NEW SQM (SOFTWARE QUALITY MANAGEMENT) METHODS, TOOLS RAISE THE BAR FOR SOFTWARE DEVELOPERS

By: Geoffrey T. Hervey
Partner
Bregman, Berbert, Schwartz & Gilday, LLC
Bethesda, Maryland

The following article originally appeared in the April, 2003 issue of Software Quality Management Magazine, Vol. 3, No. 2 , http://www.sqmmagazine.com/issues/2003-02/sqmmethods.html.

In the January 2003 issue of Software Quality Management Magazine (Vol. 3, No. 1, “Building Secure Software: Better than Protecting Bad Software,” http://www.sqmmagazine.com/issues/2003-01/bss.html), Gary McGraw, Ph.D., described the benefits of "software security" over "application security." As Dr. McGraw explained the differences, application security employs standard approaches such as penetrate-and-patch and input filtering, and is primarily based on finding and fixing known security problems only after they are exploited in fielded systems. Software security, on the other hand, involves the process of designing, building and testing software for security so that it will withstand attack and exploitation in the first place: an admittedly harder task. Other approaches, including extreme programming (XP) and the use of unified modeling language (UML) models are also being debated as potential cures for software's ills.

Let's assume that one of these new approaches (or something like one or more of them) catches on. Would that have any real-world legal implications for the software development industry? In a real sense, the answer is "yes." If, in fact, a new method is better and does produce better software, sooner or later developers will have to embrace it or be prepared to explain why they did not. This is because a developer who chooses to ignore new methods that are embraced by others may face legal liability. (This is separate and apart from the commercial forces that will drive developers to employ such concepts.)

While software developers do not have to leap on every fad that comes along or live on the bleeding edge of innovation, they do have to be sensitive to trends that mark a shift in standard industry practices and certain practices that are not yet standard but will be or should be. Why? Because a developer who produces a product that contains problems, such as security vulnerabilities, may be accused of negligence if the developer's production practices fall below standard industry practices and if the problems could have been avoided by using standard industry practices. In this regard, a "standard industry practice" could, in fact, be a relatively new one.

Establishing Negligence: The "Duty of Care" Test

Here is how it could work. Negligence is a legal theory under which one person may be held liable for causing an injury to another person if he breached a duty of care that he owed to that person and damages result. For example, a doctor owes her patient a duty to exercise good medical practices and judgment in diagnosing and treating an ailment. If the doctor fails to diagnose a certain problem because she failed to use practices (e.g., a certain diagnostic test for cancer) that would have been used by a reasonably competent doctor, and the patient is harmed because of that failure (e.g., the cancer is not treated and spreads), the doctor could be liable for negligence. Whether the duty of care was breached depends on what a reasonably prudent doctor would have done under those same circumstances. If such a doctor would have used a specific blood test or diagnostic procedure that our doctor failed to use, and if the disease would have been properly diagnosed if that test or procedure had been used, our doctor could be found to have breached the standard of care and may be found negligent and liable for damages.

Applying this concept to software developers is not as straightforward. For one thing, a negligence action can be difficult to assert if the matter is governed by a contract, such as a software licensing agreement. Second, most well-written licensing agreements will contain numerous clauses excluding liability for negligence and limiting damages. (These issues were discussed in the January 2002 and July 2002 issues of SQM Magazine; see http://www.sqmmagazine.com/issues/2002-01/liability.html and http://www.sqmmagazine.com/issues/2002-03/ucita.html.) However, a determined plaintiff may be able to overcome such obstacles and a negligence action could be maintained against software developers even when a contract is in place. Moreover, a plaintiff could argue that the developer breached the contract by failing to employ industry standard practices. In addition, many software development contracts contain warranty clauses that the developer will use commercially reasonable procedures – that is, will comply with the standard of care – in developing the software.

If a plaintiff manages to launch a negligence claim or a claim for breach of contract or warranty against a developer for damages allegedly caused by problems with the software, a significant issue will be whether the developer complied with the standard of care in developing the program. Neither the judge nor the jury will be competent to determine what the standard of care is in this regard. Therefore, the plaintiff and the defendant developer will each have their own competing "expert witnesses" to opine as to what the standard of care is and what it requires. In this regard, a defendant developer can count on the fact that the plaintiff will present an expert who will testify that the standard of care required the developer to employ some strict SQM practice(s) that would have prevented the program from being released with the particular problem at issue. The plaintiff's expert must be careful not to go too far, because he will be made to look foolish on cross-examination if no one (or only a handful of developers) actually uses the SQM method described. If, however, the expert can convince the jury that a reasonable developer would have used the method, and our developer did not, our developer will be in trouble.

SQM and Raising Software Development Standards

Another example from the medical world may help readers better understand the notion of negligence and how developing standards can create a potential for liability where none previously existed. When MRI and CAT scans first came out, they were experimental, expensive and not available to most doctors. A doctor who failed to order such a scan in the early years would almost certainly not have been in breach of the standard of care at that time. Now, however, such scans are routine. Today, a doctor with reasonable access to such equipment may be found to have been negligent for not using it to diagnose certain symptoms. The introduction of these devices, therefore, raised the standard of care for the entire medical community to the point where what was once a cutting-edge notion is now a mainstream requirement.

Similarly, the adoption of new SQM methods will raise the standard of care for the entire software development industry. Moreover, this will become particularly the case as the industry matures and as the adverse consequences of "bad software" become even more obvious and foreseeable. For example, no one can dispute that enormous amounts of valuable data (e.g., credit card and social security numbers, medical histories, design data for the country's critical infrastructure) are entrusted to data and security applications. It is also clear today that such data is a ripe target for hackers, criminals, terrorists and others, and that such nefarious people will exploit any defects in guardian software to get at that data. Soon, it will simply be unacceptable to release software with security vulnerabilities, especially with privacy concerns growing all the time. Moreover, if huge damages occur, irreplaceable information is lost, identities are stolen, or personal injury or other damage results, the techniques used by the developer who produced the software will be placed under a microscope in an effort to see whether that developer could have avoided the problem by using tighter development procedures, such as the SQM methods being proffered today. This will happen regardless of whether a contract exists and what the terms of that contract may say. While the largest developers may have the resources to litigate such cases and survive an adverse decision, most developers do not have that luxury and must be more careful.

Additionally, developers need to pay attention to these new methods because, regardless of any potential liability for negligence that the developers face, their customers are getting more sophisticated and demanding. Savvy customers will monitor the industry and will begin demanding statements and warranties in their contracts to the effect that the product was developed using SQM methods. Failure to use such methods will expose the developer to liability for breach of warranty under the contract (or maybe even to a claim of fraud) even if the plaintiff might be unable to bring a claim for negligence. In addition, insurance companies that write business policies for software developers may, in the not-too-distant future, require adherence to certain SQM standards as a condition of coverage. Insurance carriers might even remain quiet about such matters until a claim is made, at which time the carrier may raise the developer's failure to adhere to a popular SQM standard as a basis for denying coverage.

To make matters worse, certain methods being discussed these days as part of an overall SQM solution, such as cost estimating and risk-management tools and techniques, must be used with care. These tools allow developers to make cost-benefit analyses of risks and help them decide which development corners to cut and by how much. Invaluable business tools, no doubt, but the very thought of them would make a defense lawyer cringe. I can imagine the plaintiff's lawyer asking at trial: "So, Mr. Developer, you used this fancy risk-management process to decide that it was okay to take these development shortcuts that resulted in a security vulnerability that allowed my client's competitor to steal my client's $70 million secret formula, correct? You decided that it was a risk worth taking, isn't that right?"

A Delicate Balancing Act

The reality is that you're damned if you do, damned if you don't. Clearly, SQM is a balancing act, but the point is that you won't be able to maintain your balance at all if you don't know what SQM methods are being used in the industry.

Let's not get ahead of ourselves. The point of this article is not to suggest that developers should rush to deploy the latest risk-management fad or development process. Far from it: leaping to an untested method or tool could be as bad as failing to employ an established standard. However, new ideas and methods cannot be dismissed out of hand, either. The point, rather, is to remind and encourage readers to keep an eye on changes in the industry and to evaluate fairly new SQM ideas and methods. Software developers cannot afford to be complacent in their development methods if a better method is reasonably available. Pay attention to what is happening in the industry: your customers will demand it... and a jury could punish you if you don't.