Communications of the IIMA

Volume 8 | Issue 1 Article 6

2008

Ethical Issues in the Software Quality Assurance Function

Jim Nindel-Edwards
Microsoft, Inc.

Gerhard Steinke
Seattle Pacific University

Ethical Issues in the Software Quality Assurance Function

Jim Nindel-Edwards
Microsoft, Inc.
USA
jimne@ieee.org

Gerhard Steinke
Seattle Pacific University
USA
gsteinke@spu.edu

ABSTRACT

The responsibility for correct execution of software, as well as its fitness in any given setting, becomes increasingly complex, especially when the software impacts life and death. This paper addresses the role of the software quality assurance (SQA) function and explores the ethics of the SQA organization as the last point of contact between the software manufacturer and the consumers of the software application. We explore the range of potential ethical issues, possible mitigation strategies, and responsibilities to disclose findings to both software manufacturers and consumers of software applications.

INTRODUCTION

Software plays a role in many facets of our lives—from software embedded in ovens, to software in cars, and software applications in offices. Some of these systems don't have much impact on our lives, should they fail. If the microwave doesn't come on when set to time bake, or an email to a friend does not get sent, it typically is of little consequence. On the other hand, the failure of some software could cause catastrophic consequences for individuals and even society. The failure of the antilock brake system in a car, or the avionics in an airplane, or the software in an x-ray machine could have serious consequences.

One widely cited software related accident in safety-critical systems involved a computerized radiation therapy machine called the Therac-25. Between June 1985 and January 1987, six known accidents involved massive overdoses by the Therac-25, with resultant deaths and serious injuries. "Overdoses, although they sometimes involved operator error, occurred primarily because of errors in the Therac-25's software and because the manufacturer did not follow proper software engineering practices." (Leveson & Turner, 1993; Birsch, 2005).

Early on we learn, not only to accept but perhaps even expect that software bugs and errors are a normal part of computer applications. The small print when opening and installing software packages may specify that the software manufacturer provides no guarantee of reliability or expectation of correctness. It may even appear that the marketplace rewards low quality in that it may reward additional features and timely release dates, even if they come at the expense of quality. "Companies find that it is cheaper to weather the occasional press storm, spend money on PR campaigns touting good security and fix public problems after the fact, than to design security in from the beginning" (Schneier, 2007).

Who has the ethical responsibility for testing software? Schneier (2005) says that the software vendors that should be liable, not individual programmers. If end users could sue software manufacturers for product defects, then the cost of those defects to the software manufacturers would rise. And if the cost of poor software is higher, there would be more incentive for manufacturers to make their software more secure and with fewer defects.

Life and death implications are usually associated with doctors. Software engineers and software quality assurance may face similar "life and death" situations and have corresponding high ethical responsibilities (Qureshi, 2001).

It would seem that after decades of software development there would be some assurance that software works as specified in the customer requirements. Is it that software vendors are unwilling to perform sufficient testing? Is it possible to test everything? Finding a certain number of bugs, doesn't mean that the software has no more bugs. On the other hand, not finding any defects doesn't mean there aren't any defects in the software either. Perhaps there are known bugs, but the time and resources to fix these bugs and defects are often not provided and the software is released with known (but not publicly stated) bugs. Is it because there is a low expectation of quality? Is it even possible to get rid of all bugs, especially when we are integrating components from multiple sources and we are dependent on the software that was developed and tested by others?

Software quality assurance is a challenging task. There are many questions raised by software being released with defects. What are the ethical responsibilities of a software vendor releasing software with bugs, especially if it is system-critical software, but also when releasing non system-critical software? In this paper we look at such ethical issues faced by organizations during the software quality assurance and testing process. We suggest that the higher, ethical approach is for software vendors to be open about the quality of the testing, and be willing to share known defects in their systems.

Industry Standards for Software Quality

It seems that typical commercial software is created and modified, and then, due to time-to-market pressures or cost considerations, the developer may limit the software quality assurance function and not choose to conduct independent reviews or complete testing of the software. Most commercial code is only reviewed or tested to an acceptable level of confidence to meet the business objectives of the manufacturer and perhaps/hopefully for the criticality of the software.

But there are industry standards requiring and specifying software quality assurance standards. In 1992, the Radio Technical Commission for Aeronautics (RTCA) approved specification DO-178B for the aviation industry. This specification, the Software Considerations in Airborne Systems and Equipment Certification, was created to provide the aviation community with guidance for determining, in a consistent manner and with an acceptable level of confidence that software aspects of airborne systems and equipment comply with airworthiness requirements. There are multiple levels of rigor that are applied to this software, from level E to level A. For example, at level A, the most stringent, the DO-178B objectives must be verified by independent parties, no dead code is allowed in the system, and all the requirements, code, and test information must be audited and traced. This guidance ties together system requirements, system life cycle processes, safety assessment processes, software life cycle processes, and documented traceability to show that the software has been appropriately tested.

Historically, DO-178B was mandated for air transport class of aircraft and commercial avionics to comply with Federal Aviation Administration (FAA) regulations in safety critical systems. However in recent years, due to the Global Aviation Traffic Management (GATM) agreement which has international validity and applicability, airborne military and space systems must also comply with DO-178B guidelines and certification for the safety of all aircraft (Rose, 2003).

Criteria for Ethical Software Quality Assurance

Moving beyond the industry standards we must establish some ethical standards and expectations for software testing. It would seem obvious that the ethical standard should be based on the nature of the application and the criticality of use. As a trivial example, it would be inappropriate to use the same standard of software quality for a demonstration application as we would apply to software that lands a commercial airplane safely in zero visibility (fog) at an airport. Software intended to demonstrate functionality, for example, applications to demonstrate prototypical web sites, may be of low overall software quality yet still serve their purpose well. Embedded avionics software used to land commercial airplanes in zero visibility, however, have a much higher requirement testing and verification expectation. To apply one standard (demo vs. flight critical) would either:

a) Cause much more software testing costs than warranted (applying flight critical standards to demo software), or
b) Put passengers and equipment at grave risk by applying the more casual software development and test standards associated with demonstration software. Failure of the flight control system would be a critical failure, and reflects a need for a higher level of testing.

While the bulk of any software quality assurance effort will be to address the functionality of software, the concept of fitness for use clearly has a significant influence on the software test effort and, by extension, on the measure of a quality testing effort. For example, applying the embedded avionics software test standard to a shrink wrap product – say a photo editing package bundled with a digital camera – would be an inappropriate test standard that would add significantly to the cost of the camera in a very competitive market. Simply stated, a camera company could not recoup the cost of quality invested in the software package in the sale of the associated camera.

One could imagine a scale of some sort, based on the criticality of the application along with user impact and expectation. On the one end would be the expectation that the software will have defects and may provide inaccurate results. On the other extreme would be the expectation of software that has been completely tested and works according to functional and non-functional requirements (Nindel-Edwards & Steinke, 2007).

Another aspect of this issue is the cost of quality, remembering that work put into verification that software is "well tested", in general, adds no real value to the software product. It is only assurance that the software will perform its function to the requirements of the application. If the cost of quality associated with any given product exceeds the benefit (revenue) associated with the product it obviously is a poor business investment to engage in that type of business model.

Looking to the ethical side of the software quality assurance issue brings us to four key subjects:

1) The business decision on how much of the software development cost to invest in software testing. What is good and sufficient and complete testing? The question, how much to test, is a business decision – potentially with serious ethical and legal ramification – on how much time, effort, and money is to be invested in assuring the quality of the software product.

2) Communication to the user of the software. Communication to the software user what level of testing was applied to the software product is a type of disclosure statement. Interestingly enough, testing can "add value" to a software product. All other things being equal, most of us would choose a well test application over one that had minimal testing

3) Defects detected in the package and released into the "production" environment. The choice to release known defects into "production" undoubtedly requires serious consideration for disclosure on product release notes. To not disclose is a significant ethical issue, with potential legal ramifications. The timing of the disclosure of the remaining defects in the software may also be critical. Releasing information regarding known defects before one buys the product may discourage sales. Releasing defect information when the user calls in for help may be too late.

4) Consequences of defects beyond the intended use of a software application. From a security perspective, for example, if an application might allow a hacker to gain access and extract personally identifiable information (PII) and/or high business impact (HBI) information from an application the results may not be "life or death" of an individual, but could initiate a life or death situation for an organization once the mitigation costs are fully realized.

The obvious starting point for ethical software quality assurance then starts with assurance that the software product meets the product specification, however formal or informal these specifications may be. While much of a software tester's job is finding defects, defects are simply an artifact of either a code or interpretation issue in the development phase of the project. If one were to envision a software product that was wholly without defects, however unlikely that might be, the software tester's function would still be to verify the correctness of the product prior to releasing it into the production environment, or shipping the product to consumers.

The first ethical obligation for software testing then is to assure that the product is completely tested, and does in fact meet the specifications. To provide backing for this claim the quality assurance function will typically prepare test plans, generate test cases, and execute tests producing test results, and prepare a test summary report for the project management. Note here that even in the absence of defect reports (aka bugs) the test function has produced at least two documents and as many test cases and test results as required to demonstrate that all the features of the application work correctly.

At this point we must look briefly at the exact nature of the tests themselves to further examine the ethics of testing. While at a minimum a well tested software product will be verified to perform its required functions, a truly robust test plan will also verify that product responds reasonably when the input to the application is less than perfect, and if the operation environment is not properly set up that the application responds reasonably. Case in point, some application requirements will specify responses to the user should their input not match the requirements (e.g. alphanumeric characters entered in a zip code) or the need for logging errors in a log file. But many requirements/specifications are silent in this area. The software test function here – in a good test plan – will call for the testing of all error conditions to assure that the application responds gracefully when the input and/or environment is not ideal.

Another ethical testing criterion is to test not only the correctness of the application's functionality given good input, but to test the error conditions as well. To not test error handling – in all its forms – is to assume (as unfortunately some developers assume) that the data presented to the application is always "good", and that an application's response to erroneous data is a user problem, not the application's problem.

A further step in the quality assurance process takes us into the most complex area – that of "fitness for use" of the application. This in itself can be bifurcated into, first – meeting the stated requirements for the software product, and, second – exploring the application's use when applied to situations that the software product was not in fact designed or envisioned to support. We will explore both of these perspectives.

Looking first to the stated and/or implied requirements of the application, one would expect that the artifacts of software testing would verify not only the behavior of the application against the specification, but also against the intended use of the application. In point of fact here, what is actually being tested – with the real application in hand – is the correctness of the specification intended to fulfill the application requirement. This would be better accomplished with a detailed review of the requirements and a detailed review of the specification – i.e. do the specifications actually fulfill the requirements? An interesting discussion in the application of quality assurance to a development cycle is the ethical question, even if the best quality assurance practices have been applied, should we in fact test the application back to the base requirements? Recognizing that thorough specification/design/build/test/defect fix cycles that many decisions can be made, we suggest that the answer is "yes" – testing should in fact verify that the base requirements are in fact met.

Fitness for use also has a different perspective. What if the application is applied to some use that was not covered by the base requirements and/or not considered to be "in scope" for the requirement and/or specifications? Depending on the nature of the application fitness of use can be a very narrow or very broad subject. For embedded software fitness of use is typically defined by the hardware associated with the product, an example being a pacemaker. On the other hand, particularly in the case of shrink wrap software, fitness of use could also be quite broad. An example could be a spreadsheet application used for calculating orbit re-entry for say, the space shuttle. If the precision specifications for the spreadsheet application were not up to the requirements for the applicable calculations we would not typically assign fault to the spreadsheet, but the engineering function that chose to use the spreadsheet application to perform this critical calculation. If, on the other hand, the specification (and documented behavior of the spreadsheet) suggested that there was sufficient accuracy in the calculations, and software testing did not verify that the calculations met these specifications, is there an ethical issue? We suggest that there is the obligation that every documented characteristic of a software product can, and must be tested, to verify the integrity of the software specifications. An unknown bug in complex calculations could easily impact many use cases. In particular, a known calculation bug, undisclosed, can easily lead to unwarranted confidence in the application. Also, cost/benefit and risk analysis approaches can be used to determine how much effort needs to be applied to validate that software applications are safe from unintended information disclosure at the hand of hackers (Takanen et al., 2004)

What if a third party vendor releases an update to their module? Does the whole system have to be re-tested, to ensure that the updated code still enables the whole system to function as before? The question remains, as to the responsibility for third party software. If I include code from a third party, what are my expectations of correctness for this piece of software, as well as for its integration into my system? Do I need to (re-?) test that third party software or can I assume that it is correct?

Moving beyond fitness for use then takes us to the final area which might be termed "prevention of abuse". Software – in the wrong hands – poses many threats ranging from information disclosure to system failure. A software virus running on a workstation could cause catastrophic problems. We are inclined to believe that software vendors (particularly operating system vendors) will consider and test for these conditions (Weckert, 2001). The more critical the function, the more important and valuable the data being maintained by the software, the more important it is to test to what degree the system/data is vulnerable to abuse from an outside agent. The software producer cannot be expected to take full responsibility for poor installation environments, e.g., insecure configuration settings in corporate firewalls, but neither can they escape responsibilities if their software package has inherent vulnerabilities, e.g., cross site scripting and/or SQL injection issues.

Another question to address is the use of open source software. For example, is there a higher expectation of the correctness of Microsoft's SQL versus the open source MySQL? Is there an ethical question as to supportability of the product and the modules integrated into it? What are the ethical implications of integrating a module that doesn't have support, e.g., perhaps open source software, or software no longer supported by a vendor?

In summary then we see four areas where the ethical issues confront the software quality assurance function:

1. Testing the application's functionality, per the test plan, and recording of defects (material and non-material) as a natural function of the test process. Recording of test results – independent of the recording of defects – is essential here to demonstrate that the testing has been accomplished per the test plan and leaves the artifact of what was tested.
2. The test plan, test cases, and test execution, must go beyond the obvious simple testing of the functionally (positive testing) to test predictable error conditions. If people can enter incorrect or unacceptable data into the application, it will happen. Verifying that the application's response to such invalid input is meaningful, actionable, and has no negative consequences (ranging from data loss to dangerous situations for embedded code) is the responsibility of the test function.
3. Testing beyond the functional testing, specifically reviewing all the environmental, security, performance, reliability, and maintainability issues associated with the application in its intended deployment environment, and potential deployment environment. This is not to say that the tester must actually test the application in environments and/or situations that go beyond the target deployment environments but appropriate consideration of such issues as, for example, overloading a web site, should be reviewed by the test function.
4. Disclosure to the project team all defects, tests executed (or not executed), results and findings. Disclosure to the intended (or unintended) users of the application of "material defects" to allow them to be fully informed of potential problems associated with the application. The decision of materiality here is best worked with the project management and with legal counsel. But ultimately the ethical quality assurance function cannot delegate this responsibility to project management (and in particular the marketing function), otherwise the user to the application may miss important limitations and/or (material) shipped defects in the product.

CONCLUSIONS AND FUTURE RESEARCH

Software development organizations face multiple ethical issues in the quality assurance function, especially in terms of the testing of their software products. The highest ethical standard calls for complete testing of all software and openness in sharing the results of the testing along with defect (bug) reports. Testing cost may be high, especially to try to meet the "complete" testing goal. All would also agree that the quality of the testing should relate to criticality of the use of the software.

The ethical issues are faced by all concerned. The tester has an ethical responsibility to perform complete testing of all aspects of the system. The business has the ethical responsibility to provide the resources and time needed for the testing team to complete the appropriate tests. The user has the ethical responsibility to use applications as they were intended, since unintended applications may assume testing that had not been done.

This paper has suggested the need for further research in a number of areas. It would be helpful to have a software criticality scale. If the quality of the testing is dependent on the use of the software, then it would be helpful to identify, on some scale, the criticality of the application. How could one then define levels of testing that would correspond to the levels of criticality?

Further research is also needed to try to develop a scale to measure the completeness of the software testing. This could be divided into testing of the functional and non-functional specification of the system. What is sufficient testing? How does an organization portray how much testing was done?

A high level of ethical software testing standard would suggest that known defects should be shared openly. Such openness would allow a customer to evaluate the potential impact of these known bugs. A system for classifying known defects according to severity level and affected application area would be helpful.

REFERENCES

Birsch, D., (2005). Moral responsibility for harm caused by computer system failures. *Ethics and Information Technology*, 6, 233–245.

Leveson, N. & Turner, C., (1993). An Investigation of the Therac-25 Accidents. *IEEE Computer*, 26 (7), July , 18-41.

Nindel-Edwards, J. & Steinke, G., (2007). The Development of a Thorough Test Plan in the Analysis Phase leading to more Successful Software Development Projects. *Journal of International Technology and Information Management 16* (1), 65-72.

Qureshi, S., (2001). How Practical is a Code of Ethics for Software Engineers Interested in Quality? *Software Quality Journal*, 9, 153-159.

Rose, G., (2003). Safety Critical Software. *CompactPCISystems*, April 2003, http://www.compactpci-systems.com/PDFs/Lynux.Apr03.pdf

Schneier, B., (2005). Liabilities and Software Vulnerabilities. http://www.schneier.com/blog/archives/2005/10/liabilities_and.html

Schneier, B., (2007). Information Security and Externalities. *ENISA Quarterly*, 2 (4), Jan 2007, 3. http://www.enisa.europa.eu/doc/pdf/publications/enisa_quarterly_01_07.pdf

Takanen, A., Vuorijarvi, P., Laakso, M., & Roning, J., (2004). Agents of Responsibility in Software Vulnerability Processes. *Ethics and Information Technology*, 6, 93–110.

Weckert, J., (2001). Computer Ethics: Future Directions, *Ethics and Information Technology*, 3, 93–96.