

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

LESSONS FROM THE TRENCHES

George Martin

Defects For Sale

Although it might not be the best slogan to put on your next batch of business cards, if you design custom hardware and software, defects (bugs, failures, etc.) are a reality of every project. According to George, the key to staying in business is understanding your ability to “design”, find, and repair defects.



Imagine a riddle like the ones you come across in a Sherlock Holmes story. It might go something like this:

You don't want any but, the more you get the cheaper they are while the more you have the costlier they become.

What are we talking about? Defects of course. I prefer the term defect instead of failures or bugs. The latter definitions don't convey the overall picture of what's involved in a defect. But you know what I'm talking about. These defects are the things that keep you up late at night and make you lose your hair.

What is a defect? Let's define a defect as an operation that is different than the customer expects. For example, if your software is required to calculate an average value, and you total the readings and then divide that total by the number of readings, -3,

we would all agree that the math is just plain wrong. The defect is in the math. If however, we provide the mathematically correct answer but with no decimal digits, and the specification didn't specify the number of decimal places, but the customer wants two decimal digits, then it's still a defect. This defect has a different cause. But it's still a defect. The bottom line is that defects make for unhappy customers. Some of the sources of defects are coding errors, design errors, specification errors, or customer errors.

Now, you may think that I have taken an unrealistic point of view. It's got to be more complicated. Well, perhaps it is, but if you can't keep it simple you will never get to the bottom of your problems and attempt to fix the issues.

There have been lots of articles about the number of defects found in Windows 95/98. The most interesting comment I've read was that the defect level was the same as IBM's OS360 and that's the level that makes it a commercial success. NASA or the medical industry could not accept the same defect rate as a commercial OS. Have you come across any of these defects. Are you unhappy with them. Unhappy enough to switch OSs? Could you have the same defect rate in your product and stay in business?

This brings us to my second point. What is your defect rate? Do you know? Why not! Mine is 2-6 defects per typical project. Is that a lot? How do I know? How do I ever get customers to pay for my work if it has defects?

Let me explain. First, my typical project is an embedded computer running custom hardware and software. It's real-time, interrupt driven, written in C (small amount of assembly to start up), and has 20-40 KB of

code. The routines are designed, coded, and tested without hardware as much as possible. Much of the code is derived from existing modules and not newly created. Typical math routines might include a Fourier transform or trigonometric functions.

Half my defects come from decisions I've made that are not really what the customer wants. I may have even discussed it with them but they weren't informed enough to make a valid reply. The other defects are just plain coding errors on my part. Some of them were created by changes in a set of requirements, but they are still defects.

My primary type of coding defect is an array index (or pointer) running past either end of the array. And, my second most popular type of coding (perhaps design) defect is state machine states or variables not used as designed, or having unanticipated side effects. Notice how I take ownership for these defects. Yes, they are all mine. I made them and I shipped them. Perhaps specification changes cultivated their growth, but if you don't take ownership then you'll never eliminate them.

My first type of defect is easy to test for and only requires careful design, coding, and testing to resolve. I build test routines that exercise the indexing and find all of the defects. The ones that get through come after major changes, usually as a result of customer feedback. I repeat the testing but don't spend the time necessary to find the new defects. Now we can ask, "Should I spend more time to find them?" This is perhaps a management question and a profitability issue. You will have to be the judge in this area. But, shipping late would also be unacceptable.

The second group of defects I can almost never test for and is usually more difficult to duplicate and find. The customer finds these because they have real-world inputs. With careful design, I can usually isolate the section of code or variable that's giving me the problem, but this class of defects is much harder to reproduce.

Which are your most common sources of defects? Do you know? What steps did you take last year, or last month to eliminate their occurrences?

If you're building embedded systems, then you probably can't have any defects. And, any defect reported is corrected and that fix is included in the next release of software. I believe that if you start keeping defect records, you will be able to detect more defects in design and testing, release software with fewer defects, and respond to field problems more rapidly.

How did I get to this topic? Well I'm working on an embedded '486 and it has a battery-backed real-time clock. I'm writing the code to let the user enter the time and date, which brings me right smack dab in the middle of the Y2K issue. I was hoping to stick my head in the sand until January, then look up. The clock chip is supposedly Y2K compliant. Valid dates are 1999 to 2098. What happens in 2099? Well, we then have a Y2K++ problem. I could also force the user to enter four digits for the date, so 1999, 2000, and 2001 are valid. I'm defining 1998 and 2099 as invalid dates. Should I let the user enter 99 or 00?

I soon came to the conclusion that no matter which rules I choose for date entry, I will probably have built in a defect. If I force the four-digit year, then I get less confusion but more typing for the user. If I permit two or four digits, then I have to use some deductive logic and I expect all cases might not be covered correctly. Also, the code is starting to get complicated. So, I don't know what I'll do. I have another couple of days to let the dust settle. Maybe I can delay until next year. This is beginning to sound like a Dilbert cartoon. ☹

George started his career in the aerospace industry in 1969. After five years at a real job, he set out on his own and cofounded a design and manufacturing firm. Typical systems that George designs include servo-motion control, graphical input and output, data acquisition, and remote control. George is a charter member of the Ciarcia Design Works Team

and most recently, he's been working on the people-tracking system for Bill Gates's new house. You can reach him at george.martin@worldnet.att.net.

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, subscribe@circuitcellar.com or www.circuitcellar.com/subscribe.htm.