The Increasing Threat of Legal Liability for Software Developers

JANICE C. SIPIOR
Villanova University, USA

BURKE T. WARD,
Villanova University, USA

WILLIAM P. WAGNER
Villanova University, USA

This paper discusses the increasing threat of legal liability for developers should software malfunction and cause financial loss or harm to the user. Recent events in the software industry have signaled a changing environment for development organizations. A discussion of the mounting market expectations for software to function properly is presented to underscore the increasing potential for users to seek legal recourse. Various theories in the U.S. legal system may form the basis for legal action on the part of the user, based upon the characteristics of the individual case. The focus of this discussion is on software which is developed for sale, as opposed to in-house development, since most legal actions would arise between a vendor and purchaser. Although we conclude that the legal liability of the software developer is currently unclear and varies by jurisdiction, the threat is nonetheless present and may indeed be on the rise.

Software contributes to improvements in organizational efficiency, supports managerial decision making, and aids in gaining or maintaining competitive advantage. The resulting demand for software has attracted development organizations to the lucrative commercial and retail market. Software may be mass marketed canned software sold at retail under a shrink-wrap, included as a part of turnkey systems which require either minimal or no in-house modifications, or specially developed for systems designed to fulfill user’s particular needs. Tailored systems may be developed by an internal systems development team or may be outsourced, but the trend toward outsourcing is rising (Lacity and Hirschheim, 1993). Developing software for use outside of the development organization may open the floodgates to potential legal liability. If software is not deemed to function properly, do the users have any legal recourse? Are programmers, system analysts, Information Systems (IS) managers, or organizations involved in development efforts exposed to legal liability? Many developers are rightly concerned about the extent of their liability should the software they developed and sold malfunction (Samuelson, 1993).

This paper seeks to raise the awareness of programmers, system analysts, IS managers, and organizations involved in software development about the increasing threat of legal liability should software malfunction and cause financial loss or harm to the user. A discussion of the mounting market expectations for software to function properly is presented to underscore the increasing potential for users to seek legal recourse. Various theories in the U.S. legal system may form the basis for legal action on the part of the user, based upon the characteristics of the individual case. These are reviewed to inform developers of the potential bases for cases against
them. The focus of this discussion is on software which is developed for sale, as opposed to in-house development, since most legal actions would arise between a vendor and purchaser. Although we conclude that the legal liability of the software developer is currently unclear and varies by jurisdiction, the threat is nonetheless present and may indeed be on the rise.

Expectations of Properly Functioning Software

Should software be expected to contain errors or function properly all of the time? Flaws, resulting in both trivial and dire consequences, have plagued software for decades. A classic example of a serious malfunction occurred in a U.S. nuclear missile warning system on October 5, 1960 (Belsie, 1994). Radar sensor input from Thule, Greenland was erroneously interpreted as a massive attack by the then Soviet Union, with a certainty of 99.9 percent. The actual cause of the warning system to issue the attack alert was identified, precluding nuclear warfare. The rising moon had caused echoes from the radar sensors, a factor overlooked during development. Unfortunately, software defects are not always detected in advance of disaster. Perhaps one of the most tragic incidents associated with software malfunction occurred in a system developed by Atomic Energy of Canada Ltd. to control radiation doses delivered to cancer patients (Byte, 1995). Onscreen editing performed with the up-arrow key caused two modes of operation to be mixed, resulting in a radiation dose more than 100 times higher than the average dose (Joyce, 1987). At least four patients believed to have received the erroneous radiation overdose died; others were seriously injured.

Software errors have become more prevalent, ranging from financial loss to life-threatening situations (Henderson, 1995). The reason for the increasing number of software failures is not just because software itself is more prevalent (Gianturco, 1995). A widespread move away from highly standardized mini or mainframe systems to networks, comprised of various brands of hardware and software connected across long distances, has occurred. Coupled with the complexity of such networks is more flexibility in user interaction, marked by the emergence of Microsoft’s Windows. Various tasks can be performed with no predetermined sequence or combination of events, unlike old programs which performed tasks through a structured series of command sequences. The new style software requires exhaustive testing to assess every possible sequence, permutation, and combination of events, which is virtually impossible (Diefenbacher, 1995). In response, the automated testing tool market has experienced rapid growth, projected to double sales in one year to $100 million (LaMonica, 1995). Automated testing tools capture the input of a human tester and generate test scripts to be run repeatedly. Errors are detected and testing resumes once the cause is determined and the fault repaired. However, each subsequent error is more difficult to detect and correct. Although automated testing tools are increasingly available, only about 75% of the code in the 60 leading products in the software industry has been tested (Henderson, 1995). In the overall development community, only about 35% of the code in a typical application is tested (Henderson, 1995). The top four development organizations, however, have been reported to be committed to quality development, detecting up to 95% of software defects before delivery to users (Henderson, 1995).

In the highly competitive software market, development organizations are driven to develop software rich in both features and quality, within a tight schedule (Moskun, 1996). Even developers with the strictest quality control may distribute software with some remaining defects. What defects remain may or may not be known by the developer before the software is released. Users have reported frustrating instances of hours wasted in attempting to get software to perform some simple task, only to have the development organization finally admit the software will not function in the manner desired due to a known problem (Foster, 1996a). Negative effects on sales may force the industry to inform users of known defects (Rigdon, 1995a).

A legal implication of the argument that some level of software defects may be inevitable is the proposal for revisions to the Uniform Commercial Code (UCC). Disclosure of known defects could ultimately be required by law, should proponents be successful in including this provision in the proposed Article 2B of the UCC (Foster, 1996b). Development organizations would be held liable should they not make a good faith effort to appropriately disclose known defects. This approach may be preferable to both developers and users. If known defects are disclosed appropriately, the developer’s liability could be reduced since the user could read about and plan for any potential defects or incompatibilities in advance of software purchase or use. To disclose known defects may seem like a simple request to developers, but what exactly is a defect, as opposed to a design decision, a feature, or a performance limitation? Since no standard exists in the software industry for certifying the quality of software, the determination remains a matter of personal opinion (Patterson, 1990).

The increasingly complex computing environment has made it more difficult to develop complex systems, correct in terms of the design specifications, with no defects. Even if it were possible, the development team would certainly be unable to foresee and accommodate all unanticipated circumstances which may arise during use. “Are we justified in saying that the designer, who operated without the benefit of hindsight, can be held liable for the flaw?” (Denning, 1990). Many software errors have been attributed to human error, rather than to the design (Norman, 1990). “Software’s written by people and people make mistakes” (Joyce, 1987). Will users accept the inevitability of flawed software without seeking recourse, even in cases of devastating consequences?