When Failure *IS* an Option …

I'm sitting here writing this BACKTALK while on business in Baltimore. I flew into the Baltimore Washington International Airport (BWI). I remember a few years ago when BWI advertised itself as a great alternative to both Reagan National (in downtown D.C., always crowded) and Dulles (which is about 25 miles out of D.C.). BWI was convenient to both Baltimore and D.C., and small enough that rental cars were located within a five minute walk of the terminal. Things have changed! BWI is now under construction, and I had to walk from one end of the airport to the other to get to baggage claim. Then, I had to walk all the way to the other end of the airport to catch the "rental car bus." Car rentals are now about five miles away, so you have no option other than the inconveniently located rental car bus – and the buses were extremely crowded. Things change. What used to be a *good thing* becomes inconvenient and appears to be poorly designed.

Which brings us to "Why Software Fails." This has been a hard column to write – I was tempted to take the easy way out, and simply list and add the pictures of a few former co-workers and acquaintances who, in my opinion, have contributed to failing software over the years.[1] But instead, I have come up with a good start at a list that explains why software fails.

Software fails … one day at a time. Insidious little events occur. Small errors creep in. You have the *road not taken* syndrome. You realize that you could do better, but you don't have time to go back and start all over again. It's not always the big errors that cause failure, it's the little errors that accumulate.

Software fails … with the best of intentions. Developers, with the exception of a few TRULY unspectacular folks I have known, don't really set out to do a poor job.[2] We try and make the right choices, but we don't have the ability to predict the future. If things had turned out a little differently, we would have had a spectacular success. Instead, decisions turned out to be sub-optimal. If we only had the time to do it over.

Software fails … because we have no other choice. Sometimes politics, budgets, and schedules force us to make decisions we don't like. In a perfect world, we would have the time and budget to make perfect software. The world isn't perfect, and we are often forced into less than perfect solutions. We know better, we just can't do better. Real-world requirements change and we have to make the software react also, or the software becomes obsolete. I am relatively sure that the designers of the new BWI rental car terminal wish they were still located within a few hundred feet of the airport. However, the reality is that increased air traffic and congestion made this option infeasible. Sad to see it go, but it beats NOT having a rental car, doesn't it?

Software fails ... because we are overcome by events. Sometimes, we have to make choices before we have time to research all of the options. Schedules are tight and it's more important to make a workable decision now rather than making a better decision later. We don't like it, but it's just what we have to do.

Software fails … because we can't think of everything. Ever left the house for the grocery store with a *memorized* grocery list? You started out for milk and eggs. You added carrots and sliced cheese. Your spouse reminded you that you need toothpaste and shampoo. Only a few items. Yet, by the time you get to the grocery store, you're reduced to calling home, because all you can remember is milk, eggs, and *something else*. How many things can you juggle in your memory at one time? For most people, I suspect this number peaks out around nine or 10. Unfortunately, large-scale software has millions of lines of code, and literally tens of thousands of function points. How can we comprehend such large scale? We use architectural design to decompose the problem, and we use high-level languages, modularity, and object-oriented techniques to further break the problem down. However, with software of such large size, things just slip through the cracks. Requirements are missed or not implemented. Obvious errors are usually obvious only in hindsight – after the failure.

Software fails … because developers are only human. Have you ever spent hours (or even days) looking for an error, and had somebody wander by, glance over your code, and immediately see the problem? When you develop code, you tend to internalize your own errors, and then your brain fails to see them. An outside observer, however, can often see what you keep overlooking. Almost all good developers know that you need somebody else to review your work. This applies to all phases of software development: requirements, design, coding, and maintenance. One of my favorite quotes is, "When quality is vital, independent checks are necessary, not because people are untrustworthy but because they are human[3]." Even if you are one of the best software developers around,[4] you make mistakes. So does everybody else.

Software fails … because you failed to consult the Software Technology Support Center (STSC) for help when developing your software. Or, if not the STSC, you should learn from *somebody*! You want to emulate the best practices of others while at the same time keep from making the same mistakes that others have made. Learn from the mistakes of others and also learn from the success of others. Find out what other similar development efforts did right and wrong. Read journals. Talk to fellow developers on other projects. But then – you *are* already reading CROSSTALK, aren't you?

— David A. Cook, Ph.D.
The AEgis Technologies Group, Inc.
dcook@aegistg.com

P.S. I am not claiming my list is complete or even valid (after all, I didn't review this with anybody else!). Feel free to e-mail me your additions or comments, and maybe you'll see them in a future BACKTALK column.

Notes

1. Worried that you'll find your name here, aren't you?
2. STILL worried that you'll find your name here, aren't you?
3. Humphrey, Watts S. Managing the Software Process. Addison-Wesley, 1989.
4. Well, you certainly aren't expecting to find any name other than mine, are you?