Informatics and Professional Responsibility*

Donald Gotterbarn, *East Tennessee State University, USA*

Keywords: computer ethics, informatics, professional responsibility

ABSTRACT: *Many problems in software development can be traced to a narrow understanding of professional responsibility. The author examines ways in which software developers have tried to avoid accepting responsibility for their work. After cataloguing various types of responsibility avoidance, the author introduces an expanded concept of positive responsibility. It is argued that the adoption of this sense of positive responsibility will reduce many problems in software development.*

INTRODUCTION

In the summer of 1991 a major telephone outage occurred in the United States because an error was introduced when three lines of code were changed in a multi-million-line signaling program. Because the three-line change was viewed as insignificant, it was not tested. This type of interruption to software systems is too common. Not only are systems interrupted but sometimes lives are lost because of software problems. A New Jersey inmate under computer-monitored house arrest removed his electronic anklet. "A computer detected the tampering. However, when it called a second computer to report the incident, the first computer received a busy signal and never called back."[1] While free, the escapee committed murder. In another case innocent victims were shot to death by the French police acting on an erroneous computer report.[2] In 1986 two cancer patients were killed because of a software error in a computer controlled X-ray machine. Given the plethora of these kinds of stories, it is not surprising that informatics and computing have not enjoyed a positive image.

* An earlier version of this paper was written for inclusion in S. Rogerson and T. W. Bynum, eds., *Computer Ethics and Professional Responsibility*, Blackwell, (in press). Prepublished here with permission from the author and the editors.

Address for correspondence: Donald Gotterbarn, Professor, Computer and Information Sciences, East Tennessee State University, Box 70711, Johnson City, TN 37614, USA; gotterba@access.etsu.edu (email).

Published by Opragen Publications, POB 54, Guildford GU1 2YF, UK. http://www.opragen.co.uk

How can such problems result from the actions of moral software developers? The existence of such cases is a problem, but that is not my major concern in this paper; rather my concern is the narrow concept of responsibility which contributes to these disasters. I argue that, although informatics has been undergoing a rapid development, there has been only minimal corresponding development in the concept of responsibility as it applies to computing practitioners (CPs). Computing is an emerging profession that will not succeed until it has expanded its sense of responsibility. I describe a broader concept of responsibility that is consistent with professionalism in computing.

The focus of cases like the ones cited above is computer failures. In the early days of computing, CPs sought immunity from blame for their failure to develop reliable systems. CPs developed their own special language. Flaws in computer programs were not errors introduced by the programmer but were "bugs" found in the program. Notice how the emphasis is on finding the "bug" and not on determining how it got into the program or taking preventative action so that similar "bugs" will not get into future programs. Another favorite exculpatory euphemism used by CPs is "computer error". "I am not to blame. It was a computer error." The developer sometimes attempts to avoid responsibility for undesirable events by assigning the responsibility to the client who failed to adequately specify what was "really" needed. If the specifications are precise and the client cannot be used to exempt the developer from responsibility, the fact that "no program can be proven to be error free" is used to excuse critical system failures. And as a last resort, one can simply appeal to the complexity of the system. Complex systems are expected to fail. This is like the engineering concept of an "inevitable or normal accident". This concept holds that as the complexity of a system increases so does the likelihood of an accident. The accident should not be attributed to anyone's errors or failures to act. The implication of all of these excuses is that the responsibility for these events is borne by the computer or the complexity of the system rather than being borne by the developer of the computer system. This side-stepping of responsibility by software developers is based on inaccurate computer science. The problem here is more than bad science, these excuses are used to justify the development of systems that are detrimental to society and these excuses inhibit the development of computing as a profession.

The news media like to emphasize catastrophic cases of software development. This emphasis sometimes misleads us into ignoring questions of responsibility in more common cases of software development. Let us look at a common example in computing which can be used to illustrate a fuller, more positive concept of responsibility.

AN INADEQUATE INTERFACE

Fred Consultant, a computer consultant, developed several quality computer systems for the national government of NewLand. He attributed the quality of some of his systems to the good working relationship he had established with potential system users. The government of NewLand has an unnecessarily complex accounting system.

The system has so much overhead that administering it wastes significant amounts of taxpayer's money. Jim Midlevel, a local manager of this accounting system, understood where the waste was in the system. Even though he did not understand the day-to-day procedures of the system, he was able to design modifications to the systems which would significantly reduce the overhead costs of running it. Jim convinced his upper level management to implement his modifications to the system. Because of Fred's previous accomplishments his company was given the contract to write the first stage of the more efficient accounting system that will be used by the government and will save taxpayers a considerable amount of money. Fred met with Jim to discuss the system and carefully studied the required inputs and outputs of the revised system. Fred asked one of his best software engineers, Joanne Buildscreen to design the user interface for the system. Joanne studied the required inputs to the system and built an interface for the revised system. The system was developed and shown to Jim Midlevel. Jim was satisfied that the accounting system and the interface contained all of the functionality described in the requirements. The system passed its acceptance test which proved that all stated requirements had been met. The system was installed, but the user interface was so hard to use that the complaints of Jim's staff were heard by his upper level management. Because of these complaints, upper level management decided that it would not invest any more money in the development of the revised accounting system. To reduce staff complaints they would go back to the original more expensive accounting system.

What is the net result of the development effort described in this case? There is now a general ill will toward Fred's company and NewLand's officials do not give his company many contracts. The original, expensive, accounting program is back in place. The continued utilization of this program is a significant burden on the taxpayers. The situation is worse than it had been before this project was undertaken, because now there is little chance of ever modifying the system into a less expensive system.

SIDE-STEPS: AVOIDING OR DODGING RESPONSIBILITY

One of the first questions to be asked about this undesirable situation is "Who is responsible?" Generally this question is associated with seeking someone to blame for the problem. One of the reasons why the "blame-game" is so popular is that once it has been decided who is to blame, no one else needs to feel accountable for the problem. Finding a scapegoat to bear the blame for all others who may be involved is just as popular a model in computing as it is in literature.

I believe that there are two primary reasons why CPs side-step the assignment of responsibility, especially after a system failure or a computer disaster. Both of these reasons are errors grounded in misinterpretations of responsibility. These erroneous reasons are the belief that software development is an ethically neutral activity and belief in a malpractice model of responsibility.

Ethical Neutrality

The first error is that responsibility is not related to a CP because computing is understood by many CPs as an ethically neutral practice. There are a number of factors which contribute to this mistake. One factor contributing to why CPs find it reasonable to look elsewhere for someone to blame is the way we train them in the university. We train CPs to solve problems; and the examples we use, such as finding the least common multiple (LCM) for a set of numbers, portrays computing as merely a problem-solving exercise. The primary goal of the exercise is to solve the problem exactly as it is presented to the CP. All energy (and responsibility) is focused on finding a solution in an almost myopic fashion. This is analogous to the way people approach crossword puzzles. Solving the puzzle is an interesting exercise, but it generally lacks any significant consequences. There is no responsibility beyond solving the puzzle, other than properly disposing of the paper on which it is written. The same assumptions are made about solving computing problems.

The crossword-puzzle approach to computing problems leads to a failure to realize that computing is a service to the user of the computing artifact. This failure makes it easy to assign blame elsewhere. If there is no responsibility, there is no blame or accountability. The failure to see one's responsibility has other significant consequences. One result of the crossword-puzzle view is seen when we consider the real case of a programmer who was asked to write a program that would raise and lower a large X-ray device above the X-ray table, moving the machine to various fixed positions on a vertical support pole. The programmer wrote and tested his solution to this puzzle. It successfully and accurately moved the device to each of the positions from the top of the support pole to the top of the table. The difficulty with this narrow problem-solving approach was shown when, after installation, a X-ray technician told a patient to get off the table after a X-ray was taken and then the technician set the height of the device to "table-top-height". The patient had not heard the technician and was later found crushed to death between the machine and the table top. The programmer solved a puzzle but didn't consider any consequences of his solution to the user. If the programmer had considered the broader context, rather than limiting his attention to moving the X-ray machine on the pole, then he might have required an additional confirmation when moving the machine to the table top.

This first misunderstanding of responsibility is dangerous in that it is used to justify a lack of attention to anything beyond the job specification. The absurd degree to which this side-step can be taken is illustrated in the following real case. A defense contractor was asked to develop a portable shoulder-held anti-aircraft system. The specifications required that the shoulder-held system be capable of destroying a particular type of attack helicopter at 1000 yards with 97% efficiency. The system the contractor developed did effectively destroy incoming helicopters. Its kill rate was better than a 97%. It also had another feature. Because of a software error, the shoulder held missile launcher occasionally overheated to the extent that it burned off vital portions of the anatomy of the person holding the launcher. The extent of the burns killed the person who launched the missile. The government was, of course, dissatisfied with the product and declined to make the final payment to the contractor. The company took the government to court over the final payment. The company owners declared that they should be paid and that they are not responsible for the deaths because the system they developed "is in full compliance with the specifications given to them by the user". The contractors viewed this problem like a crossword puzzle. They solved a crossword puzzle exactly as it was presented to them, and they denied any further responsibility.

Diffuse Responsibility

The second side-step error is based on the belief that responsibility is best understood using a malpractice model which relates responsibility to legal blame and liability. It is important to find the correct parties to blame in order to bring legal action against them. Generally the concept of blame is tied to a direct action which brought about the undesirable event. A typical approach to determining blame is to isolate the event which immediately preceded and was causally related to the undesirable event, and then blaming the party who brought about the preceding event. In the case of NewLand's inadequate interface, Joanne's design of the interface screens was the direct cause of the user's dissatisfaction with the system. Joanne's screens were the immediate cause of the dissatisfaction so the tendency is to blame her. If the blame is both severe and public, then others will feel excused from responsibility for the unhappy event.

Joanne will not want to bear the blame and will point to other people's failures as contributing to the problem. This leads to the belief that one can avoid responsibility if the blame can be diffused by being widely distributed. This second side-step is based on the claim that individual software developers are too far from the event which causes the problem. It also distributes the blame so widely that it becomes negligible or cannot be clearly attributed.

This side-step is a paradoxical denial of responsibility since it starts by identifying multiple locations of failure of responsibility, namely the particular irresponsible acts of each member of the development team. This diffusion technique might be used in the Inadequate Interface case. Fred did not behave responsibly because he did not adequately understand the nature of the task.. Jim, because of his lack of specific system details, should have coordinated the development activities with the system users. Joanne should have shown preliminary screen designs to the system users. Everyone failed to meet his or her responsibilities. This distribution of failure is then used to deny legal fault or blame. The absurdity is that this identification of multiple individuals failing to meet their system development obligations is also used to deny each individual's responsibility. Like the first side-step, this diffusion of responsibility is a very dangerous practice. It follows from the diffusion side-step that whenever there are many people contributing to a project, no individual will be held accountable for their contributions to the project. If I am not responsible, then I have no prior commitment to do a competent job or worry about the overall quality of a product.

The diffusion of responsibility has a corollary which Ladd[3] has called “task responsibility”, which ties responsibility to one narrowly defined task. An example of task responsibility can be generated by giving more details from the “Inadequate Interface” case. What was the problem that made the interface unusable? The multiple input screens used in the new accounting system did contain fields for all the required data, but the input sequence on the screens was not consistent with the structure of the input forms used by the clerks. To enter the data from a single input form, the clerks had to go back and forth between several screens. Using task responsibility, Joanne would maintain that it is “not her job” to get copies of the input forms. If “they” wanted the sequence of the data on the screens to match the input forms, then “someone” should have provided her with sample input forms. It is not her job to get the forms.

The association of responsibility with blame leads to a variety of excuses for not being accountable. These excuses include:

a. Absence of a direct and immediate causal link to the unacceptable event,[4]
b. Denial of responsibility since a responsible act conflicts with one’s own self-interest,[5]
c. Responsibility requires the ability to do otherwise but CPs do most of their work in teams and for large organizations,[6]
d. Lack of strength-of-will to do what one thinks is right,[5]
e. Blaming the computer,[4]
f. Assuming that science is ethically neutral,
g. Microscopic vision.[7]

Both the neutrality and the diffusion side-stepping approaches to responsibility are inconsistent with efforts to professionalize computer science and engineering. Any profession should be strongly motivated to pursue the good of society. It should understand its primary function as a service to society. To professionalize computing, therefore, we need to revisit the concept of responsibility, separating it from the legal concept of blame, and separating it from direct and immediate causes of undesirable events. What sense of responsibility would meet these objections and mitigate the urge for side-stepping?

POSITIVE AND NEGATIVE RESPONSIBILITY

The philosophical concept of “responsibility” is very rich and is frequently tied to philosophical conundrums like “free will”. Philosophers have long been concerned about the relationship between individual responsibility and free will. This concern derives in part from the implicit connection of the concept of blame with the concept of responsibility. If people have no free will then it is difficult to blame them for their actions. In opposition to this dependency of “responsibility” on the concept of blame and liability, Ladd distinguished the traditional sense of responsibility – which he calls “negative responsibility” from “positive responsibility”. Negative responsibility deals with or looks for that which exempts one from blame and liability. An exemption from blame is an exemption from moral responsibility and an exemption from liability is an exemption from legal responsibility. Negative responsibility is distinguished from positive responsibility.

The concept of positive responsibility is consistent with many philosophies. One can extend Ladd's concept of positive responsibility to be justifiable under most philosophical theories. Positive responsibility can be grounded in any of the classical and contemporary theories. Such theories can be organized into a matrix created by the intersection of two of the following dimensions: rules/consequences and collective/individual.[8]

	RULES	CONSEQUENCES
COLLECTIVE	Collective rule-based	Collective consequentialist
INDIVIDUAL	Individual rule-based	Individual consequentialist

The emphasis in positive responsibility is on the virtue of having or being obliged to have regard for the consequences of his or her actions on others. We can place this sense of positive responsibility in each quadrant of the matrix. This sense of responsibility can be founded in: collective rule-based ethics based on the logic of the situation; individual rule-based ethics based on universal duties applicable to all;[9] collective consequentialists like Mill providing the greatest good for the greatest number; or individual consequentialists like Adam Smith who maintain that the social welfare is advanced by individuals doing good acts which have good consequences for society. No matter which ethical theory is used to justify positive responsibility, the focus of positive responsibility is on what ought to be done rather than on blaming or punishing others for irresponsible behavior.

Positive responsibility is not exclusive. It does not seek a single focus of blame. Negative responsibility, on the other hand, seeks a single focus of blame who, once found, exonerates all others from blame. With positive responsibility, saying that Joanne is responsible and should be held accountable for her failings does not exclude Fred. A virtue of positive responsibility is that several people can be responsible to varying degrees. Not only can we attribute responsibility to Fred, but we can say that he bears more of the responsibility in this case because he knew that Jim was only working with limited knowledge of the system.

This point illustrates a second and more significant virtue of positive responsibility, namely that it does not require either a proximate or direct cause. This extension of causal influence beyond the immediate and proximate cause is more consistent with assigning responsibility in the disasters that affect computing. Nancy Leveson,[10] in her article about the technical difficulties of the Therac-25 X-ray machine which led to multiple deaths, concludes that because of the involvement of many hands, responsibility for the Therac-25 incidents cannot be assigned. Leveson uses a limited-negative concept of responsibility and after identifying failures of multiple software engineering practices refers to the deaths that resulted as “accidents”. Nissenbaum[11] correctly criticized such an approach to responsibility when she said, “If we respond to complex cases by not pursuing blame and responsibility, we are effectively accepting agentless mishaps and a general erosion of accountability.” The positive sense of responsibility allows the distribution of responsibility to software development teams, designers, etc. and can apply the concept of responsibility even to large development teams. In the Therac-25 case there may not be a single locus of blame, but under positive responsibility the developers are still responsible.

Any preliminary definition of responsibility starts from the presumption that others are affected by the outcomes of CPs’ particular actions or failures to act. This presumption is embodied in many codes of ethics of computing associations. Such codes tend to organize responsibilities by the roles of the people involved. Most codes talk about the CPs’ responsibilities to other professionals, to the client or employer, and to society in general. Only a few codes include the obligations of CPs to students. Although such codes try to recognize most of these relationships, most of them make the mistake of not distinguishing employers, clients, and users. In Joanne’s case, her employer was Fred, the client was Jim, and the users were the accounting clerks. Because she stood in different relations to each of these parties, she owed them different and perhaps conflicting obligations. Some recent codes, such as the Software Engineering Code of Ethics and Professional Practice (SE)[12] provide the CP with techniques for adjudicating between conflicting obligations.

There are two types of responsibilities owed in all of these potential relations. One type of positive responsibility is technically based and the other positive responsibility is based on values. These two types of positive responsibility are both necessary for a concept of professional responsibility.

Positive responsibility points both forward and backward. It points backward when it identifies unmet obligations and what people ought to have done. Fred had an obligation to meet with the clerks to understand the structure of the interface they would need. This sense of responsibility goes beyond the malpractice model. Responsibility is more than just blame, there should also be some lessons learned from failures of responsibility. Thus there should be some lessons learned from the Inadequate Interface case. As a result of this event, Fred is responsible for preventing similar system’s development failures in the future. Knowledge of this kind of failure and its consequences also places responsibility on other computer practitioners and places responsibilities on the profession of computing as a whole. For example, the activity of establishing computing standards of practice is justified by the forward looking sense of positive responsibility of the CP and the responsibility of the profession.

A RESPONSE TO AVOIDANCE

The concept of positive responsibility can be used to address several of the responsibility-avoidance techniques referred to earlier. This broader concept of responsibility meets the diffusion side-step and the positive aspect of this concept of responsibility meets the malpractice side-step.

Positive Responsibility and the Profession of Computing

Computing is an emerging profession. Computing already bears several of the marks of a profession. In order for computing to be a profession there must be some agreement among its members of goals and objectives or ideology. This agreement is of two kinds. One is technological and the other kind is moral. These match technical positive responsibility and moral positive responsibility. In accordance with the malpractice model, a CP has a responsibility to conform to good standards and operating procedures of the profession. These are generally minimal standards embodied in software development models and model software engineering curricula. This kind of technical knowledge and skill does not distinguish a technician from a professional. To make this distinction one must go beyond mere technical positive responsibility.

A Broader Sense of Responsibility

In a profession, the members pledge to use their skills for the good of society and not to merely act as agents for the client doing whatever a client asks. This commitment is generally embodied in a professional organization's code of ethics. To be a professional, one assumes another layer of responsibility beyond what has been described in positive responsibility. The professional commits to a "higher degree of care" for those affected by the computing product. Most occupations have a principle of "due care" as a standard. For example, a plumber is responsible that the results of his work will not injure his customers or users of the plumbing system. But the plumber does not bear the responsibility to advise the customer of potential negative impacts a new system may have on the customer's business, customer's quality of life, or the environment. The concern to maximize the positive effects for those affected by computing artifacts goes beyond mere "due care", mere avoidance of direct harm. The addition of this layer of responsibility to positive responsibility is what is necessary to change a computing practitioner into a computing professional. The inadequate interface met the contract specifications, but it did not meet the user's needs. Although the system technically was capable of doing all the required functions and met Jim Midlevel's requests, the computing professional had the responsibility to be sure that the system met the user's needs. The forward looking sense of positive responsibility also means that the computer professional has the obligation to meet with upper-level management in order to convince them to re-instate the new accounting system. The computing professional has an obligation to the client, the users and the taxpayers.

This broader sense of responsibility goes beyond the malpractice model. It incorporates moral responsibility and the ethically commendable. This concept of professional responsibility can be used to address the above-mentioned ways used to deny accountability. This sense of responsibility provides a way to address distributed responsibility as well as diffusion of collective responsibility. The ability to deal with collective responsibility is important because it enables meaningful discussion of the "professional responsibility" of organizations which produce software and organizations which represent computing professionals. It is clear that the computing disasters mentioned at the beginning of this paper would not have occurred if computing practitioners understood and adopted the positive sense of professional responsibility. The recent development by software engineers of a code of ethics and professional practice[12] is an attempt to define for them this sense of professional responsibility.

REFERENCES

1 Joch, A. (1995) How Software Doesn't Work. *Byte*, December: 48-60

2 Vallee, J. (1982) *The Network Revolution*, And/Or Press, Berkeley, CA, USA.

3 Ladd, J. (1988) Computers and Moral Responsibility: A Framework for an Ethical Analysis, in: Gould, Carol (ed.) *The Information Web: Ethical and Social Implications of Computer Networking*, Westview Press.

4 Dunlop, C., et. al. (1991) Ethical Perspectives and Professional Responsibility, in: King, R. (ed.) *Computerization and Controversy: Value Conflicts and Social Choices*, Academic Press, California, USA.

5 Harris, C.E., et. al., eds. (1995) *Engineering Ethics: Concepts and Cases*, Wadsworth, USA.

6 Johnson, D. (1994) *Computer Ethics*, Prentice Hall., USA.

7 Davis, M. (1989) Explaining Wrongdoing, *Journal of Social Philosophy*, Spring/Fall: 74-90.

8 Laudon, K. (1995) Ethical Concepts and Information Technology, *Communications of the ACM* **38** (12): 33-40.

9 Ross, W.D. (1969) *Moral Duties*, Macmillan, USA.

10 Leveson, N., et. al. (1993) An Investigation of the Therac-25 Accidents, *IEEE Computer Magazine* **26**: 18-41.

11 Nissenbaum, H. (1994) Computing and Accountability, *Communications of the ACM* **37**: 73-80.

12 SE "Software Engineering Code of Ethics and Professional Practice", adopted by the IEEE-CS and the ACM. (1998) http://computer.org/computer/code-of-ethics.pdf. Also republished in *Science and Engineering Ethics* **7**(2): 231-238.