DOI:10.1145/3567606

George Neville-Neil

Article development led by acmqueue
queue.acm.org

Kode Vicious

The Four Horsemen of an Ailing Software Project

Don't let the pale rider catch you with an exception.

Dear KV,

Are there any reliable measurements one can use to judge the health of a software project? I have seen many things written about the quality of software but not very much about the quality of a project itself. I ask this because I worry that I am stuck on a failing project, but it is difficult to know if it is really failing. The company I work for alternately feeds and starves the project of resources, while also saying that completing the next release on time is the key to our success. If we are the key to success, why would they periodically starve the project? I keep wondering if I am a frog in a slow boiling pot of water and that I will only know I should have left once it is too late. If there are measures for software quality, there must surely be measures for project quality?

Heating Slowly

Dear Heating,

Software teams, unlike software projects, are made up of people, and interactions with people are messy, which is why some of us went into this field in the first place: to avoid the messy humans and to work with the wonderfully logical and exact machines. Unfortunately, it is difficult to build anything interesting with one person, so you wind up working with a team, and teams are made of people, and as Jean-Paul Sartre wrote, "Hell is other people."

There are plenty of books and articles written about how software projects live or die, the most famous of which, *The Mythical Man Month* by Fred Brooks, I recommended in these pages long ago, and I stand by that recommendation. Brooks's work continues to be relevant because—unlike the technology we work on—people do not change very quickly, and some, including KV, would argue that people rarely learn anything from their experiences. If you doubt my cynicism, pick up a newspaper and read the front page.

Without delving deeply into specific cases of how software teams fail, we can talk about the four harbingers of the ignominious end to a software project. The harbingers bear a strong resemblance to the mythological Four Horsemen of the Apocalypse: War, Famine, Pestilence, and Death.

When a team starts to fail, one of the first harbingers to appear is War. Functioning teams can get along—at least in the work environment—and share tasks, hand them off when one member is overburdened, and generally

IMAGE BY TJ BARNWELL

work in a congenial manner. As a team starts to fail, team members become increasingly paranoid because they do not want to be blamed for the failure.

This paranoia often exhibits itself as extreme defensiveness, the idea being, "It's not my fault we're failing. My code works!" In a large and complex project, once enough of the team has hunkered down in this paranoid state, they will lash out at anything or anyone who might be seen to be impugning the quality of their work. The lashing out leads to arguments, which look a lot like war, although one carried out with code commits, snarky reviews, and nasty email threads. Hardly the stuff of immortal legend, but enough of a drain on the team to make it fall into a downward spiral of failure.

As teams fail and projects get delayed, management may decide it is time to focus effort elsewhere and to move developers off the team and into other areas of work. Removing developers starves the project of resources and leads to Famine. At this point, it would probably make sense to kill the project and completely reconstitute the teams in some more productive fashion, but managers—like developers—can often be too hopeful of a miracle save and, therefore, continue a project long after the team that is developing it should have been disbanded. Dying of famine, like death by a thousand cuts, is long and painful. If you are on a project that is constantly being deprived of resources, it is time to find something else to work on or somewhere else to work. Once famine starts, recovery is difficult and it's best to seek sustenance elsewhere.

KV has talked about various measures of software quality in past columns, but perhaps failing software quality—in the form of increasing bug counts—is one of the most objective measures that a team is failing. This Pestilence, brought about by the low morale engendered in the team by War and Famine, is a clear sign that something is wrong. In the real world, a diseased animal can be culled so that disease does not spread and become a pestilence over the land. Increasing bug counts, especially in the absence of increased functionality—which is when code fixes cause more bugs rather than actual fixes—are a sure sign of a coming project apocalypse.

Without delving deeply into specific cases of how software teams fail, we can talk about the four harbingers of the ignominious end to a software project.

The final horseman is not a harbinger of Death, but Death itself. Eventually, either management or the VCs will be forced to see the failure for what it is, kill off the project, and disband the team. In the most extreme cases, this will also destroy the company itself. It is a moment those of us who have worked in the industry for any length of time have seen—often firsthand—and it is never pretty. When you see War, Famine, and Pestilence on a team, if you are not able to fix the problem—and few of us are—then it is time to move along to somewhere or something else, lest the pale rider catch you with an exception when you are deep inside a complex function from which you will fail to return.

KV

Related articles on queue.acm.org

Velocity in Software Engineering
Tom Killalea
https://queue.acm.org/detail.cfm?id=3352692

The Hyperdimensional Tar Pit
Poul-Henning Kamp
https://queue.acm.org/detail.cfm?id=2108597

The Demise of the Waterfall Model Is Imminent and Other Urban Myths
Phillip A. Laplante and Colin J. Neill
https://queue.acm.org/detail.cfm?id=971573

George V. Neville-Neil (kv@acm.org) is the proprietor of Neville-Neil Consulting in Brooklyn, NY, USA, and co-chair of ACM *Queue* editorial board. He works on networking and operating systems code for fun and profit, teaches courses on various programming-related subjects, and encourages your comments, quips, and code snips pertaining to his *Communications* column.