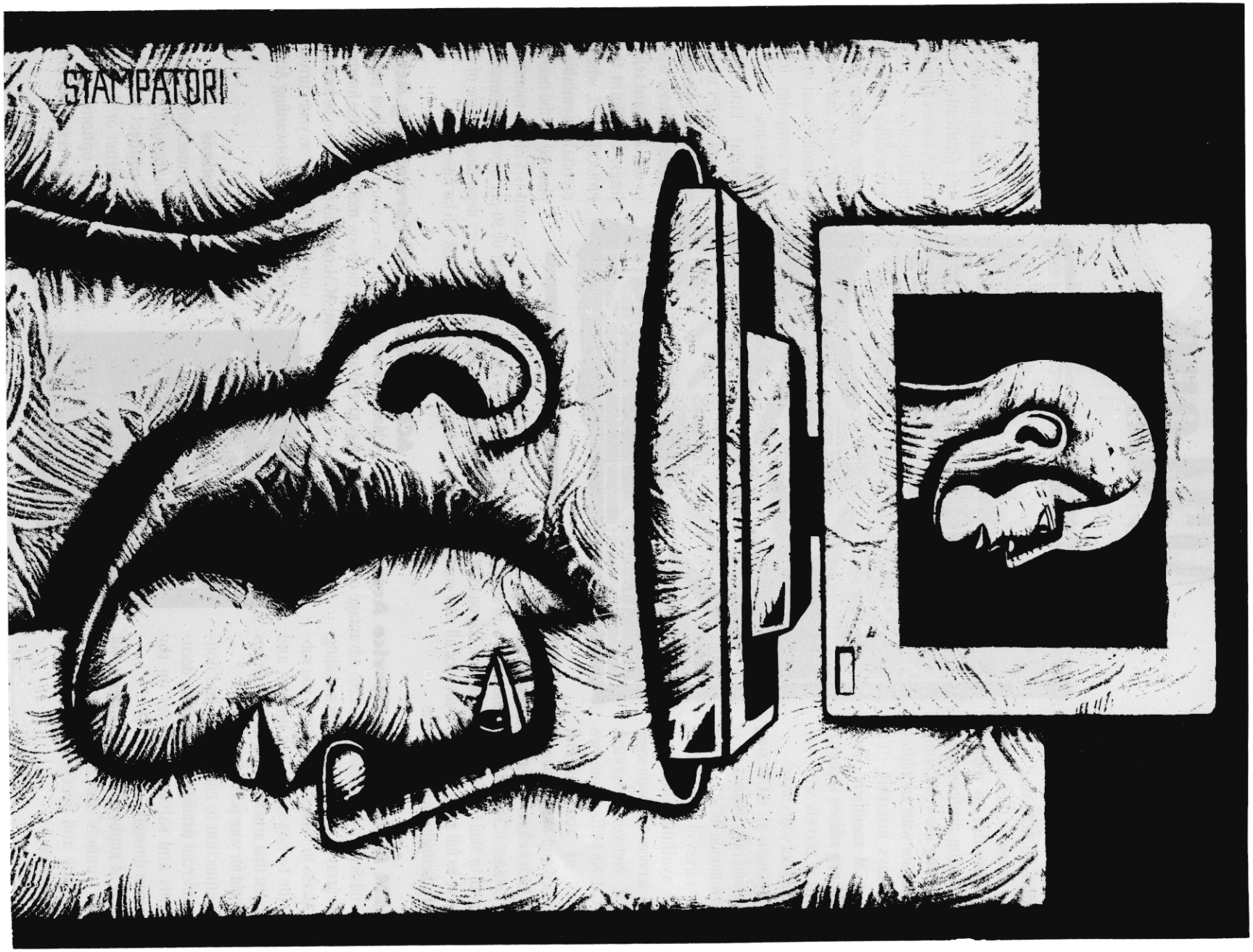
STAMPATORI

COMPUTING AND ACCOUNTABILITY

Helen Nissenbaum

A teacher stands before her sixth-grade class demanding to know who shot the spitball in her ear. She threatens punishment for the whole class if someone does not step forward. Eyes are cast downward and nervous giggles are suppressed as a boy in the back row slowly raises his hand.

The boy in the back row has answered for his actions. We do not know whether he shot at the teacher intentionally or merely missed his true target, whether he acted alone or under goading from classmates, or even whether the spitball was in protest for an unreasonable action taken by the teacher. While all of these factors are relevant to determining a just response to the boy's action, the boy, in accepting responsibility for his action, has fulfilled the valuable social obligation of accountability.

In an increasingly computerized society, where computing, and its broad application, brings dramatic changes to our way of life, and exposes us to harms and risks, accountability is very important. A community (a society or professional community) that insists on accountability, in which agents are expected to answer for their work, signals esteem for high-quality work, and encourages diligent, responsible practices. Furthermore, where lines of accountability are maintained, they provide the foundations for just punishment as well as compensation for victims. By contrast, the absence of accountability means that no one answers for harms and risks. Insofar as they are regretted, they are seen as unfortunate accidents—consequences of a brave new technology. As with accidents due to natural disasters such as hurricanes and earthquakes, we sympathize with the victims' losses, but do not demand accountability.

This article maintains that accountability is systematically undermined in our computerized society—which, given the value of accountability to society, is a disturbing loss. While this systematic erosion of accountability is not an inevitable consequence of computerization, it is the inevitable consequence of several factors working in unison—an overly narrow conceptual understanding of accountability, a set of assumptions about the capabilities and shortcomings of computer systems, and a willingness to accept that the producers of computer systems are not, in general, fully answerable for the impacts of their products. If not addressed, this erosion of accountability will mean that computers are "out of control" in an important and disturbing way. This article attempts to explain why there is a tendency toward diminished accountability for the impacts, harms, and risks of computing, and it offers recommendations for reversing it.

My concern over accountability has grown alongside the active discussion within and about the computer profession regarding the harms and risks to society posed by computers and computerized systems. These discussions appeal to computer professionals,[1] to the corporate producers of computer systems, and to government regulators, to pay more heed to system safety and reliability in order to reduce harms and risks [1, 12–14, 16, 18, 22, 28]. Lives and well-being are increasingly dependent on computerized systems. Greater numbers of life-critical systems such as aircarft, spacecraft, other transportation vehicles, medical treatment machines, military equipment, and communications systems are controlled by computers. Increasing numbers of "quality-of-life-critical" systems, from the vast information systems (IS) supporting infrastructures of govern-

ments, corporations, and high finance, to community networks [21] and workplace systems [2], down to personal conveniences such as telephones, microwaves and toys, are controlled by computers. Consequently, lives, well-being, and quality-of-life, are vulnerable to poor system design and failure.

While this vulnerability gives compelling grounds for directing greater attention to system safety, reliability, and sound design, and for the technical strategies for overcoming shortcomings, it also indicates the need for greater accountability for failures, safety, risk, and harm. Why? Because those who are answerable for harms or risks are the most driven to prevent them. In this way, accountability serves as a powerful tool for bringing about better practices, and consequently more reliable and trustworthy systems. Accountability means there will be someone, or several people, to answer not only for the malfunctions in life-critical systems that cause or risk grave injuries and cause infrastructure and large monetary losses, but even for the malfunctions that cause individual losses of time, convenience, and contentment. Yet because of barriers generated by the contexts in which computer systems are produced, and assumptions about computing and its limitations, accountability for the harms and risks mediated by computing is becoming elusive. How does this occur? To understand how the barriers to accountability arise, some clarification of key concepts is needed.

Accountability, Blame, and Responsibility

For centuries, philosophers and legal scholars have sought to understand accountability and the related concepts of responsibility, blame, and liability, through definitions, prototypical cases, and sets of conditions that would capture their meanings and provide clear grounds for legal principles.[2] Take the concept of responsibility: A common denominator in most analyses of responsibility are two conditions that determine whether someone is responsible for a harm: (1) a causal condition, and (2) a mental condition (that lawyers refer to as *mens rea*). According to the causal condition a person's actions (or omissions) must have *caused* the harm; according to the mental condition, the person must have *intended* (decided), or *willed* the harm.[3] For example, a person who intentionally installs the Explode virus on her employer's computer is responsible for extensive damage to files because her *intentional* actions were *causally* responsible for the damage.

The concepts of accountability, liability, and blame, extend somewhat farther than the scope suggested by the two preceding conditions. Take for example, the mental condition. Blame for harm is not limited to cases in which an individual willed, or intended it. Recall the case of Robert Morris's Internet Worm. Although the widespread harm it caused was a result of an unintended bug in Morris's code, few were willing to exonerate Robert Morris on the grounds that he had not directly intended the harm he, in fact, wrought. This case illustrates how the mental condition can be weakened to include even unintended harm, if the harm is brought about through negligence, carelessness, or recklessness. In general, if a person fails to take precautions of which he is capable, and that any reasonable person with normal capacities would have taken in those circumstances, then he is not excused from blame merely because he did not intend the outcome. We refer to this generalized version of the mental condition, which includes intent to harm, as well as negligence and recklessness, as "the fault condition" [5].

The causal condition, too, can be weakened to cover cases in which an agent's actions were not *the* cause, but merely one significant causal factor among a number of others. For example, we may blame a person whose actions, in conjunction with those of another, causes harm. We may even blame a person who causes injury while acting under someone else's orders. These variations on the two conditions, though truer to realistic notions of blame and responsibility, make drawing lines difficult. In an actual case, a judgment over whether an individual is blameworthy can depend on numerous factors particular to that case.

Responsibility and blameworthiness are only a part of what is covered when we apply the robust and intuitive notion of accountability—the notion exemplified by the boy in the back row "stepping forward." When we say someone is accountable for a harm, we may also mean that he or she is liable to punishment (e.g., must pay a fine, be censured by a professional organization, go to jail), or is liable to compensate a victim (usually by paying damages). In most actual cases these different strands of responsibility, censure, and compensation converge because those who are to blame for harms are usually those who must "pay" in some way or other for them. There are some important exceptions, including for example, the case of parents who must answer for injuries caused by their children's reckless behavior, or insurance companies who must cover damages caused by their clients. Strict liability is another. In its bearing on the goal of maintaining accountability in a computerized society, strict liability is of great importance.

To be strictly liable for a harm is to be liable to compensate for it even though one did not bring it about through faulty action. (In other words, one "pays for" the harm even though the fault condition is not satisfied.) This form of liability, which is found in the legal codes of most countries, typically applies to the producers of mass-produced consumer goods, to the producers of potentially harmful goods, and to owners of "ultrahazardous" property. For example, even if they have taken a normal degree of care, milk

[1]Here and elsewhere, I use the term "computer profession" very broadly to refer to the loose community of people who dedicate a significant proportion of their time and energy to building computer and computerized systems, and to those engaged in the science, engineering, design, and documentation of computing.

[2]For excellent, and more thorough, contemporary discussions, see for example [5, 7, 8].

[3]A precondition for blameworthiness, especially relevant to the legal domain, is that a person be in possession of certain mental capacities, including the capacities to distinguish right from wrong, and to control his or her actions. Since the issue of mental capacity has no bearing on computing and accountability, I will take it no further.

producers are strictly liable for illness caused by spoiled milk; owners of dangerous animals (for example, tigers in a circus) are strictly liable for injuries caused by escaped animals even if they have taken precautions to restrain them. Although critics of strict liability argue that it is unjust—because people are made to pay for harms that were not their fault—supporters respond that strict liability is nevertheless justified because it contributes significantly to the good of society. It serves a paramount public interest in protecting society from potentially harmful or hazardous goods and property, and provides incentive to sellers of consumer products and owners of potentially hazardous property to take *extraordinary* care. It assures compensation for victims by placing the risk on those best able to pay, and those best able to guard against the harm. And it reduces the cost of litigation by eliminating the onerous task of proving fault.

Four Barriers to Accountability

Accountability is obscured when we apply these conceptual understandings to the types of contexts in which computer systems are produced, combined with commonly held views about the nature of computing—both its capabilities and limitations. The barriers I will discuss are: 1) The problem of "many hands"—because computer systems are created predominantly in organizational settings, 2) Bugs—because bugs not only cause problems but commonly are conceived of as a fact of programming life, 3) The computer as scapegoat—because it can be convenient to blame a computer for harms or injuries, and 4) Ownership without liability—because in the clamor to assert rights of ownership over software, the responsibilities of ownership are neglected. These barriers to accountability can lead to harms and risks for which no one is answerable and about which nothing is done.

1. The problem of many hands. Most computer systems in use today are the products not of single programmers working in isolation, but of groups, collectives, or corporations. They are produced by teams of diverse individuals, that might include designers, engineers, programmers, writers, psychologists, graphic artists, managers, and salespeople. Consequently, when a system gives rise to harm, the task of assigning responsibility, the problem of identifying who is accountable, is exacerbated and obscured because responsibility, characteristically understood in terms of a single individual, does not easily generalize to collective action. In other words, while our conceptual understanding of accountability directs us in search of "the one" who must step forward (for example, the boy in the back row answering for the spitball), collective action presents a challenge.

Where a mishap is the work of "many hands," it can be difficult to identify who is accountable because the locus of decision making (the "mental condition") is frequently different from the mishap's most direct causal antecedent; that is, cause and intent do not converge. Take for example, a bad course of action taken by a political leader, which was based on the word of a trusted adviser. Although the action is taken by the leader, the adviser's word has been a decisive factor. How do we figure the adviser's role into the question of accountability? Further, with the collective actions characteristic of corporate and government hierarchies, decisions and causes themselves are fractured. Boards of directors, task forces, or committees, make decisions jointly, sometimes according to a majority vote. It is the collective efforts of a team that give rise to a product. When high-level decisions work their way down from boards of directors to managers, from managers to employees, ultimately translating into actions and consequences, it is difficult to trace precisely the source of a given problem. As a result, the connection between outcome and the one who is accountable is difficult to make. The problem of many hands, also known as the problem of collective responsibility, is not unique to computing but plagues other technologies, big business, government, and the military [4, 5, 11, 25, 27].

Computing is vulnerable to the obstacles of many hands because, first, as noted earlier, most software systems in use are produced in institutional settings, whether in small and middle-sized software development companies, or large corporations, government agencies and contractors, or educational institutions. (Some cynics argue that institutional structures are designed precisely to avoid accountability.) Second, computer systems themselves—usually not monolithic—are constructed out of segments, or modules. Some systems include code from earlier versions, while others borrow code from different systems entirely. When systems grow in this way, sometimes reaching huge and complex proportions, there may be no single individual who grasps the whole system, or keeps track of all the individuals who have contributed to its various components (See [10, 28].) Third, performance in a wide array of mundane and specialized computer-controlled machines—from rocket ships to refrigerators—depends on the symbiotic relationship of machine with computer system. When things go wrong, it can be difficult to discern whether the call goes to the manufacturers of the machine or to the producers of the computer software.

To see the problem of many hands in action, recall the case of the Therac-25, a striking example of the way many hands can obscure accountability, and at the same time a stark reminder of the practical importance of accountability. In a series of mishaps, now quite familiar to the computer community, the Therac-25, a computer-controlled radiation treatment machine, massively overdosed patients in six known incidents. (The primary sources for my discussion are Leveson and Turner's excellent and detailed account [13] and an earlier paper by Jacky [9].) These overdoses, which occurred over a two-year period from 1985 to 1987, caused severe radiation burns which in turn caused death in three cases, and irreversible injuries (one minor, two very serious) in the other three. Built by Atomic Energy of Canada Limited (AECL), the Therac-25 was the further development in a line of medical linear accelerators which destroy cancerous

tumors by irradiating them with accelerated electrons and X-ray photons. Computer controls were far more prominent in the Therac-25, both because the machine had been designed from the ground up with computer controls in mind, and because the safety of the machine was largely left to software. Whereas earlier models had included hardware safety mechanisms and interlocks, designers of the Therac-25 did not duplicate software safety mechanisms with hardware equivalents.

The origin of the malfunction was traced not to a single source, but to numerous faults, including (among others) at least two significant software coding errors ("bugs"), and a faulty microswitch. The impact of these faults was exacerbated by the absence of hardware interlocks, obscure error messages, inadequate testing and quality assurance, exaggerated claims about the reliability of the system in AECL's safety analysis, and, in at least two cases, negligence on the parts of the hospitals where treatment was administered. In one instance monitors enabling technicians to observe patients receiving treatment were not operating at the time of malfunction; in another, the clinic kept poor treatment records. Aside from the important lessons in safety engineering that the case of Therac-25 provides, it offers a lesson in accountability—or rather the breakdown of accountability due to "many hands."

If we apply standard conceptions of accountability to identify who should step forward and answer for the injuries, we see an intricate web of causes and decisions. Since we can safely rule out intentional wrongdoing, we must try to identify causal agents who were also negligent or reckless. If none can be identified, we conclude that the mishaps were truly accidental, that no one is responsible, no one is to blame. First, consider the causal antecedents: AECL designers, safety engineers, programmers, machinists, corporate executives, hospital administrators, physicians, physicists, and technicians. Since each group bore a significant causal relationship to the existence and character of the machine, it is reasonable to examine their relationship to the malfunction, the massive overdoses, deaths and injuries. The machine technicians, who entered the doses and push buttons, are the most direct causal antecedents. The others are more distant. In one of the most chilling anecdotes, a machine technician is supposed to have responded to the agonized cries from a patient by flatly denying that it was possible he had been burned.

Although the machine technicians are most closely causally linked to the outcomes, they are not necessarily accountable. The second condition on responsibility directs the search to faulty action (the fault condition). According to Leveson and Turner's account, which spotlights the work of software engineers and quality assurance personnel, there is evidence of inadequate software engineering and testing practices, as well as a failure in the extent of corporate response to signs of a problem. The safety analysis was faulty in that it systematically overestimated the system's reliability, and evidently did not consider the role software failure could play in derailing the system as a whole. Computer code from earlier Therac models was used on the Therac-25 system and assumed unproblematic because no similar injuries had resulted. Further investigation showed that although the problems had always been present, because earlier models had included mechanical interlocks, they simply had not surfaced.

The practical implications of diminished accountability are tragically clear in the deaths and injuries at six different locations where Therac-25 accelerators were used. Until the physicist Fritz Hager, at Tyler Hospital, Tyler, East Texas took it upon himself to trace the source of the problem, and many months later, the FDA stepped in, insisting on a regimen of upgrades and improvements, early responses to reports of problems were lackluster. AECL was slow to react to requests to check the machine, understand the problem, or to remediate (for example by installing an independent hardware safety system). Even after a patient filed a lawsuit in 1985 citing hospital, manufacturer, and service organization as responsible for her injuries, AECL's follow up was negligible. For example, no special effort was made to inform other clinics operating Therac-25 machines about the mishaps. (Because the lawsuit was settled out of court, we do not learn how the law would have attributed liability.)

In sum, the Therac-25, a complex computer-controlled system, whose malfunction caused severe injury, provides an example of the way many hands can lead to an obscuring of accountability. Because no individual was both an obvious causal antecedent and decision maker, it was difficult, at least on the face of it, to identify who should have stepped forward and assumed responsibility. Collective action of this type provides excuses at all levels, from those low down in the hierarchy who are "only following orders," to top-ranking decision makers who are only distantly linked to the outcomes.

We should not, however, confuse the *obscuring* of accountability due to collective action, with the *absence* of blameworthiness. Even Leveson and Turner, whose detailed analysis of the Therac-25 mishaps sheds light on both the technical aspects as well as the procedural elements of the case, appear unwilling to probe the question of accountability. They refer to the malfunctions and injuries as "accidents" and say they do not wish "to criticize the manufacturer of the equipment or anyone else." [13] Contrary to Leveson and Turner's own assessment of what they were doing, in identifying inadequate software engineering practices and corporate response, I think their analysis produces at the very least an excellent starting place for attributing accountability. If we consistently respond to complex cases by not pursuing blame and responsibility, we are effectively accepting agentless mishaps and a general erosion of accountability.

2. Bugs. To say that bugs in software make software unreliable and can cause systems to fail and be unsafe is to state the obvious. However, it is not quite as obvious how the way we think about bugs affects considerations of accountability. (I use the term "bug" to cover a variety of types of software errors including modeling, design, and coding errors.) The

Even when we factor out sheer incompetence, **bugs in significant number are endemic to programming.** *They are the natural hazards of any substantial system.*

inevitability of bugs escapes very few computer users and programmers and their pervasiveness is stressed by most software, and especially safety, engineers. The dictum, "There is always another software bug," [13] especially in the long and complex systems controlling life-critical and quality-of-life-critical technologies, captures this fact of programming life. Errors in complex functional computer systems are an inevitable presence in ambitious systems [3]. Many agree with the claim that "errors are more common, more pervasive, and more troublesome, in software than in other technologies," and that even skilled program reviewers are apt to miss flaws in programs [18]. Even when we factor out sheer incompetence, bugs in significant number are endemic to programming. They are the natural hazards of any substantial system.

While this way of thinking about bugs exposes the vulnerability of complex systems, it also creates a problematic mind-set for accountability. On the one hand the standard conception of responsibility directs us to the person who either intentionally or by not taking reasonable care causes harm. On the other, the view of bugs as inevitable hazards of programming implies that while harms and inconveniences caused by bugs are regrettable, they cannot—except in cases of obvious sloppiness—be helped. In turn, this implies that it is unreasonable to hold programmers, systems engineers and designers, accountable for imperfections in their systems.

An illustrative parallel can be drawn from the annals of bridge building and the collapse of the Tacoma Narrows bridge. Analysts tend to agree that although the bridge collapsed because of its defective design, no one should be blamed for it, because it was built according to the best specifications of the day and did not fall short of the state of knowledge in civil engineering. By contrast, in the case of the Challenger, another oft-cited case, critics blame NASA for their recklessness in going ahead with the launch in spite of known limitations of the O-Rings. Insofar as we accept bugs as an inevitable byproduct of programming, we will tend to draw parallels between bug-related failures and the collapse of the Tacoma Narrows Bridge: no one need "step forward" and be accountable. The problem with this as a blanket approach is that we are likely to miss the cases that more closely parallel the Challenger, in which it is important that someone "step forward" and be accountable. A more discerning approach to bugs in a system would better enable us to discriminate the "natural hazards," from those that with reasonable effort and good practices, could have been avoided. I would go further and say that even for the bugs that persist, despite reasonable efforts and good practices, there should be accountability, for reasons to be revealed in the section on recommendations.

3."It's the computer's fault": The computer as scapegoat. It is likely that most of us have experienced the bank clerk explaining an error, the ticket agent excusing lost bookings, the students justifying a late paper, by blaming the computer. But while the practice of blaming a computer, on the face of it, appears reasonable and even felicitous, it is a barrier to accountability because, having found one explanation for an error or injury, the further role and responsibility of human agents may be underestimated.

Let us try to understand why, in the first place, blaming a computer appears plausible by applying the conceptual analysis of blame discussed earlier: cause and fault. Consider first the causal condition: Computer systems frequently function as mediators of interactions between machines and humans, and between one human and another. This means, first, that human actions are distanced from their causal impacts, including harms and injuries, and second, the computer's action is the more direct causal antecedent. Thus the first condition on blameworthiness is satisfied by the computer. But causal proximity is not sufficient. We do not, for example, excuse a murderer on grounds that it was the bullet entering the victim's head and not he who was directly responsible for the victim's death.

The mental condition must be satisfied too. Here, computers present a curious challenge and temptation. As distinct from many other inanimate objects, computers perform tasks previously performed by humans in positions of responsibility. They calculate, decide, control, and remember. For this reason, and perhaps even more deeply rooted psychological reasons [26], people attribute to computers and not to other inanimate objects (like bullets) the array of mental properties, such as intentions, desires, thought, preferences, that make humans responsible for their actions. Where a loan adviser approves a loan to an applicant who subsequently defaults on the loan, or a doctor prescribes the wrong antibiotic and a patient dies, or an intensive care attendant incorrectly assesses the prognosis for an accident victim and denies the patient a respirator, we hold accountable the loan adviser, the doctors, and the attendant. Now replace these human agents with the computerized loan adviser, the expert systems (ES)

Accountability, and *the responsible practice of* **computing, are social values worth sustaining** *and, when necessary, rehabilitating.*

MYCIN, and APACHE (a computer system that predicts a patient's chance of survival [6]). While on the face of it, it may seem reasonable to associate the blame with the *functions* even though they are now performed by computer systems and not humans, the result of not working out alternative lines of accountability means ultimately a loss of accountability for that function. (For other discussions and proposed solutions see [11, 15, 23].)

We can fairly easily explain some of the cases in which people blame computers. In the first place there are cases in which an agent, by blaming a computer, is obviously shirking responsibility. In the second place, there are cases in which an agent cites a computer because he is genuinely perplexed about who is responsible. For example, when an airline reservation system apparently malfunctions, it may be that accountability is already so obscured that the computer is indeed the most salient agent. In these cases, the computer serves as a stopgap for something elusive, the one who is, or should be, accountable. In the remaining cases, in which computers perform functions previously performed by humans who were held accountable for their actions, we need to rescue accountability. It is important that the ethical issue of who is accountable not hang in the balance on an answer to the metaphysical question of whether computers really decide, calculate, intend, and think. We need to adjust the lines of accountability to identify other humans who will be accountable to the impacts of these systems.

4. Ownership without liability. The issue of property rights over computer software has sparked active and vociferous public debate. Should program code, algorithms, user interface ("look-and-feel"), or any other aspects of software be privately ownable? If yes, what is the appropriate form and degree of ownership—trade secrets, patents, copyright, or a new (*sui generis*) form of ownership devised specifically for software? Should software be held in private ownership at all? Some have clamored for software patents, arguing that protecting a strong right of ownership in software, permitting owners and authors to "reap rewards," is the most just course. Others urge social policies that would place software in the public domain, while still others have sought explicitly to balance owners' rights with broader and longer-term social interests and the advancement of computer science [17, 19, 24]. Significantly absent in these debates is any reference to owners' responsibilities.[4]

While ownership implies a bundle of rights, it also implies responsibilities. Along with the privileges of ownership comes responsibility. If a tree branch on private property falls and injures a person under it, if a pet Doberman escapes and bites a passerby, its owners are accountable. Holding owners responsible makes sense from a perspective of social welfare because owners are typically in the best position to directly control their property. In the case of software, its owners (usually the producers) are in the best position to directly affect the quality of the software they release to the public. Yet the trend in the software industry is to demand maximal property protection while denying, to the extent possible, accountability.

This is expressed in, for example, the license agreements that accompany almost all mass-produced consumer software which usually includes a section detailing the producers' rights, and another negating accountability. Accordingly, the consumer merely licenses a copy of the software application and is subject to various limitation on use and access to it. The disclaimers of liability are equally explicit: "In no event will Danz Development Corporation, or its officers, employees, or affiliates be liable to you for any consequential, incidental, or indirect damages . . . "; "Apple makes no warranty or representation, either expressed or implied, with respect to software, its quality, performance, merchantability, or fitness for a particular purpose. As a result, this software is sold 'as is,' and you, the purchaser are assuming the entire risk as to its quality and performance." The Apple disclaimer goes on to say, "In no event will Apple be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect in the software or its documentation, even if advised of the possibility of such damages."

[4]For an exception see Samuelson's recent discussion of liability for defective information [20].

Maintaining Accountability in a Computerized Society—Recommendations

An underlying premise of this article—and one I hope is shared with readers—is that accountability, and the responsible practice of computing, are social values worth sustaining and when necessary, rehabilitating. We have seen how features of the organizational contexts in which computer systems and computerized systems are created, and broadly held views about the power and limitations of computing can erode accountability for risks and injuries; namely, many hands, bugs, computers-as-scapegoat, and ownership without liability. Rehabilitating accountability in a computerized society does not, however, imply an obsession with pinning the blame on someone, or an insistence that someone be punished no matter what.

Rather, it recommends an approach to harms, injuries, and risks, that is cognizant of the contexts and assumptions that are apt to obscure accountability. We should hold on to the assumption that someone is accountable, unless after careful investigation, we conclude that the malfunction in question is, indeed, no one's fault.

Beyond this general approach to rehabilitating accountability, I propose three specific lines of approach to promote accountability—one conceptual, the other two practical. The recommendations are addressed to the professional community—those actively engaged in the computing profession, their professional organizations, and the institutions that educate them. They are addressed also to policy makers, and to all of us living in this increasingly computerized society. With lines of accountability recovered, responsibility for the impacts of computing will, we hope, become as clear to the computing profession and the rest of society, as to the boy in the back row taking the first step forward toward accountability.

1. Keep accountability distinct from liability to compensate. The problem of many hands is profound and unlikely to yield easily to any general, or slick, solution. Greater success, at least for the present, is likely to come from careful case-by-case analysis, in which accountability is determined according to the details of a specific situation. A good system of liability offers a *partial* solution because, while we wrestle with the conceptual puzzles of blame and accountability, at least the needs of victims are being addressed.

Liability, however, is not the same as accountability. It ought not be accepted as a substitute for it because this would further obscure accountability. Liability is grounded in the plight of a victim. Its extent, usually calculated in terms of a sum of money, is determined by the degree of injury and damage suffered by the victim. For example, when harm is the result of collective action, because the weight of compensation can be shared, its burden on each agent is considerably eased. Furthermore, since compensation is victim-centered, identifying one satisfactory source of compensation lets others "off the hook." By contrast, accountability is grounded in the nature of the action, and the relationship of the agent to an outcome. (Recall the causal and fault conditions.) If several individuals are collectively responsible, we hold each fully accountable because many hands ought not make the burden of accountability light. Further, holding one individual accountable does not let others off the hook because several individuals may all be fully accountable for a harm.[5] From the general annals of engineering ethics, the fatal calculation of Ford executives in which they offset the value of life and injury against the cost of improving the Pinto's design, we see an example in which considerations of liability were primary. Had they been thinking about accountability to society and not merely liability, they would surely have reached a different conclusion.

2. Clarify and vigorously promote a substantive standard-of-care. A growing literature, including several of the articles cited earlier (for example, [13, 18]) discusses guidelines for producing safer and more reliable computer systems. Among these guidelines is a call for simpler design, a modular approach to system building, formal analysis of modules as well as the whole, meaningful quality assurance, independent auditing, built-in redundancy, and excellent documentation. If such guidelines were to evolve into a standard of care, taken seriously by the computing profession, promulgated through educational institutions, urged by professional organizations, and even enforced through licensing or accreditation (a controversial issue), better and safer systems would be the direct result. Another result of a standard of care would be a nonarbitrary means of determining accountability. The standard of care provides a tool to distinguish between malfunctions (bugs) due to inadequate practices and those that occur in spite of a programmer or designer's best efforts. A standard of care provides a tool for distinguishing the analogs of the Tacoma Narrows Bridge, from those of the Challenger space shuttle. For example, had the guidelines discussed by Leveson and Turner been known at the time the Therac-25 was created, we would have been able to conclude that the developers of the system were accountable for the injuries. In not meeting these standards they were negligent.

A standard of care could also be of benefit to systems engineers working within large organizations. It would provide an explicit measure of excellence that functions independently of pressures imposed by the organizational hierarchy.

3. Impose strict liability for defective consumer-oriented software, as well as for software whose impact on society and individuals is great. Strict liability would shift the burden-of-accountability to the producers of defective software and thereby would address an anomaly (perhaps even a paradox) in our current system of liability. We have seen, on the one hand, that strict liability is a way of assuring that the public is protected against the potential harms of risky artifacts and property. On the other hand, while the prevailing lore portrays computer systems as prone to error in a degree surpassing most other technologies, most producers of software explicitly deny accountability for harmful impacts of their products, even when they malfunction. Software seems, therefore, to be *precisely* the type of artifact for which strict liability is necessary—assuring compensation for victims, and sending an emphatic message to producers of software to take *extra*ordinary care to produce safe and reliable systems.

Conclusion

In the twentieth century B.C. the Code of Hammurabi declared that if a house collapsed and killed its owner, the builder of the house was to be put to death. In the twentieth century A.D. many builders of computer software would deny responsibility and pass the "entire risk" to the user. While the centuries have placed

[5]Compare this to the judge's finding in the "Red Hook Murder." Even though it was almost certainly known which one of the three accused pulled the trigger, the court viewed all three as equal and "deadly conspirators" in the death of Patrick Daley.

a distance between the harsh punishments meted out by Hammurabi's Code and contemporary legal codes, the call for accountability remains a standard worth restoring, and one whose achievement would be a source of professional pride.

References

1. Borning, A. Computer system reliability and nuclear war. *Commun. ACM 30*, 2 (1987), 112–131.
2. Clement, A. Computing at work: Empowering action by 'low-level users.' *Commun. ACM 37*, 1 (Jan. 1994) (this issue).
3. Corbato, F.J. On building systems that will fail. *Commun. ACM 34*, 9 (1991), 73–81.
4. De George, R. Ethical responsibilities of engineers in large organizations: The Pinto case. In *Collective Responsibility*, L. May and S. Hoffman, Eds., 1991, Rowman and Littlefield, pp. 151–166.
5. Feinberg, J. Collective responsibility. In *Doing and Deserving*, J. Feinberg, Ed., 1970, Princeton University Press, Princeton, N.J.
6. Fitzgerald, S. *Hospital Computer Predicts Patients' Chances of Survival.* The *Miami Herald*, 1992, Miami, p. 6J.
7. Hart, H.L.A. *Punishment and Responsibility.* Clarendon Press, Oxford, 1968.
8. Hart, H.L.A. and Honore, T. *Causation and the Law.* Second ed., Clarendon Press, Oxford, 1985.
9. Jacky, J. Safety-critical computing: Hazards, practices, standards and regulations. Unpublished manuscript. University of Washington, 1989.
10. Johnson, D.G. and Mulvey, J.M. *Computer Decisions: Ethical Issues of Responsibility and Bias.* Statistics and Operations Research Series, Princeton University, 1993.
11. Ladd, J. Computers and moral responsibility: A framework for an ethical analysis. In *The Information Web: Ethical and Social Implications of Computer Networking*, C.C. Gould, Ed. Westview Press, Boulder, Colo. 1989.
12. Leveson, N. Software safety: Why, what, and how. *Comput. Surv. 18*, 2 (1986), 125–163.
13. Leveson, N. and Turner, C. An investigation of the Therac-25 accidents. *Computer 26*, 7 (1993), 18–41.
14. Littlewood, B. and Strigini, L. The risks of software. *Sci Am.* (1992), 62–75.
15. Moor, J. What is computer ethics? *Metaphilosophy 16*, 4 (1985), 266–275.
16. Neuman, P.G. Inside RISKS. *Commun. ACM.*
17. Nissenbaum, H. Should I copy my neighbor's software? *Comput. Philos.* To be published.
18. Parnas, D., Schouwen, J. and Kwan, S.P. Evaluation of safety-critical software. *Commun. ACM 33*, 6, (1990), 636–648.
19. Samuelson, P. *Adapting Intellectual Property Law to New Technologies: A Case Study on Computer Programs.* National Research Council, 1992.
20. Samuelson, P. Liability for defective information. *Commun. ACM 36*, 1 (1993), 21–26.
21. Schuler, D. Community networks: Building a new partipatory medium *Commun. ACM 37*, 1 (Jan. 1994) (this issue).
22. Smith, B. The limits of correctness. Center for the study of language and information, Rep. CSLI-85-35, Stanford, 1985.
23. Snapper, J.W. Responsibility for computer-based errors. *Metaphilosophy 16* (1985), 289–295.
24. Stallman, R.M. The GNU manifesto. *GNU Emacs Manual*, 1987, 175–84.
25. Thompson, D. The moral responsibility of many hands. In *Political Ethics and Public Office.* Harvard University Press, Cambridge, Mass. 1987, pp. 40–65.
26. Turkle, S. *The Second Self.* Simon & Schuster, Inc., New York, 1984.
27. Velasquez, M. Why corporations are not morally responsible for anything they do. In *Collective Responsibility*, L. May and S. Hoffman, Eds. Rowman and Littlefield, 1991, pp. 111–131.
28. Weizenbaum, J. On the impact of the computer on society. *Science 176*, 12 (1972), 609–614.

CR Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging; D.4.5 [**Operating Systems**]: Reliability; K.1 [**Computing Mileux**]: The Computer Industry—*standards*; K.4.1 [**Computers and Society**]: Public Policy Issues—*human safety, regulation*; K.5.0 [**Legal Aspects of Computing**]: General; K.7.m [**The Computing Profession**]: Miscellaneous—*codes of good practice*; ethics; K.7.3 [**The Computing Profession**]—*testing, certification, and licensing*

General Terms: Human Factors, Legal Aspects

Additional Key Words and Phrases: Liability, responsibility

About the Author:

HELEN NISSENBAUM is associate director of the Center for Human Values. Her research interests in computer ethics include property rights over software, the value dimensions of computing, accountability, and privacy. **Author's Present Address:** University Center for Human Values, Marx Hall, Princeton University, Princeton, NJ 08540. Email: helen@phoenix.princeton.edu