It Takes a Community: The Open-source Challenge

**A DISCUSSION WITH
REYNOLD XIN,
WES MCKINNEY,
ALAN GATES, AND
CHRIS MCCUBBIN**

Of the many challenges faced by open-source developers, among the most daunting are some that other programmers scarcely ever think about. And that's because most programmers work in settings where "other people" attend to such matters—people who work in the legal department or human resources, for example. But when there aren't any people like that to turn to, what then?

Building a successful open-source community depends on many different elements, some of which are familiar to any developer—a clear and present market opportunity, an intelligent approach, efficient coding, and so forth. Just as important are the skills to recruit, to inspire, to mentor, to manage, and to mediate disputes—all without the use of various forms of compensation to reward and provide incentives to contributors.

What exactly does it take to pull all that off? We'll let people with track records as leaders of some of the most successful open-source projects yet mounted address that from their own experience. Participating in the discussion that follows are Reynold Xin, chief architect of Databricks, best known for his work on Apache Spark; Alan Gates, co-founder of Hortonworks, who helped develop Hadoop, Pig, HCatalog, and Hive while at Yahoo Labs; and Wes

McKinney, founder of Ursa Labs, responsible for creating pandas (Python Data Analysis Library), and currently charged with leading the Apache Arrow effort.

On behalf of ACM, Chris McCubbin, a senior applied scientist with Amazon Web Services, helps steer the discussion.

CHRIS McCUBBIN Linux was released as open source in 1992. Then came a second wave of open-source offerings that emerged throughout the dot-com era. What's it like at this point to launch an open-source project?

REYNOLD XIN One big difference is that the whole foundation concept took hold. Linux was basically just a hobby project early on, and, in that respect, it was similar to a lot of the other open-source projects started back in the '90s. Now you have the Linux Foundation, which has a multimillion-dollar annual operating budget. And while the Apache Software Foundation, which is run by volunteers, doesn't have an operating budget anything like that, it has managed to create a significant brand for itself.

One of the reasons a lot of open-source projects, from the late '90s through 2010 in particular, started out as foundations was so they'd have a better way to deal with the communities that grew up around those projects. Over the past few years, that trend has reversed a little. Largely thanks to the rise of GitHub, more and more open-source projects now launch simply by putting a repository there. Many of the projects that have started out in this way have managed to achieve a fair amount of success without any sort of help from a foundation. I definitely see that as being one of the more important current trends.

CM Certainly, the field has gotten to be a lot more congested of late. Which is to say, for every problem, it now seems there are at least a few projects offering potential solutions. But it can be hard sometimes to figure out which of those are actually being actively maintained.

RX Exactly. But even being associated with a foundation doesn't necessarily mean a project is going to be actively maintained. Project communities can come and go. In truth, many open-source projects—especially the smaller ones—depend on just one or two key contributors. As soon as those contributors move on, there's not much that remains to stand behind that code. Also, even with medium-sized projects that are backed by successful foundations, you can't be completely confident the code is going to be well maintained.

ALAN GATES To take another spin on this, I'd say that over the past 20 years we've also witnessed a growing corporate presence. Even 15 years ago, when Hadoop was launched, there were companies that would get behind certain projects and offer various types of support. By then, lots of people were already using Linux.

Companies also started letting it be known what projects they were getting behind so they could promote that as part of their identity. Red Hat was one of the first to be really successful at that. Then some others started to get behind Linux as well. At this point, corporate involvement in open-source projects has expanded far beyond that—both in terms of how they use open source and how they organize their development efforts.

RX In a way, open source has already won, to quote a friend who shall remain anonymous.

CM I definitely think that's the case. My own experience is that, with a startup I helped launch in 2012, we basically went entirely with open source for our framework. That represented a huge shift from anything I'd ever done before, which had all been pretty much DIY.

Is that a trend you still see? Or is there now a bit more pushback on open source, owing to maintainability issues and things like that?

AG There's some pushback now. Some companies are starting to say, "We really want to be involved with open source, but what's the right way to go about that now?" You see different companies trying out different license models, so it sure feels like they want to continue being involved. As Reynold says, open source has won. But the question is, What's the right form of engagement?

WES MCKINNEY The general trend is that corporations increasingly want the core platforms they depend upon to be entirely open source. But some unnerving security-related issues have come up over the years in places like the npm/JavaScript ecosystem with projects that weren't supervised by foundations or maybe just didn't have the benefit of large centralized development teams. Increasingly, corporations have come around to deciding that, while they want all of their core platform software to be open source, they're also willing to pay for the development of premium enterprise features, as well as for support and indemnification—or, at minimum, for priority-one and priority-two bug fixes.

Even going back to before 2010, there was a push away from proprietary products and vendor lock-in, and yet it took some time before people recognized

just how important it was to work with open-source software that was well maintained and well supported. In fact, the vendors that emerged as part of the Apache Hadoop ecosystem during that period—companies such as Cloudera and Hortonworks, in particular—were those created specifically to provide the peace of mind and level of security, as well as the support required for very large companies, financial institutions, and insurance companies, to have sufficient confidence to bet their businesses on open-source software.

So, what I think has happened is that companies are still paying for software but in a different way than before. It used to be that they paid for software licenses, but now they pay for assurances against faults and potential loss. Which is to say indemnification has become a much bigger issue as organizations have started putting billions of dollars on the line. And yet, we still have problems like the Equifax hack that came about as a consequence of that organization's failure to apply security patches that had been made readily available in the open-source ecosystem. That has now become a classic example of what can happen whenever users fail to maintain their open-source software properly.

CM This is where commercial software offers an interesting contrast. Vendors such as Microsoft, for example, have *forced* users to make updates. Do you think the open-source world ought to start moving more in that direction?

AG The reality in many open-source ecosystems already is that vendors offering commercial products based on open-source software tend to provide what are often referred

to as *downstream builds*, which essentially roll together open-source releases along with any applicable bug fixes or security patches.

And that, in a way, relieves the open-source projects themselves from having to shoulder the full support burden. Should a security loophole be found in a piece of open-source software, customers who have a relationship with a vendor involved with that project are likely to turn to that vendor for a patched version of the software, primarily because they know they'll probably get patched software a lot quicker that way. In fact, that's the reason many organizations entered into contracts with these vendors in the first place.

RX Open source is not necessarily about free software. Instead, it has more to do with the inherent interest companies have in building ecosystems and communities that will help them lower their cost of hiring new employees and then ramping them up. That is, if you have a development environment that's based on some wildly popular open-source technology, it's not going to be all that difficult to find people who meet your requirements.

WM That reminds me of my first involvement with open-source software about 10 years ago when I was working for a company called AQR Capital Management, essentially a quantitative investment manager. I made the same argument back then that Reynold made just now to explain why we should open source a library I'd been working on—reasoning that people out in the world would then start to become skilled at using that software, which in turn would create a pool of people for us to hire from. The software I'm talking about here, by the way, ultimately went on to

Governance is often just an afterthought now.

become the basis for the pandas project.

Back then, in 2009, my argument was one that sort of stretched the imagination, but now this has pretty much become the norm. The pattern of making internally developed software available as open source so it can attract a community of users is one that has proved to be beneficial to the companies that gave rise to those new software capabilities in the first place. Certainly, many large corporations have been moved to start open-source projects in hopes of attracting lots of users—some of whom they might be able to recruit later.

AG Assuming you've got a good project that actually manages to solve a common problem, I definitely see this as a great way to get people on board with your software.

WM Another issue has to do with project governance, which is something people tend to think about a lot less now in this modern era of grassroots projects that start out on GitHub. That is, governance is often just an afterthought now, simply because so many projects have only one or two core developers or maintainers who make all the decisions.

But as projects grow and you learn there are dozens or even hundreds of people with write access to your project repository, the decision-making process and any determinations as to who gets to decide whether a patch or a pull request ought to be merged start to get a lot more important. And, particularly when a number of commercial interests are involved in a project, this can become much more politically charged. In fact, you sometimes end up with direct competitors contributing

to a project that each considers to be important, and that can often lead to fights over whose work gets merged and whether certain work is even ready to be merged.

When it comes to some of the larger projects, I think strife over governance issues has ended up sapping a lot of energy out of those project committees. So, I regularly encourage open-source developers to plan ahead in terms of how they intend to handle governance issues, while also thinking about what they're going to do once they expand to a larger community of developers. The problems that are going to come up with 20 core developers or 100 core developers are really quite different from those you're going to see with just two core developers.

It's important to talk about what sort of mechanism you're going to use to make decisions, whether that's consensus or anointing some kind of benevolent dictator for life or whatever it might be, because you want to avoid the frustration that's sure to come if you wait until you already have a conflict on your hands to start figuring this out.

***T**here's much to be sorted out before any actual coding begins. Besides clearly defining the governance structures and mechanisms for dispute resolution, there's also the matter of settling on appropriate tooling and testing for the job that lies ahead.*

But, of course, not all of a project's requirements can be anticipated up front, and a certain amount of adaptability is always called for. Personalities change and priorities evolve as projects move along. Nothing more sorely tests

the assumptions and preparations made early on than continued growth in the size of the community. Roles change, many additional features are proposed, and more and more issues accumulate that call for management and tracking. The ability of the project to scale in keeping with these requirements becomes critical to not only the undertaking's success, but also its survival.

CM How would you describe a day in the life of a project maintainer?

WM It depends on the day. In some cases, it can feel like a day in the life of a firefighter. But, particularly for those projects that include lots of contributors, I'd say much of the day-to-day concern has to do with just making sure contributors get timely feedback on where their work stands in terms of testing and continuous integration.

The continuous integration effort to ensure something is actually ready to be merged into a project needs to happen in a timely manner or you'll end up with a backlog. This can result in frustrated contributors who might disengage from the project if they don't feel they're getting feedback promptly enough or don't see that their work is being merged within a reasonable timeframe. In the worst instances, this can even lead to a fork once people start to feel they can't make meaningful contributions to the project.

AG On the flip side, I'd say nobody comes to an open-source project just to be a maintainer. I encourage maintainers to try to strike some balance. As Wes says, you definitely need to spend some time engaging with contributors and trying to get their code merged in a timely fashion.

But you also need to get your own stuff done, which can prove frustrating to the contributors if they start to think that's why *their* stuff isn't getting in as fast as it might otherwise. Another way of looking at this, though, is that maybe it signals the time has come for some of those contributors to step up into committer roles so work can be spread out a bit.

RX I've noticed over the years that the type of work I do on Spark has changed a lot. Initially, I spent much of my time doing evangelism and development. In that particular stage of the project, we were always thrilled to bring on new contributors. But then, at some point, the job kind of changed to me saying, "No, no, no, no," to patches and new features since, somewhere along the line, a project just *has* to sharpen its focus if only because there's limited bandwidth. Even if you had every single software engineer in the world at your disposal, you still wouldn't be able to take on the whole world at the same time.

Also, contributors generally don't just come to a project randomly. It's usually because they need something out of the project for themselves or their company. They typically come with a very strong incentive to push their own agendas.

Sometimes that can fit in naturally with the project's focus, but that's not always the case. As a project committee grows larger and larger, there's a tendency for people to start pushing for things that diverge from the project's core focus—even as that focus itself is increasing in scope. So, the maintainers need to be comfortable with basically saying no to anything that might detract from that focus.

It's very easy for a project to turn into a grab bag of jumbled objectives.

Otherwise, it's very easy for a project to turn into a grab bag of jumbled objectives. Then, everything will slow down, and it will become more and more difficult to keep the project on track.

CM How do you deal with those situations where everyone is pushing their own agenda?

RX That depends on the project's communication architecture, as well as its governance structure. Different projects go about this differently. But some projects—including Spark and Kafka—adopted the idea of determining the project's direction by way of project improvement proposals. Those proposals then get voted on by the governance body of the project. Whatever decisions are made at that level then are communicated through mailing lists and web pages to the rest of the community.

AG Another approach is along the lines of something we did on the Pig project, where we sat down early as a community and wrote down our thoughts as to what we believed Pig was and what it wasn't. In that way, we essentially fashioned the Pig philosophy. Then, whenever an idea would come up to add this or that feature, we were able to reference this guiding framework we'd agreed upon as a community to determine whether or not the new idea was a good fit.

This approach helps because clarifying exactly what the project is about not only simplifies the decision-making process, but also helps everyone involved understand the reasoning behind whatever decisions are made.

CM What about contributor accountability? Say you have someone actively working on a critical part of your project

who ends up just cutting out on you for whatever reason. What can you do, given that these are volunteers?

AG Your only defense is to have a big active community so that, if any one person should leave, it won't end up crippling the project. You just have to assume some people are going to come and go—often for perfectly legitimate reasons.

RX In practice, a nontrivial percentage of the engineers working on some of the larger projects are actually paid fully by participating organizations, whether those happen to be companies or nonprofits. Those organizations are generally quite capable of backfilling for any engineers who leave. But the more tenured a person is on a project, the more difficult it's going to be to replace that person.

WM Since we're talking about some of the challenges, I'd like to note something else that regularly consumes a lot of energy—and that has to do with issue triage and issue management. All projects track and manage issues, of course, with both GitHub and Jira offering popular solutions for that. They also can be used for a number of other tasks. Besides reporting and tracking bugs, they tend to be used for tracking design discussions and the decisions that come about as a consequence. For example, if some engineers have a design document they want to propose, they might put that in the issue tracker to enable discussions.

You could also use a mailing list to accomplish much the same thing, but, many times, people are asking questions or even proposing features. So, as a project becomes increasingly popular, the volume and number of issues grow accordingly—meaning it's not unusual for successful projects that have been around for many years to have

tens of thousands of issues to deal with at any time. In fact, if a project has a large scope and a large developer and user community, it almost inevitably will end up with thousands of valid issues that require tracking and management.

Maintainers, just by the nature of their role, need to help curate all this information and decide, once an issue has been reported, whether it might be a duplicate of an earlier report or perhaps is related to some other issue that's already been noted. Truly active maintainers effectively end up functioning as librarians for their issue trackers. That's so the contributors working on the project can start taking into account some of the other open issues that might relate to the problem they happen to be concerned with at the moment such that work to address those issues might be better coordinated. That is, if you can close two issues with just one pull request, then it's probably a good idea to try to do that. But keeping all this information organized, particularly when you're talking about more than a thousand open issues, is a real challenge that can sap a lot of energy from the overall development effort.

CM As a user, I rely heavily on the information that's contained in issue trackers, but my experience is that it can be a real challenge to *find* the data I'm looking for. Are you aware of a solution for that? I'm pretty sure Apache hasn't found it yet.

AG If there's a solution, I'm not aware of it. I would agree Apache certainly hasn't found it. As it stands, Stack Overflow, Google Search, and Jira pretty much drive you to mailing lists, so that's where you end up looking, for the most part.

RX This partly explains why customers want vendors like Databricks to be standing behind projects like Spark. They want to know that, if they have a specific issue, they can just ask the supporting vendor to address it instead of having to pore over Jira and Google and Stack Overflow themselves. That's one of the biggest values we have to offer as a supporting vendor.

WM There's something else the people responsible for a project can do to enable the actual software-development process. That has to do with creating systems that perform automated testing and thus enable projects to scale in terms of the number of contributions that can be made. The value of this is often underestimated, which is why I believe many projects tend to invest insufficiently in test automation and the associated support systems. As a result, contributors often really struggle as the number of contributors grows by orders of magnitude, since the development tools at the heart of it all simply can't keep up with that sort of expansion.

And this, I think, really surprises big companies whenever they get involved in open-source projects, since they tend to take these sorts of capabilities as givens. They see all these pull requests and patches going into projects and then just expect them to take effect in short order. But what they don't see is that the developer tooling around that process ends up falling on the shoulders of a very small number of people at the core of it all.

CM Is there anything else you wish you'd focused on more during a project's early stages in anticipation of potential scaling requirements down the road?

AG For Apache Hive we ended up building our own

integration test since we didn't think there was a great open-source option at the time. Many of us wish we'd invested a bit more time in that effort since tests are hard to run now. You can't just run them on a single machine; they're too complex to use unless you have some infrastructure available.

So, I wish we'd spent more time thinking about the scaling factor instead of rushing to wrap things up as fast as we could. The project was smaller back then, so it could all run fairly quickly. But no one sat down and really thought about what the project would look like once it was fully realized. We ended up paying for that later.

RX We had a somewhat different experience in that we decided to focus on something early in the project that turned out to be pretty useful later. In fact, we ended up open sourcing it as a pull request viewer for Spark. And this is not limited just to Spark either, since we also adapted it for other projects.

Basically, it gives you a better view of GitHub pull requests. As part of that, it shows who the submitter is, what the latest status is, and which pull requests were out for comment for a long time. That's proved to be really useful when it comes to triaging. If GitHub could incorporate something of the sort to provide for better reporting on pull requests, I think many people would see tremendous value in that. Because this was built on Google App Engine, it's already shipping and is super easy to deploy.

WM One challenge here is that the tools you often find available on GitHub and open-source websites in general tend to be optimized for smaller projects with a limited

number of developers. This effectively means that, for projects to scale 10 or 100 times, they generally need to rely more on homegrown tools. It's not at all surprising that a project as large and active as Apache Spark has found it necessary to develop some of its own tools to support its process and help its maintainers become more productive.

In a sense, this is a good problem to have, but it can also really creep up on projects. As a consequence, you'll find some projects where just running a comprehensive test suite can take as much as five to ten hours of CPU time. The tooling limitations also make it difficult to plan for how projects are going to run all of the necessary builds in a timely fashion, which ultimately will impair your ability to deliver feedback to contributors at various scalability horizons. Projects often struggle as they reach those scalability breaking points.

To programmers who work in a traditional, top-down structured business environment, many of the coding challenges and conventions found within the open-source world should seem familiar. But there's a starker contrast when it comes to the environment within which this work is performed, since open-source developers operate not within an organization, but instead as part of a community—with all the sociological dimensions that might imply.

In particular, that means taking a different approach to the recruitment, nurturing, and cultivation of new contributors. And yes, it also involves creating explicit

mechanisms for dispute resolution along with codes of conduct that address a whole range of potentially disruptive human behaviors.

CM What do you see as the biggest challenges ahead for open-source projects?

WM One of the biggest challenges surely has to do with recruiting new contributors and working to get those people more and more involved—even to the point of grooming them to become maintainers. Of course, it's not always possible for contributors to become maintainers, since it's often the case that the only way you can be really effective in that role is if this work happens to be largely covered as part or all of your day job.

One of the consequences is that you often see a divide in open-source communities between those who are paid to work on the project full- or part-time and those who can afford only to make smaller contributions from time to time. Often, the experience of being involved in a community and contributing to a project tends to be optimized for the needs of those people who work on it every day.

One particularly prolific open-source developer, Pieter Hintjens, who has since passed away but at one time was very involved in the ZeroMQ project as well as a number of other open-source projects, recommended the radical approach of just going ahead and merging contributions so long as the builds passed. This was based on the assumption that maintainers would follow up later to address whatever needed to be fixed or improved—with the whole motivation being simply to reduce the amount of

People who are best equipped to improve the contributor experience tend to be the same people whose attention is required across many other areas of the project.

friction new contributors would otherwise encounter.

RX I've used that model at times, where I'd accept a patch even though I knew it wasn't great, and then I'd just rewrite it immediately afterwards. That's not a very scalable approach, and it would be difficult to push other maintainers to do the same thing. But I've personally reworked at least 100 patches that way.

Part of my reasoning was that this was a training exercise for new contributors. I'd actually copy the contributor with a note saying, "Hey, here's how you could dive deeper into that patch, and here's a somewhat better way for how to go about doing that." My thinking was that they'd get some motivation from knowing their patch got in. Then they'd also learn about how they could improve on that and become better contributors. As it turns out, some of those people have gone on to become core maintainers of the project.

CM Is there anything in particular that might be done to improve the experience of being a project maintainer?

WM It's like the old saying, "If it feels like an unsolvable problem, it probably is." I see this as a catch-22 situation in the sense that those people who are best equipped to improve the contributor experience tend to be the same people whose attention is required across many other areas of the project. Beyond that, they're often some of the most prolific contributors themselves. It's also the case that if you ask many of your best people to work as maintainers and they then succeed in making things easier for contributors, you might receive two or three times as many contributions. But, then, you'd also end up with two or three times as much code to review.

Basically, if a project proves to be successful, the maintainers almost inevitably will become overburdened. Much of the knowledge and understanding about the project that factors into making sound decisions about whether or not to accept patches will end up getting concentrated in just a small fraction of the project's overall contributor base. This is one of the central problems we face now: As a project scales, how can you maintain some sort of balance between the review and contribution processes?

CM What are the best mechanisms for determining where a project should next turn its focus?

RX We have a number of different parties that push for their own agendas. That tension tends to work out pretty well. Sometimes you have the tragedy of the commons, though, in that you have this part of the infrastructure that nobody seems to be pushing for and yet it turns out to be important for the health of the overall project.

At Databricks, we have people whose full-time jobs are basically just to work on infrastructure. What they do actually contributes to the upstream project since that work ends up having a major impact on our own systems.

CM In terms of dealing with the tragedy of the commons, perhaps some of that work could then be paid forward to other projects for the betterment of the overall ecosystem.

RX In a quirky kind of way, we've done that with Spark. One of the reasons Spark's build system works pretty well is that, back in our early days when we were running out of the [UC Berkeley] AMPLab data center, all our Jenkins stuff (used to build and test products continuously) didn't

actually run on Apache infrastructure but instead on the AMPLab infrastructure. There's one guy whose job was dedicated to maintaining important infrastructure for AMPLab's various projects. Even now that Spark has migrated from AMPLab's umbrella to Apache's, that guy is still doing the same job. Anytime something goes down for us, he brings it back up, and he makes all the Linux upgrades for us to make sure we stay on top of security vulnerabilities.

That's all really important to the success of the project, of course, but here's the thing: He doesn't do this just for Spark, but also for four or five other projects that AMPLab has an interest in. In this way, much of what has been done to improve Spark has also benefited other projects.

CM And then there are all those open-source users out there. What's being done now to improve their experience?

RX To offer an example, one of the most recent changes made to Apache Spark has to do with the data-source API, which is what you use to specify how Spark should connect with various data sources. We had a very different design not so long ago, but now that has been completely revamped, largely driven by a push from users. That is, this really has nothing to do with any of Databricks' paying customers since most of them don't actually care about this underlying data-source API.

So, it was really the open-source community that would ping us to ask, "Hey, how do you connect to Mongo? What do I need to do to write a MongoDB connector for Spark?"

The beauty of open source is that you get just this sort of multichannel feedback, which I find extremely useful. The one thing to watch out for, though, is to make sure

Identify your high-stakes users and contact them proactively to get their feedback.

this doesn't become something where the loudest voice wins. You need to make sure you're not just hearing from one guy and all of his friends as part of an effort to push one direction in particular. One way to accomplish that is to identify your high-stakes users and contact them proactively to get their feedback whenever any such change is proposed.

CM Whenever people don't see eye to eye on these sorts of things, what are your mechanisms for maintaining peace in the community? What do you do once it becomes necessary to adjudicate some dispute?

WM Well, disputes certainly do happen. Not so long ago, in fact, there was a fairly high-profile case where some developers in the Scala community effectively managed to ban a developer from contributing to a couple of open-source projects. There are developers who are sometimes deemed to be problematic, whether that's because of something they've done within the context of the project or something they've done outside the community. The GNU community has certainly gone through some turmoil following the resignation of Richard Stallman from the project, as well as from his post at MIT over comments he made about [the late financier and convicted sex offender] Jeffrey Epstein.

I think it goes without saying that you'd prefer to avoid the disruptions and distractions that come along with this sort of drama. Having some means for resolving disputes in an orderly way is quite important. Certainly, you don't want things to get to the point where the project ends up forking, since that's a really destructive action that's bound to have long-term consequences. With that said, the

potential to fork is always there. Once you go down that road, it can lead to a dark and dangerous place. Simply put, if developers cannot find a way to peacefully reconcile their differences, the community can die as a result.

It's really important to have a code of conduct, along with a culture that emphasizes civil discussions rooted in fact and logic-based arguments rather than appeals to emotion. Otherwise, you oftentimes see developer disputes degenerate essentially into ad hominem attacks and name-calling, which never are going to lead to a positive outcome.

Besides providing a more productive means for dispute resolution, a code of conduct is a means for achieving greater diversity and inclusion in the realm of open-source software, which I'm sorry to say remains dominated by men. When people see the toxic dynamics in some of these open-source communities, they might choose not to participate. So, putting structures in place that regulate conduct and create an expectation of good behavior from developers would—in addition to everything else—create a more welcoming environment.

Copyright © 2021 held by owner/author. Publication rights licensed to ACM.