Coding Therapy for Software Developers

Author: Steven Feuerstein, steven.feuerstein@quest.com

www.ToadWorld.com/SF
www.PLSQLChallenge.com
www.StevenFeuerstein.com

This session drives home the following critical points:

- Software is written for the most part by human beings.
- Human physiology *and* psychology will, therefore, have a very powerful effect on the way we write code.
- And just like you might find real therapy helpful in your non-software lives, there is a very good chance that you will benefit from some "coding therapy."

This paper offers a small portion of what is covered in the presentation itself.

Don't act like a bird: admit weakness and ignorance. AKA: Ask for help (or at least take a break) after 30 minutes on a problem.

Problem: Steven is a hypocritical programmer.

I spend a lot of my time in public talking about best practices. In other words, I stand up in front of other developers and act "holier than thou," offering advice and admonitions along the lines of "Do this, don't do that, and certainly *never* do the other thing."

Occasionally, I am honest enough to point out that I do not always follow all my best practices. And students in my classes are, not infrequently, delighted to point out violations of best practices in the code I present to the class.

I do think that lots of my code is at least *reasonably* well-written. Sometimes, though, the way that I ignore my own recommendations is so over-the-top and painful that my hypocrisy is brought to the fore – and I feel compelled to make a confession. Back in June of 2007, I was in Europe presenting the Quest Code Tester product to developers, DBAs, and managers in Paris, Brussels, and Maidenhead (UK). When not in the public eye, I kept myself very busy debugging some of Quest Code Tester's backend code (on which I am the lead developer) as well as working on the second edition of *Oracle PL/SQL Best Practices* for O'Reilly Media (the book from which this content is drawn!).

Now, as I'm sure you all know, debugging code can be a frustrating and time-consuming adventure (even if you take advantage of Toad's great source code debugger). And this brings me to the source of my hypocrisy. On June 13, I sat in my room at the Holiday Inn in Maidenhead from 7 PM to midnight putting dents in the desk with my head. My problem? In Quest Code Tester, you can define dynamic test cases based on predefined groups of values or values retrieved via a query from a test data table. That wasn't the problem—in fact, that is a great feature. Unfortunately, the results of these tests (generated at test time) were not rolling up properly when the program being tested raised an unhandled exception. In other words, they were showing success when they should show failure, and vice-versa.

No doubt about it: the code that performed the rollup was complicated stuff, made more challenging by my reliance on a three-level, string-indexed, nested collection structure. Those things are not at all easy to debug. My debugging adventure also turned into one of those scenarios in which, as you fix one bug, you realize that the code you thought was working *by design* was actually working *by accident.* That is, bugs that were previously being masked were now exposed by other fixes. Wow, software can be incredibly complex!

So I would fix one bug and then find others, fix those, and then...it was 10 PM and I started to encounter behavior (duplicate result rows with different outcomes) that I simply could not explain. So I struggled for two more hours, eyes tired, back sore, thirsty, and increasingly *angry*, until I gave up and went to bed.

Solution: Give your brain a break, and ask others for help.

I woke up, six hours later, with an idea hopping around excitedly in my head: I could suddenly and very clearly see why I was getting duplicates, where it *must* be occurring in my code. I hurried to my laptop (easy: it was four feet away) and 10 minutes later confirmed my analysis. I fixed that bug, found another, analyzed it, fixed it. After 30 minutes, my code was working for all known test scenarios.

I sat back in my chair, a little bit stunned – excited, sure, at having found the solution, but also thoroughly disgusted at myself for wasting so much time the night before. How could I have done that? I knew better. But more than that, I regularly *preached* better. What a hypocrite!

So, this is what I learned (or was so painfully reminded of) from my terrible, horrible, no good, very bad [1] evening with the *qu_result_xp* package (and other similar experiences):

- *Apply the Thirty Minute Rule rigorously:* This rule really does work. Do *not* spend hours banging your head against the wall of your code. Ask for help. If your manager has not set up a process or fostered a culture that says it's OK to admit ignorance, you will have to do it yourself. This is especially important if you are (or are seen as) a senior developer on your team. Go to one of your junior team members and ask for help. They will be flattered, their self-esteem will increase, *and* they will help you to solve your problem.
- *Get help from anyone:* If you are stuck and cannot turn to another programmer for help, then ask a *nonprogrammer* for a sympathetic ear. My friend and Quest Code Tester codeveloper, Leonid, told me that he used to ask his grandmother (not known for her programming skills) to listen to him talk about his work. *Externalizing your thoughts*, even to someone ignorant of the content, helps *you* organize and clarify your thoughts.
- *Take a break:* If you are stuck and alone and cannot turn to another programmer—or any other human being—then STOP! Take a break. Get away from your work. Best alternative: get some exercise. Move your body. Go out for a walk or a run. Jump up and down, stretch, do sit-ups. Let your brain relax and make its connections. Suddenly, as if by a miracle, that incredible brain of yours will come up with a solution! And when you come back from your break, try talking out loud to yourself, describing the problem. Or take out a piece of paper and write it down. The key thing is to *get it outside of your head.* You will then find it easier to visualize the problem *and* the solution,
- *Watch out for irrationality:* You will know that you are past the point of productive work when you find yourself thinking in less than rational ways. I can still remember back in 1992 when I was building a debugger for SQL*Forms 3 on Oracle's brand-new PC implementation (the first software to use memory above the 640K limit!). I started to get runtime errors, and I discovered that if I added a tab character to the code, the location of the error would change. I sat there for hours trying to find the source of the problem (typing in extra spaces, returns, etc.), when OBVIOUSLY it was a bug down deep in Oracle. PL/SQL does *not* care about white space. When you find yourself saying "What my program is doing is impossible and makes no sense." you really should stop and take a break.

Beyond the actions of individual programmers, team leaders and development managers need to cultivate an environment in which we are encouraged to admit what we do not know, and to ask for help sooner rather than later. Ignorance isn't a problem unless it is hidden from view. And by asking for help, you also

[1] I draw that phrase from my days of reading books to my son, Eli. It's a reference to the book *Alexander and the Terrible, Horrible, No Good, Very Bad Day* by Judith Viorst. If you have small children and are not already familiar with this book, I recommend that you get it.

validate the knowledge and experience of others, building the overall self-esteem and confidence of the team.

To be very honest with you, I don't think I am all that great a programmer. I have a quick mind and am good at *communicating* ideas. But I need lots more discipline and patience as I write my code. And I need to apply my (and others') best practices more regularly and thoroughly.

So remember: do as I say, not as I do!

We need more than brains to write software.
AKA: Take care of your "host body": fingers, wrists, back, etc.

Science fiction is an awful lot of fun to read in books and to watch in the movies. It's amazing what computers and cyborgs and so on can accomplish when they are not constrained by the economic and technical realities encountered on Planet Earth.

Well, we not only live in the real world, we write software that attempts to mimic within cyberspace a small fraction of that real world. As the real world is always changing, there is always great pressure on us and our applications to "keep up." That results in great job security, but also tremendous challenges.

One fundamental reality we must keep in mind as we explore best practices for writing software is that our programs are written almost entirely by us, human beings. Sure, we can and should take advantage of the code-generation tools discussed earlier, but for the most part, software development doesn't happen without our sitting in front of a screen and typing.

The ramifications of this simple, undeniable fact are far-reaching:

- The software we write and how we go about writing it are affected greatly by the physiology (hard-wiring) of our brains, and the psychology of humans as we interact with each other and our environment.
- Our brains can handle only so much complexity at a time (though we certainly can train our gray matter to juggle larger and larger volumes of data and structures). We need to find ways to organize our activity so that we don't get overwhelmed, confused, or lost.
- We are lucky that we are able to make a living off the product of our brains. Our brains need our host bodies in order to function, so we need to make sure we take care of those hosts. Humans didn't evolve in order to sit in front of a monitor 6, 8, or 10 hours a day.

Here are my top recommendations for taking case of your host body. You can think of these as best practices for *you*, rather than your code:

- *Drink lots of water:* You probably hear this advice a lot. Without a doubt, you will generally be much healthier if you drink more water and less coffee, Coke, and other caffeinated, sugared products. But specifically when it comes to brain work, if you get dehydrated, it's like a car engine without enough oil. Your brain gets sluggish and dull. Here is my concrete suggestion: the next time you come back from a heavy lunch and find yourself nodding off at your desk, do *not* get a cup of coffee. Do *not* refill your cup with soda. Instead, drink down a large glass of water. You will immediately feel more awake, alive, and ready to take on your challenges.
- *Take lots of breaks, move around, get exercise, and go outside:* If your body becomes too sedentary, it will be hard for your brain to stay focused. Plus, if you don't move around, various parts of your body will start to decay and hurt. It's a basic case of "Use it or lose it." Of course, it's hard to think about any sort of vigorous exercise routine if your lower back hurts and your knees are cracking. So start simple: do some situps every day. You can do these without any special equipment, and as your abdomen gets stronger, your back will feel better. You will sit up straighter. You will feel much better about yourself. Finally, spend as much time as you can out of doors. Indoor air is usually stale, recycled, and unhealthy. Plus if you go outside, you might even feel the sun on your face. Small joys.
- *Make sure your workspace is comfortable and ergonomic:* Don't compromise mobility or range of motion for the sake of a few programs. I strongly recommend that you buy an ergonomic keyboard like

> Microsoft's Natural Keyboard. It will greatly reduce the strain on your wrists; believe me, you will adjust to its different key configuration in a matter of an hour. Make sure that your monitor, chair, and desk are properly aligned so you do not feel stress in your shoulders, neck, back, arms, or hands as you type. Don't rest your hands or wrists directly on hard desk surfaces. I recently discovered the IMAK wrist glove and now use it whenever I am away from my home office. It provides great support and comfort for my wrist.

Writing software is lots of fun, but remember to keep it in perspective: it's not worth sacrificing your physical health to write lots of exciting, cool code. And that sort of tradeoff is totally unnecessary.