

Watts New

## The Quality Attitude

Watts S. Humphrey

This is the third in a series of columns on software quality and security. The first column, “Defective Software Works,” discussed the software quality problem, why customers don’t care about quality, and how bad software quality really is. The conclusion was that even defective software is of higher quality than just about any other humanly produced product. The second column, entitled “Security Changes Everything,” described why testing alone cannot produce quality software. It explained that defective software cannot be secure, and it described the need to improve quality by at least 100 times over where we are today.

This column discusses the quality attitude and how software professionals and their managers must change their view of quality if we are to make much headway with software quality and software security. Quality is key to software performance, whether the concern is with function, usability, reliability, security, or anything else. In the next column, I will outline a strategy for solving these problems and describe steps that software professionals and their management can take.

### The Testing Attitude

For as long as I have been writing programs, programmers have believed that when they clean up their programs in test, the programs will work. While this is often true, it isn’t always true. This difference is the source of many of our software quality and security problems.

Our programs are often used in unanticipated ways and it is impossible to test even fairly small programs in every way that they could possibly be used. To appreciate the testing problem, consider the simple maze in Figure 1. The highlighted corners are possible branches and an example path from corner A to corner B is indicated by the arrows. Considering only the forward-going paths and ignoring loops, there are 252 possible paths from corner A to corner B in this 5 by 5 matrix. This matrix has only 25 branches. For 100 branches, we would need a 10 by 10 matrix, which would have 184,756 possible forward-going paths.

Most useful programs have many more than 100 branch instructions. For example, one of my C++ programs has 996 lines of code (LOC) and 104 branches. This is about one branch for every ten instructions. Even if large programs only had ten branches per thousand lines of code (KLOC), a 1,000,000 LOC program would have 10,000 branches. For a 100 by 100 square matrix with 10,000 branches, there are an astronomical  $2.28 * 10^{58}$  possible forward-going paths from corner A to corner B. Few developers are even willing to run 252 tests, let alone  $2.28 * 10^{58}$  tests.

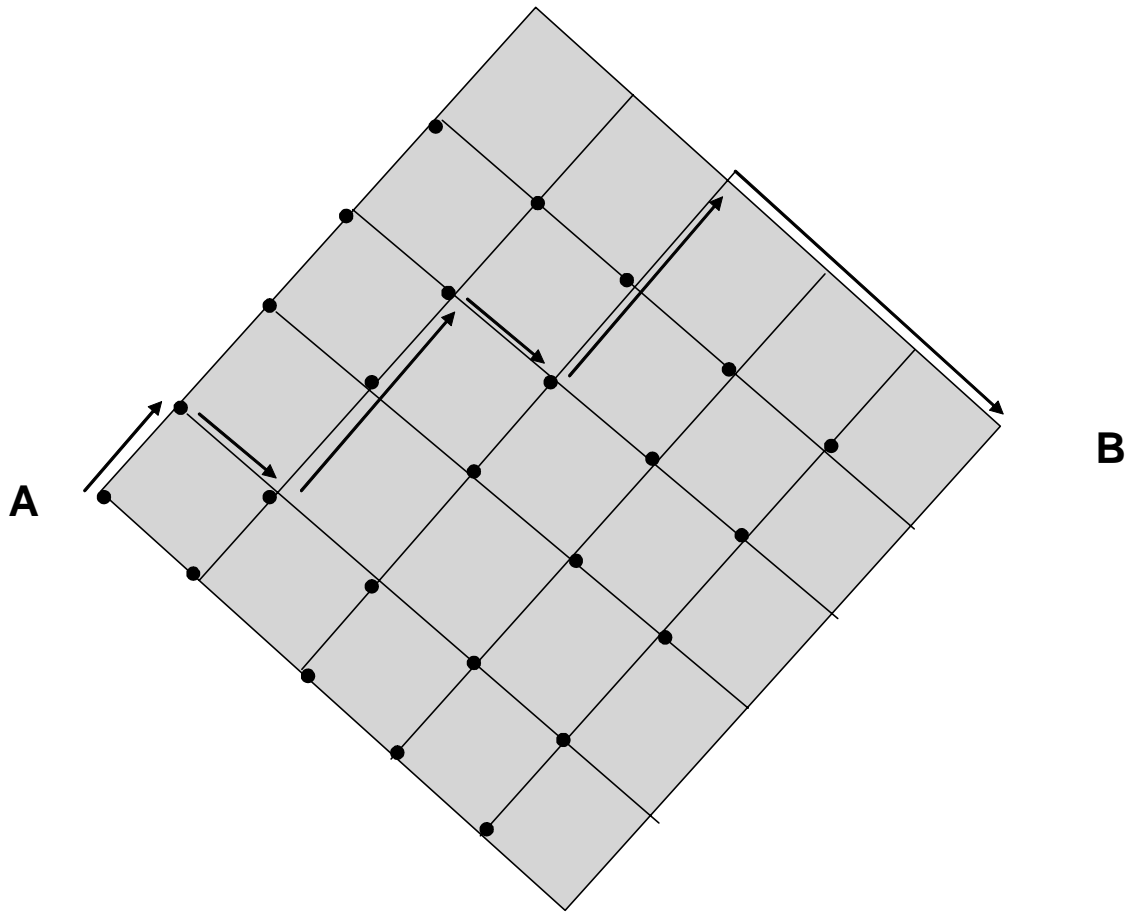


Figure 1. Possible Paths Through a Matrix

A brief analysis shows that testing this number of paths is impossible. For example, if you simultaneously ran a million tests a second for 24 hours a day on a million computers, it would take longer than the lifetime of the universe to run all of these tests. Even then, you would only have tested one set of data values.

Since it is impossible to exhaustively test large programs, we face the next question: “Is this much testing really necessary?” To answer this question, we must consider how many defects there are in typical large software systems. We now know how many defects experienced software developers inject. On average, they inject a defect about every ten lines of code. An analysis of data on more than 8,000 programs written by 810 industrial software developers is shown in Table 1.

The average injection rate for these developers is 120 defects per KLOC, or one defect in every eight lines of code. Even the top 10% of the developers injected 29 defects/KLOC and the top 1% injected 11 defects/KLOC. Even at the injection rate for the top 1% of software developers, a 1,000,000 LOC system would enter compiling and testing with 11,000 defects. More typical developers would have 120,000 defects in their products.

*Table 1. Defect Injection Rates for 810 Experienced Software Developers*

<b>Group</b>	<b>Average Defects per KLOC</b>
All	120.8
Upper Quartile	61.9
Upper 10%	28.9
Upper 1%	11.2

That is why most large programs, even after compiling and testing, have from 10 to 20 defects per KLOC when they enter system testing. For a 1,000,000 LOC system, that would be between 10,000 and 20,000 defects. With current practices, large software systems are riddled with defects, and many of these defects cannot be found even by the most extensive testing. Clearly, while testing is essential, it alone cannot provide the quality we need to have secure systems.

So, the first required attitude change for software professionals and their managers is to accept the fact that testing alone will not produce quality software systems. Also, since defective software cannot be secure, testing alone won't produce secure systems either.

### **The Defeatist Attitude**

The next attitude that we must change concerns the feasibility of producing secure and high-quality software. Many software experts will tell you that there is no such thing as defect free software. Their obvious conclusion is that we must learn to live with poor-quality and insecure software products. While the previous discussion of testing may have convinced you that this attitude is correct, it is not. What these professionals are really saying is that there is no way to prove that a software system is defect free.

Unfortunately, it is true that there is no way to prove that a software system is defect free. But that is also true for every other kind of product. There is no way to prove that an automobile or a jet airplane or any other product is defect free, but we don't accept that as an excuse for poor quality automobiles or jet aircraft. The question is not to prove that a software product is defect free, but rather to prove that we have taken all practical steps and used all available methods to make the product as close to defect free as possible. There are many ways to do this, and they have been proven highly effective in many other technical fields. While the methods are simple, they are not easy. They require measurement, analysis, a lot of personal attention, and, above all, a conviction that quality is absolutely essential.

The current common view that it is impossible to produce defect free software is an excuse for not making the effort to do quality work. A more subtle form of this attitude is relying exclusively on tools to identify errors. How many times has some programmer told you that a new environment, language, debugger, or analyzer is the answer to the quality or security problem? But then, even after using all of these fancy new gadgets, the resulting products still have defects and are still insecure. The problem is attitude, and no one who counts on some tool or gadget to handle quality will ever produce large, secure software products.

## **The Quality Attitude**

Having the right attitude can make an enormous difference. There is growing evidence that defect free software is possible. Some development groups are now producing reasonably large-scale software products that have had no defects found by their users. While these products actually may have latent defects, for all practical purposes, they are defect free. Today, a few development teams can consistently produce such software.

## **The Challenge Ahead**

We need to learn from and extend these proven quality methods and practices to our highest priority software needs. Today's critical need is to apply these proven quality practices to the really complex and interconnected systems that support the nation's critical infrastructure.

What should be clear from this discussion is that we have not tried hard enough to produce defect free software, so we really cannot know if such high-quality products are possible or not. The next column will discuss the quality methods that people are using today to produce exceptionally high-quality software and the steps required to extend these practices to the truly massive systems that will be common in the future. These systems must be secure and, to be secure, they must be essentially defect free. Learning how to consistently produce such systems is our challenge for the future.

## 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 Tim Morrow, Julia Mullaney, Bill Peterson, Marsha Pomeroy-Huff, Alan Willett, and Carol Woody.

## 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

## 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.