BY JEFF HOLTZMAN

How Not To Be A Bad Coder

I WISH I COULD SAY THAT THE EVENTS RELATED BELOW WERE PART OF SOME ELABORATE APRIL FOOL'S JOKE. UNFORTUNATELY, THEY'RE NOT. AS A CONSULTANT, I FREQUENTLY GET CALLED IN TO HELP OUT ON PROJECTS THAT ARE BEHIND schedule and up against a serious deadline. Sometimes it's stressful, but I enjoy it. Recently I worked on a project with a team of people who were putting together a set of programs to model use of health-care facilities by the U.S. military. The model ran on a huge UNIX server to which Windows-based clients would attach, via the Internet. The clients gathered data from users, submitted it to the server, and presented results when they became available. The server portion was written in POSIX-based C running against an Informix database, and the client portion was written in Delphi. I worked on the Delphi client.

(Aside: When Delphi first came out, I couldn't interest clients in it. Now they are coming to me. This is a "Good Thing.")

As it turned out, the most interesting part of the experience for me was not the health-care model, not even the Delphi development, but what was really a personnel issue. One member of the team was not pulling his share of the load. He certainly tried, but just wasn't capable. Eventually management saw that he was not cutting it, and really needed to find another line of work. In the meantime, the rest of the team suffered.

Part of my job in this case was to analyze what Mr. X had been doing, and either fix it or re-do it from scratch. In the process, I worked closely with Mr. X. Watching him work, and trying to work with him, was a massive experience in frustration. To ease the frustration, I decided to make it a learning experience. I hoped that by trying to understand exactly how he was going wrong, observing his mistakes, and cataloging them, I could thereby develop a smorgasbord of tips for how to be a better programmer. Of course, much of the advice is negative ("Don't do this, don't do that."). But that seems to be mostly how we learn anyway.

The Scenario

The project had been undergoing development for about six months when I joined, with about eight weeks left until a major milestone and beta delivery to the client were to occur. Most parts of the application were in good shape, and delivery was set to occur on schedule.

Shortly before I joined, the project manager (PM) realized that Mr. X's area had problems. It was a complex area from several points of view. The underlying concepts were complex, the required user interface was complex, and the coding necessary to make it all work was complex. When I first arrived, the PM held several meetings to review Mr. X's plans, and subsequently decided—with the tacit agreement of the rest of the team—that Mr. X was on top of things and would be able to complete his portion of the application on time. I was assigned to work on other less-visible areas of the application.

An interesting twist was that since delivering the original requirements, the client's real interest and focus had shifted to Mr. X's portion of the application. In other words, the thing most important to the client was being developed by the weakest member of the team.

As the deadline approached, everyone was wondering about Mr. X's portion, but he kept reassuring us that things were on schedule. No one forced him to prove that he really was on target.

Next, he blew past several intermediate deadlines, and it was down to the last few days before the due date. Then it became apparent that he was behind. It

HOW NOT TO BE A BAD PROGRAMMER

Know your requirements: If you don't know what you're supposed to be doing, your only chance of succeeding is through blind luck.

Know your tool set: If it's new to you, learn it piece-wise, but learn each section well. You'll be much more valuable to your team-and yourself-if you're expert in some areas than half-fast overall.

Work on one thing at a time: Resist all temptation to jump around. Sure, in software the ankle bone's connected to the thigh bone, and so on, but if the individual bones are weak, the skeleton as a whole will never stand up.

Know underlying theory and techniques cold: You wouldn't expect an MD to practice his trade without a thorough grounding in anatomy and in giving injections. Software is no different.

Don't make fundamental errors: If you repeatedly do, you're almost certainly in the wrong job.

Strive for clarity and concision: When logic starts getting convoluted, you almost certainly need to rethink the problem.

took a few more days to discover just how far behind he was, and how deeply his code missed the mark. He was pulled from the project. We all worked several 18 and 20 hour days.

At that point, it became apparent that even if we worked around the clock, we would never be able to make up the loss. So then it became a question of cutting features. Which features were we contractually bound to provide? Which did the client really need? Which could we provide given the time constraints?

The PM made some decisions, we disabled numerous items in the user interface, and quickly tried to get enough up and running to meet the terms of the contract. Despite the lacking features, the client was happy, and the project was viewed (externally) as a success. Unfortunately, things don't always work out as well.

People Not Technology Issues

Working with Mr. X. was difficult for several reasons, not the least of which was that he was a nice guy. If he had been a total jerk, it would have been easy to write him off. But he was a nice guy, and he sincerely tried to help.

Ultimately, working with him was difficult because of his lack of technical competence. He was simply in over his head. How he managed to go unnoticed so long is a mystery.

One lesson I learned early on was not to ask him too many questions, because he would get diverted, and start down a trail of digression that was almost impossible to trace. After getting intimately familiar with his code, I believe that's how he worked as well. At one point I called his "methodology" a hypertext nightmare. Everything seemed to lead to everything else, with no priority or emphasis or structure.

The project manager was also nice—unfortunately, too nice. She did not exert enough control and authority. She did not hold people accountable, and did not force certain issues (such as having code reviews) until it was too late. That worked fine for the team members who were producing, but not so well in the other case. She had excellent technical understanding of the project, and she acted with good intentions. However, she lacked experience, and some team members felt that her inexperience gave them leave to refuse to submit to code reviews. Ironically, the biggest excuse was that there wasn't time.

The lesson here is: You can't be a project manager and be a nice guy. Conversely, if you want to be a nice guy, project management is probably not a profession in which you will excel.

Mr. X

Following summarizes the traits exhibited by Mr. X. If you ever find yourself experiencing anything like these, a little alarm should go off in your head, and you should fix the problem.

He was never exactly sure what he was trying to accomplish. He didn't have a good grasp of the requirements of the application we were trying to build.

He didn't stick to what he was trying to do. He constantly jumped from one thing to the next without ever completing anything. His code was littered with half-finished routines and unused variables.

He didn't have a good grasp of the tool set (Delphi). He lacked a good grasp of important underlying concepts including object-oriented and relational database theory.

He didn't have a good grasp of programming fundamentals. He repeatedly made basic errors like indexing strings from the wrong position, using lots of global variables, not initializing variables, and hard-coding string and numeric constants.

He created highly convoluted logic flows that were hard to understand and harder to debug. To me, that indicated that he really didn't think through the problem carefully, because if he had, he would have developed a more straightforward solution.

Given all that, you may wonder how Mr. X. ever got into a position that demanded so much technical knowledge, when his qualifications were so lacking. That indeed is a mystery of the political sort, and out of my arena.

Conclusions

So how can you become a good programmer? Even if you avoid the types of mistakes made by Mr. X, you won't necessarily become a good programmer. But by avoiding those mistakes, you can avoid being a bad programmer, and will undoubtedly become a better programmer. If you're lucky, perhaps you'll even become a good programmer. More likely, though, you were born that way . . . or not.

That's all for this month; until next time, you can contact me via e-mail at jkh@acm.org. EN