acmqueue Forked Over

Shortchanged by open source

Dear KV,

How can one make reasonable packages based on open-source software when most open-source projects simply advise you to take the latest bits on GitHub or SourceForge? We could fork the code, as GitHub encourages us to do, and then make our own releases, but that puts the release-engineering work that we would expect from the project onto us.

Forked Over

Dear Forked,

The short answer is that you can't, but if that were all I'd have to say, I wouldn't have bothered to answer this letter, so let me put a lot more explanation around this.

One of the upsides *and* downsides of the move from packaged systems to SaaS (software as a service) has been the constant rolling release. When all the interactions between users and their software are proxied through a Web browser—which, minus any client code, is really interacting with a server under the control of the software developers—then rolling out a new software release is only a matter of changing the software on the server. Most companies that provide software this way can, and often do, roll out software every day, and sometimes several times per day. SaaS has provided a segment of the software industry with an amazing amount of freedom. Why worry about bugs when they can be fixed in the next push?

The downside of this mental model of development is that it introduces a certain amount of laziness into the maintenance of interfaces. Why care about maintaining an API if you can just roll out an upgrade on the next push? That attitude has little negative impact if you have a small number of consumers of your API. Once you put up your software for sharing on GitHub or a similar service, however, you have an unknown community that is depending on your software. Should you feel some responsibility toward these external users? Well, if you don't, then you shouldn't bother sharing your software, as it's not really sharable, except in the very broadest sense of the word. Yes, anyone can “fork” your repo or download the code and use it, but they cannot depend on it if your attitude toward its public face—the APIs it presents—is so cavalier that you don't even bother marking your source tree when you make API changes.

Whether or not software was developed to be packaged or for SaaS, once it has a set of consumers, it needs to be maintained using some standard practices. You may not *cut a release*, as the term goes, where there is a single unit of packaged software available for download, although such packages do make life easier for those of us who maintain package repositories such as FreeBSD/Mac Ports, Red Hat RPMs, Yum, and the like. At the very least, however, you have to indicate when you have changed an API, as the API is the contract between your package and the rest of the world. The easiest way to indicate this API change is by marking your source tree with a release tag. Choosing the tag name is a separate, painful, and tedious discussion, which I'll not go into here, other than to say some consistency to the meaning of the tags will be helpful to your downstream users.

Thinking about when to mark your tree with a release tag has some handy side effects. First, it forces you and your team to focus on an end goal, which will help you avoid the “polishing-a-turd” model of software development. Software engineers are well known for their love of perfection and being loath to release software until it’s done, where *done* is often very poorly defined. Thinking about what constitutes a release of your software focuses the developers on an end point toward which they can all work. An API change is as good a reason as any to create such a release point.

Second, it helps break down a large project into stages that are logically related. Very few projects are so small that they’re done after the first release—unless that’s the point at which they completely fail. Since you know there will be more than one release of the software, it’s better to plan for that—though, I know, for many people and groups, *plan* is a four-letter word. While you’re at it, well-maintained release notes about changes go a long way toward making happy downstream users.

If you’re serious about sharing your software, then you should be serious about *how* you share it: think about release points, tag your trees, and don’t change APIs without notifying your users.

KV

Dear KV,

One of my least favorite parts of working with open-source software is that it never seems to be complete. I’ll download, build, and install an open-source package, try to use it, and find that it almost works, but that it fails in unpredictable ways. I’ll then read the forums or mailing lists for the project, or just search Stack Overflow, and discover that the software has serious limitations that were not called out on the project home page. There ought to be a Web page that rates the quality of open-source software so that users can quickly determine whether or not a piece of software is suitable for use.

Shortchanged by Open Source

Dear Short,

I find it odd that you call out open source in your letter. Have you never used a proprietary product that didn’t meet expectations or live up to its marketing hype? If so, I would like you to pass over a bit of whatever it is you’re smoking.

The “almost-working tool” is a constant problem in software and in computing systems in general. Developers are optimists and will promise the moon while only getting you to LEO (low Earth orbit). Yes, the view is amazing from LEO, but it’s not going to get your global communications satellite the field of view it really needs. Other than telling you to take all developer and marketing statements with a grain of salt, what else can be done to avoid surprises?

Instead of using the tool and then running to the Web when it didn’t work as you expected, you should have done these actions in reverse order. One of the great things about the Internet is the number of error messages it holds and the fact that conversations held in comments rarely, if ever, disappear. A few choice words connected to your package of choice may tell you more about its suitability for your needs than the “download-and-try” model of work. I particularly like the words: *crash*, *won’t build*, *partial failure*, *segfault*, and *slow*. Combine these with the name of your package, type them into your favorite search site, and you at least may be forewarned.

You also mentioned the forums and mailing lists for a project. Why didn’t you read them first? Would you buy a house without having it inspected? Would you buy a used car sight unseen? If

not, then why would you try a piece of software without reading what its users have to say about it? While the Romans never had a word for *download*, software is as much subject to *caveat emptor* as anything else you might buy.

Finally, I would be very careful around any software that was part of a graduate student project. While many such projects result in complete systems, a significant number result in a system just good enough to get a degree, which is then dropped the moment the degree is conferred. As governments are starting to require that funded research projects put not only their papers but also their software online—as they should—I predict we’ll see a continued proliferation of such “almost-working” tools.

KV

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

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 currently lives in New York City.

© 2014 ACM 1542-7730/14/0400 \$10.00