Schadenfreude and Mature Software Development

Ever heard of the word "schadenfreude?" It is what is called a "loanword" entering the English language from German. It can best be described as that delicious feeling you get when you see somebody cut you off at a four-way stop, but then immediately run into something. Technically, it is defined as pleasure derived from the misfortunes of others?" It is a very interesting word—it has a diverse background (being found in many languages), and various studies have shown the feeling of schadenfreude is linked to envy, possibly linked to a particular sex (some studies show that men feel it more), and also possibly linked to a feeling of low self-esteem. However, I personally think that that many/most/all software practitioners get some delight as seeing (or reading) about a colossal software failure.

Way back in 1996, a very good friend of mine, then-Captain Thomas Schorsch (now Retired Lt. Col, Ph.D.) was a student at the Air Force Institute of Technology. He wrote an article called, "The Capability Immaturity Model," published in CrossTalk (November 1996, a copy available at http://cs.hbg.psu.edu/comp413/cimm.pdf). In it, he described additional negative CMM® levels describing software immaturity. I wish I could turn back time—so I could convince Tom to let me be his co-author. I am in awe of the cynicism, sarcasm, and validity of Tom's research.

But such cynicism and sarcasm did not, unfortunately, start with Dr. Schorsch. Back in the early 1970s I had the following posted on my wall when I was an applications programmer back at Strategic Air Command:

Six Phases of A Software Project

1. Enthusiasm
2. Disillusionment
3. Panic and hysteria
4. Search for the guilty
5. Punishment of the innocent
6. Praise and honor for the nonparticipants

This list was probably not new even back in the 1970s. I can find similar sayings on the web, and this particular list was located at http://en.wikipedia.org/wiki/Six_phases_of_a_big_project). I am certainly not the originator.

In 1997, Robert Glass published a book entitled "Software Runaways: Monumental Software Disasters." I have a copy, and enjoy reading it over from time to time. You can almost feel the joy when you read about such massive failures. In fact, to quote from the Amazon.com "blurb" on the book, Runaways brings a software engineer's perspective to projects like: American Airlines' failed reservation system, the 4GL disaster at the New Jersey Department of Motor Vehicles, the NCR inventory system that nearly destroyed its customers, and the next-generation FAA Air Traffic Control System that collapsed.

I really enjoyed reading the book the first time I read it, and as I said, I try to re-read it yearly (enjoying it just as much). In fact, I am pretty sure I know why I enjoy reading about spectacular failures – there are two reasons: I was not part of the project, and, there are valuable lessons to be learned from spectacular failures.

The very next year, Glass followed up with a similarly great book, "Computing Calamities: Lessons Learned from Products, Projects, and Companies That Failed." I enjoy re-reading this book, also! You see, Robert Glass understands – sometimes the only benefit from failure is that you can learn from it. Sometimes being a bad example is the last, great act of a failing software project. There is no real joy or pleasure in seeing a list of failures and associated costs. There is, however, a lot to be learned from seeing how the failure developed, what steps were ineffectively implemented to stop the failure, and what the final straw was that broke the software engineer's back!

And that's the thing: I am not REALLY enjoying the massive failures of others. What I am appreciating is that I can learn from their failures without having to actually undergo the failure myself. I don't want to be part of a multi-million dollar failure. However, I certainly appreciate "lessons learned" that allow me to effectively reason "Wow – what's happening to me is what happened in the massive XYZ failure, and they tried this to fix it, and it didn't work. Maybe I better try something else."

I recently saw—for the umpteenth time—the movie Apollo 13. I love the part where Ed Harris, playing Gene Kranz (Flight Director) says, "Failure is not an option!" In spite of massive failures, they managed to bring the Apollo 13 crew home safe, using slide rules for complex calculations. Unfortunately, that was hardware, and this is software. Failure for software projects is, unfortunately, almost always an option—from the simplest printer driver to complex flight software. To make software work, it takes a lot of hard work, process discipline, and adherence to standards, directives, and regulations. It takes a lot of research on "lessons learned" from other projects. It also takes best practices, good lifecycle selection, and great designers making workable designs—architectural, data, interface and module design.

And—maybe learning a bit from other, similar projects that failed.

All of this takes high maturity. As I frequently tell my students, anybody can write code. Want to craft software instead? That takes maturity and discipline.

Disclaimer:

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

David A. Cook, Ph.D.
Stephen F. Austin State University
cookda@sfasu.edu