Poisonous Programmers

It's well known that the success or failure of projects depends just as much on having the right people as it does on technology. But how do you prevent certain "rogue elements" from sabotaging your project? KV has lots to say about this "nontechnical problem." And surprise! His solution involves neither physical nor verbal abuse. We encourage you to suggest your own solution, or ponder other problems, in e-mails to kv@acmqueue.com.

Dear KV,

I hope you don't mind if I ask you about a non-work-related problem, though I guess if you do mind you just won't answer. I work on an open source project when I have the time, and we have some annoying nontechnical problems. The problems are really people, and I think you know the ones I mean: people who constantly fight with other members of the project over what seem to be the most trivial points, or who contribute very little to the project but seem to require a huge amount of help for their particular needs. I find myself thinking it would be nice if such people just went away, but I don't think starting a flame war on our mailing lists over these things would really help.

Any thoughts on this nontechnical problem?

Burning Out

Dear BO,

You're right, starting a flame war won't help. I've actually never understood what people who start and maintain flame wars get out of them, but then I've always found more productive ways to get other peoples' attention—often by insulting them in person, which, honestly, is a lot more fun.

It turns out that most projects involve people, often

Got a question for Kode Vicious? E-mail him at kv@acmqueue.com—if you dare! And if your letter appears in print, he may even send you a Queue coffee mug, if he's in the mood. And oh yeah, we edit letters for content, style, and for your own good!

A koder with attitude, KV ANSWERS

YOUR QUESTIONS.

MISS MANNERS HE AIN'T.

more than eight people. Eight is about the limit where human groups seem able to cooperate on a goal, and also about the

right size for an enjoyable dinner party. Once you have to involve people in a technical project, you get management problems. If you thought it was hard to deal with people in a work setting, where at least there is the remote threat of firing the useless, the problems of managing a *volunteer* project would make you wish for that tyrannical manager you once worked for.

The type of people you are describing have been well, and amusingly, documented by Ben Collins-Sussman and Brian Fitzpatrick, two of the developers behind the Subversion source-code control system. They refer to such people as "poisonous" and have presented their highly entertaining take on how to deal with them in several venues, including Google Video (<http://video.google.com/videoplay?docid=-4216011961522818645>).

I will not rehash their entire presentation, but they make several points that are well worth keeping in mind in any project, not just open source.

One thing they point out is that the people working on any project need to have shared values, and their list of values consists of politeness, respect, trust, and humility. It will probably shock regular readers of KV to know that I share the same values and that my list does not include beatings or assassination. Although it may seem obvious to most people that everyone is deserving of politeness, respect, and all the rest, anyone who has read a mailing list in the last, say, 15 minutes, can easily find examples where conversations descend into a replay of the Argument Clinic by Monty Python's Flying Circus. If you haven't seen it, go find it, it's out there. I'll wait.

People join open source projects for many reasons, and let's face it, sometimes their reasons are less than selfless. The thing you need to decide if you are part of a project—and particularly if you run a project—is whether or not a particular person is contributing, neutral, or a drain. Contributors are great, but the majority of people on a project will have time to contribute very little. The largest

percentage of people can be considered neutral: they do not contribute much, but they also do not unnecessarily drain resources. A project's users fit nicely into the neutral category, and you want to keep them. The drainers, however, are the poisonous people who need to be weeded out for several reasons.

The first is quite obvious: they are a drain on the project. The second is less so: poisonous people often drive out contributors, who don't see the need to spend their valuable time rehashing old arguments or suffering abuse. Poisonous people make your project seem like it isn't a nice place to work. If a new user, or possible contributor, starts reading your mailing-list archives and they're simply drenched with petty arguments and bile, the newcomer is rather unlikely to go much further with your project, unless it is something he or she absolutely needs and cannot get anywhere else—and few projects fit that description.

The best way to rid your project of poisonous people is not to let them in at all. Several large open source projects have a mentoring system whereby new contributors do not become full-fledged members of the community until they are vouched for by other members of the community and in some way socialized or taught the ropes of the project. This kind of advice seems useful only when dealing with contributors, but it should also be applied to deciding whom you allow on your mailing lists or other communication systems. If someone is abusive, needy, or otherwise a drain, then ask that person to leave, and the sooner the better. The longer someone continues to be a drain, the harder he or she will be to get rid of later.

For a longer and possibly wittier elaboration of many and more of these points, please see the full presentation by Collins-Sussman and Fitzpatrick.

KV

Dear KV,

During one of our weekly project meetings at work, we were trying to figure out how to deal with our software wedging. The system is a multithreaded server, with no user-facing parts, which collects and sorts data from our networked application. One person (not me) recommended we just write another program that detects when the software is wedged and restarts it. After all, on modern computers, restarting a program doesn't take that long, and losing small bits of data is better than having the system wedged until a human operator can come along and properly shut down and restart the program. Are such programs common?

Random Reset

Dear RR,

Please, please, please tell me that you do not work for a bank, brokerage house, or credit card company. That is, unless the small bits of data you're losing are my bar tabs.

The type of program you are referring to is often called a watchdog, for reasons that should now be obvious. Watchdog code is often seen in device drivers, because some devices are easily confused and need a hard reset if they get wedged. In the device driver case, a special piece of code—the watchdog routine—is set to be called periodically, say once per second. Its job is to wake up, check the status of the device, and take the necessary actions if the device seems to be wedged. Such code should be used sparingly, and I am willing to put up with it only when dealing with systems that are hard to change, such as broken, but oh so popular, bits of hardware. I will not name names here, since I prefer not to talk to the ACM lawyers, but look at any open source operating system to find a fine example of broken hardware helped to limp along by heroic device drivers.

In a pure software system a watchdog program should not be necessary. Whereas programming is complex, and programming multithreaded systems is even more complex, a software system should be under the control of its programmers at all times. A wedge should be an infrequent event. If your system freezes so often that you find you are writing a program whose only job is to reset it, then you, and your team, have failed as programmers. Please pick up your pink slips at the HR department.

No, wait...it's actually time to debug, rewrite, or redesign your system, and your priorities should be to proceed in that order—not the reverse, as so many cowboy programmers prefer. "Oh, the program froze again. Can we take two years, with a team of 10, to do this right?" No! Too often people decide that a bug is unfixable without a rewrite just because they enjoy writing new code more than they do fixing old code. Unfortunately, these are just the types of people who should not be writing code at all.

KV

KODE VICIOUS, known to mere mortals as George V.

Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler who has made San Francisco his home since 1990.

© 2008 ACM 1542-7730/08/0100 \$5.00