

Defective Software Works

Watts S. Humphrey

Over the years, many people have written to me with questions about software quality, testing, and process improvement. Jon Hirota asked how to get organizations to invest in software quality; John Fox asked if I see a movement away from system test and toward quality processes; Bob Schaefer wondered what I thought would happen in the area of software integration and testing; Dan St. Andre asked what software development managers can do to encourage executive management to meaningfully address software quality; and Pete Lakey wondered if and how the software community should use statistical process control techniques. I won't directly answer all of these questions but I will discuss these quality and testing issues here and in my next column. While it has taken me an embarrassingly long time to respond to these letters, they still raise a critical question: "How important is software quality and how should quality practices change in the future?"

First, What Do We Mean by Quality?

While the classical definition of product quality must focus on the customer's needs, in this and the next column, I will concentrate on only the defect aspect of quality. This is because the cost and time spent in removing software defects currently consumes such a large proportion of our efforts that it overwhelms everything else, often even reducing our ability to meet functional needs. To make meaningful improvements in security, usability, maintainability, productivity, predictability, quality, and almost any other "–ility," we must reduce the defect problem to manageable proportions. Only then can we devote sufficient resources to other aspects of quality.

The functional and operational characteristics of a software product should and will continue to be important, but that is where most people now focus, and there is little risk that they won't continue to do so in the future. If a product doesn't have attractive functional content, it won't sell, regardless of how few or how many defects it contains. Unfortunately, many software groups treat the importance of functional quality as an excuse to concentrate almost exclusively on product function and devote little attention to better managing the defect problem.

While there is irrefutable evidence that the current "fix-it-later" approach to defect management is costly, time consuming, and ineffective, don't expect this to change soon. It is too deeply ingrained in our culture to be rooted out easily. However, since I'm an optimist, I'll keep trying to change the way the world views software quality. And by that, I mean the way we manage defects.

Second, How Important is Software Quality?

The key question is: "Important to whom?" Developers are necessarily preoccupied with defects. They spend the bulk of their time trying to get their products to work. And then, even when the products do work, the developers spend even more time fixing test and user-reported problems. While few developers

recognize that their schedule and cost problems are caused by poor quality practices, an increasing number do. This is a hopeful sign for the future.

Not surprisingly, development management tends to view quality as important only when executives do. And the executives view quality as important only when their customers do. When customers demand quality, executives almost always pay attention. While their actions may not always be effective from a quality-management perspective, they will almost always respond to customer pressure. And even if their customers do not demand improved quality, government regulation can cause both executives and development managers to pay more attention to quality. This is true for commercial aircraft, nuclear power plants, and medical devices. There is little question that, with commercial aircraft for example, the close and continuous regulatory scrutiny coupled with painstaking reviews of every safety incident hold this industry to high quality standards.

Third, Why Don't Customers Care About Quality?

The simple answer is: "Because defective software works." The reason it works, however, is because software doesn't wear out, rot, or otherwise deteriorate. Once it is fixed, it will continue to work as long as it is used in precisely the same way. But, as software systems support an increasing percentage of the national infrastructure, they will be used in progressively less predictable ways. When coupled with the explosive growth of the Internet and the resulting exposure to hackers, criminals, and terrorists, the need for reliable, dependable, and secure software systems will steadily increase. If experience is any guide, as these systems are used to perform more critical functions, they will get more complex and less reliable. Unfortunately, this probably means that it will take a severe, disruptive, and highly public software failure to get people concerned about software quality.

Two forces could change this complacent attitude. One is the Sarbanes-Oxley Act, which makes chief executives and chief financial officers personally responsible for the quality of their organizations' financial reports. This has caused executives to inquire into the accuracy of their financial reporting systems. What they find is often disquieting. The general accuracy of such systems is usually reasonably good, but there are many sources of error. While these errors have not been a serious concern in the past, they will become much more important when the senior executives are personally liable.

The second issue is closely linked to the first. That concerns software security. Although this is not yet well recognized, when software systems are defective, they cannot be secure. Executives are also just beginning to realize that software security is important to them because, if their systems are not secure, they almost certainly cannot be accurate or reliable. Since they are now personally liable for the accuracy of their financial systems, they now are beginning to appreciate the need for secure financial systems.

How Defective is Software?

So the basic question concerns defects. First, some facts. The number of defects in delivered software products is typically measured in defects per thousand lines of code (KLOC). Figure 1 shows some data on recent history. Here, Capers Jones has substantial data on delivered product defect levels, and he has

compared these with the CMM[®] maturity of the organizations that developed the software [1]. Noopur Davis has converted the Jones data to defects per million lines of code (MLOC), as shown in Figure 1 [2]. For example, organizations at CMM level 1 delivered systems with an average of 7,500 defects per MLOC while those at level 5 averaged 1,050 defects per MLOC. What is disquieting about these numbers is that today most products of any sophistication have many thousands and often millions of lines of code. So, while one defect per KLOC may seem like a pretty good quality level, a one million LOC system of this quality would have 1,000 defects. And these are just the functional defects.

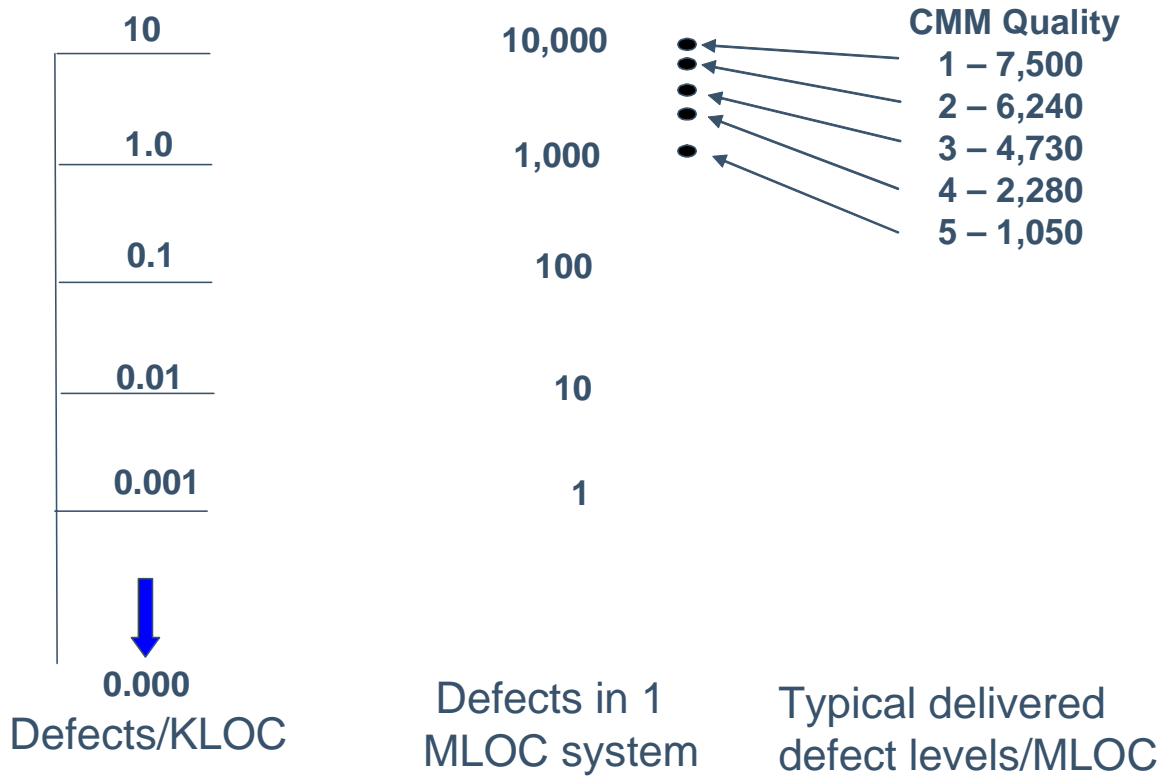


Figure 1. Typical Software Quality Levels – in Delivered Defects

Today, most programmers are unaware of many types of security defects. While some security defects are generally recognized as functional defects, others are more subtle. A security defect is any design error that permits hackers, criminals, or terrorists to obtain unauthorized access or use of a software system. Since many of these defects do not cause functional problems, systems that are riddled with security flaws may work just fine and even pass all of their functional tests. And, since many programmers are worried only about getting their programs to function, they are almost totally unaware of their products' potential security vulnerabilities.

[®] CMM is registered with the U.S. Patent and Trademark Office by Carnegie Mellon University.

In one program that had been supporting a Web site for over two years, a security audit found one functional defect and 16 security defects. So, today's one MLOC systems with 1,000 functional defects could easily have many thousands of security defects, and almost any of these could provide a portal for a hacker, thief, or terrorist to steal money, disrupt records, or to otherwise compromise the accuracy of the organization's financial system. No wonder executives are worried about the Sarbanes-Oxley Act.

Putting Software Quality into Perspective

At present, over 90% of all security vulnerabilities are garden-variety functional defects. This means that our initial goal must be to reduce functional defect levels. From a security perspective, one defect per KLOC is totally inadequate for large systems. But what does one defect per KLOC mean in human terms? One thousand lines of source code, when printed in a listing, typically take about 30 to 40 printed pages. This means that one defect in 30 to 40 printed pages is poor quality. No other human-produced product comes close to this quality level.

But even though one defect per KLOC is far beyond the levels that humans ordinarily achieve, this quality level is totally inadequate. At this quality level, most large systems will contain many thousands of defects, and any one of them could open the door to security problems. So, if 1,000 defects in a one MLOC (million line-of-code) system is inadequate, what would be adequate? That, of course, is impossible to say, but 10 defects per MLOC would be an advance from where we are now. However, I am afraid that even 10 defects per MLOC will not be adequate forever.

Reaching a level of 10 defects per MLOC would mean only one defect in 3,000 to 4,000 pages of listings. That goal seems preposterous. Are such quality levels needed, and is there any hope that we could achieve them? Unfortunately, we almost certainly need such levels today. Criminals, hackers, and terrorists now have and will continue to devise more sophisticated and automated ways to probe our systems for security vulnerabilities. And any single vulnerability could open the door to serious problems.

The problem is not that 10 defects per MLOC is difficult to achieve. The real problem is that, even if it is an adequate security level today, it will almost certainly not be adequate for very long. In the next column I will discuss this problem and explore some of the issues we face in producing systems that have such extraordinary quality levels.

Acknowledgements

In writing papers and columns, I make a practice of asking associates to review early drafts. For this column, I particularly appreciate the helpful comments and suggestions of Julia Mullaney, Bill Peterson, Marsha Pomeroy-Huff, and Carol Woody. The letters from John Fox, John Hirota, Pete Lakey, Bob Schaefer, and Dan St. Andre were also very helpful and I thank them as well.

In closing, an invitation to readers

In these columns, I discuss software issues and the impact of quality and process on developers and their organizations. However, I am most interested in addressing the issues that you feel are important. So, please drop me a note with your comments, questions, or suggestions. I will read your notes and consider them when planning future columns.

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

References

[1] Jones, C. *Software Assessments, Benchmarks, and Best Practices*. Reading, MA: Addison Wesley, 2000.

[2] Davis, N. & Mullaney, J. *The Team Software Process (TSP) in Practice: A Summary of Recent Results* (CMU/SEI-2003-TR-014). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003 < <http://www.sei.cmu.edu/publications/documents/03.reports/03tr014.html?si>>.

About the Author

Watts S. Humphrey founded the Software Process Program at the SEI. He is a fellow of the institute and is a research scientist on its staff. From 1959 to 1986, he was associated with IBM Corporation, where he was director of programming quality and process. His publications include many technical papers and several books. His most recent books are *Introduction to the Team Software Process* (2000) and *Winning With Software: An Executive Strategy* (2002). He holds five U.S. patents. He is a member of the Association for Computing Machinery, a fellow of the Institute for Electrical and Electronics Engineers, and a past member of the Malcolm Baldrige National Quality Award Board of Examiners. He holds a BS in physics from the University of Chicago, an MS in physics from the Illinois Institute of Technology, and an MBA from the University of Chicago.

The views expressed in this article are the author's only and do not represent directly or imply any official position or view of the Software Engineering Institute or Carnegie Mellon University. This article is intended to stimulate further discussion about this topic.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.