The Non-psychopath's Guide to Managing an Open-source Project

**RESPECT YOUR
STAFF, LEARN
FROM OTHERS,
AND KNOW WHEN
TO LET GO.**

Dear KV,

In the past you've written about managers and management in your columns, and I note that, on balance, you do not often have nice things to say. Even when you write about management in noncorporate culture—such as open-source projects—you describe it in such a way that no technically minded person would ever want to get involved in managing people or projects.

I'm writing now because I have recently been convinced (I fear you will say duped) to accept the leadership position in a long-running, open-source project—and not only that, I am working with both volunteers and paid staff. I agreed to this for what are probably the usual reasons, but most of all, out of a desire to help the project continue to grow and do well. Naive, perhaps, but that was the reason at the time I agreed to the job.

Now I find myself deluged on all sides with technical problems—some big and some small—as well as people problems among both the volunteers and the paid folks I work with. I feel like everything is out of my control and an emergency all at the same time, although, of course, I say, “Everything is fine,” as I don't want to let on that it's not. While I am not at the end of my wits, I fear that after a year in this position, I definitely will be, which I think is the reason the person in this position before me stepped down after two years.

I've tried watching videos that might help me, including "How Open-source Projects Survive Poisonous People," the Google TechTalk by Ben Collins-Sussman and Brian Fitzpatrick that was mentioned in one of your columns (*acmqueue*, January-February 2008), and I've skimmed a few books on managing, but the books really are not engaging at all and seem to make my head hurt even worse.

Is there some knack to managing open-source projects that I can acquire without feeling like I'm running the gauntlet?

Duped and Deluged

Dear Duped,

Transitioning from one of the technical faithful to one of the hated PHBs (pointy-haired bosses), whether in the corporate or the open-source world, is truly a difficult transition. Unless you are a type who has always been meant for the C-suite—where, it is reported, a certain percentage of CEOs are psychopaths—it's going to take a lot of work and a lot of patience, mostly with yourself, to make this transition. Doing something "for the good of [blank]" usually means you are sublimating your own needs to the needs of others, and if you don't acknowledge that, you are going to get smacked and surprised by your own reactions to people very, very quickly.

As for the fact that I "do not often have nice things to say" about managers and management, well, go back and read a few more of my pieces, and please let me know just how many of them actually say *anything* nice. Go ahead, I'll wait; it will not take long.

Most technologists like interacting with technology more than they like interacting with humans.

Probably the first hurdle most technical people have to pass in helping others get their work done—which is what good managers actually do—is forgetting all the lessons they learned from typical managers they worked with in the past, and learning some new ones.

For some reason, technology is full of managers who can be classified only as, well, awful. KV thinks this is because most technologists like interacting with technology more than they like interacting with humans. When my program crashes for the 30th time with some Heisenbug, and I scream a litany of curse words that would reduce a Navy salt to tears, no one, other than perhaps those within earshot who didn't really know that those positions were possible, is really hurt. If you were to direct even one-tenth of that anger at a typical human being, you would both wound them emotionally and make it harder to work with them in the future.

Lesson number one, people are not machines. Now, I know some smart-ass (normally that would be KV) will point out that we're all just biological machines, but until someone produces a credible architecture manual and instruction set for human emotions, let's just say that we need to treat people differently from computers. I mean, you don't put your latest Hacker Space L33T sticker on the person in the next cubicle as you would on your laptop, right? Right.

Given what we know of typical management in technical circles, how then do you find good examples to work from? Humans learn by copying other humans, whether it's how to code, or how to treat others, or how to organize work. You have to decide, first, what success in an endeavor

means to you. If success is having the most money, then become a psychopath (the *DSM-V* has the definition) and climb the corporate ladder. If success is managing a group of people who are generally happy working together on a project, then I suggest looking for guidance from successful volunteer projects outside of technology.

One of the best managers I ever met managed volunteer theater before going into technology and running a large training group. During the five years we worked together, his group was consistently the happiest and most productive. Not that they were a bunch of Pollyannas wandering the halls in a state of bliss, but his team did have the lowest turnover and the lowest absentee rate in the company. When I asked about this, he explained that once you have worked with emotionally overwrought prima donnas who are doing what they love for zero money, and probably not much more fame, you can manage anyone. This boils down to knowing what each person is capable of, what engages them most, and then providing them as much work based on those criteria as they can handle.

The role of the manager is not to tell people what to do, but to give them the opportunity to do the work that they excel at and that serves the goal (aka the project) as much as possible. You will never be able to make this happen 100 percent of the time, but it should always be your goal.

It's also not the case that there are no good examples to follow, though there may be fewer than we might like. I particularly like the leadership style of Guido van Rossum, who developed and then led the Python project for nearly 30 years. Watching the project from the outside, I believe it

The harder you hold on to control, the less you'll have until you either quit or are pushed out by those around you who will see you as more of a menace than a help.

managed to maintain an open and collaborative structure throughout its history, and, in what I always consider the sign of a good leader, Guido stepped down from the leadership position after transitioning those functions onto others on the project. Knowing when to let go is another lesson successful managers must learn.

One of the great things about open source's open nature is that—via the mailing-list archives—you can see how decisions were made and how issues got resolved. Look to other projects that have high cohesion and good culture, and you will probably find a useful example or two. Seek out those who were led by diamonds in the rough and see how they made the same transition you're making.

And now to two things that everybody lies about. The first is that taking on a leadership position gives you more control. Actually, taking on a leadership position means dropping some control rather than gaining it. Technologists overall have some serious control issues, and while this may be a broad generalization, I recommend you stop right here and think about how you feel when something you're working on is out of your control. Heart rate up? Anger level up? Sweaty palms?

Leaders who try to maintain complete control over every aspect of a project most often fail. Building the ability to trust those around you and delegate to them is something you'd better do right now or that drowning feeling you describe in your letter is going to get worse. The harder you hold on to control, the less you'll have until you either quit or are pushed out by those around you who will see you as more of a menace than a help. So, another lesson is learning how to delegate: As a leader, you must be

able to take the long view and trust others with day-to-day minutiae. I like to call this “Seeing the Greater Narrative,” but I’m sure it has a more boring name somewhere.

The second lie is, “You can continue doing the same technical work you always did.” This is, in fact, the biggest lie told to people when they first transition from a purely technological role into a managerial one. It is complete and utter bullshit, and KV may or may not have actually said that (or perhaps worse) when this little gem was tried on him many years ago. Unless you weren’t doing much when you were doing technical work, adding leadership and management responsibilities, with all the learning that entails, is initially going to crush your technological productivity. So, lesson four: as you progress in the leadership position, it will be important to stay technically adept, even if you’re no longer dumping KLOCs into a system.

This, again, is where building trust with those around you helps. Let them explain to you the things you don’t understand but want to, either to do your job or because as a naturally technically minded person this keeps you engaged with the project goals. Too often I have seen people go into management and become truly dulled to the finer points of technology, and then they really do become pointy-haired bosses.

Being the leader of an open-source project is not something you’re ever going to learn in school, but it can be learned, and it can be done well—if only more people would assimilate these four lessons: (1) People are not machines; (2) delegate; (3) know when to give up control; and (4) stay technically engaged.

Or you can become a psychopath. I hear CEOs make serious bank.

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. Neville-Neil is the co-author with Marshall Kirk McKusick and Robert N. M. Watson of *The Design and Implementation of the FreeBSD Operating System (second edition)*. He is an avid bicyclist and traveler who currently lives in New York City.

Copyright © 2020 held by owner/author. Publication rights licensed to ACM.