SF Podcast Transcription

TikiWiki tops 500 committers!

Posted on Thursday, March 8th, 2012 by Community Team
Category: Community Showcase, Podcast

Rich: I'm speaking with Marc Laporte from the Tiki Wiki CMS Groupware project. Tiki Wiki is an incredibly full-featured product for managing all aspects of pretty much any kind of website. But the really interesting thing that we want to talk about today is the fact that this project just added its 500th committer! You can read their press release HERE.

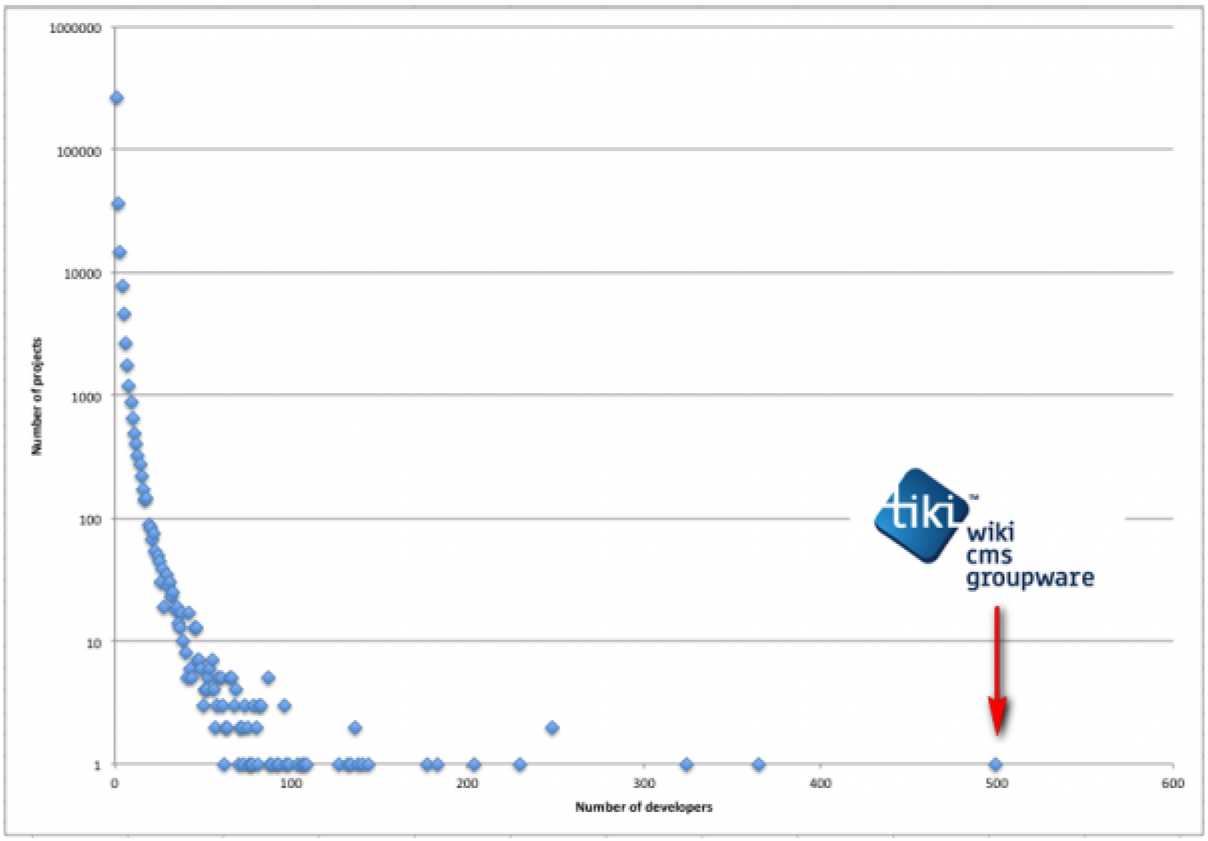
[Tiki Wiki Project Page | Browse Tiki Wiki release files | Tiki Wiki CMS Groupware website]

If the embedded audio player below doesn't work for you, you can download the audio in mp3 or ogg format.

bed height="50px" src="http://sourceforge.net/blog/communityhub/uploads/2012/03/sf16_tikiwiki.ogg" width="100px">
You can subscribe to this, and future podcasts, in iTunes or elsewhere, at http://feeds.feedburner.com/sourceforge/podcasts, and it's also listed in the iTunes store.

Rich: Now, Tiki Wiki has been the largest project on SourceForge for years, in terms of the number of committers. But 500 is a pretty significant milestone, and we'd like to talk about why you're so generous?

Marc: Well, it's Open Source, isn't it? Don't we want as many people as possible to participate? People participate in different ways - in documentation, in using the software, in reporting bugs, and participating to the code. The more people we have, the more eyeballs we have, the better chance we have of finding and fixing bugs. So that makes the perfect reason to have more people



Rich: When I’ve talked with folks about large numbers of committers, one of the objections that I hear again and again is if you have that many committers, doesn’t that open the opportunity for the “bad eggs”, so to speak, to come in and ruin it for everybody?

Marc: It’s a question that comes up a lot. The answer is not necessarily that simple, but in general, the more people you have, you’re still better off. So, “bad eggs” … we have to identify what the categories are. One of the categories could be be someone who’s not being careful with the code and just changing everything, without thinking of the other people. That’s not so much a commit access thing, but more a communication challenge. You need to have shared vision, and collaboration and communication. More than 99% of people are very respectful and they would never do that. If anything they’re too careful. They say, here, I have a patch. Can you check it? We say, just commit it, it looks okay, and if there’s a problem we’ll adapt later on. For 99% of people that’s not a problem.

The funny thing I’ve noticed is that our biggest problems have been caused by our best developers. A lot of people people think it’s going to be the occasional committer who doesn’t know the context of the system, and they do something … well, yeah, that happens once in a while, but those are typically not very big problems, and if they are, they’re fairly easy to fix. Now, why do I say biggest problems are caused by best developers? Well, it’s because they do more, and they do things which are much more tricky, so they have more chances of introducing security vulnerabilities or major bugs. So you need a model in any case to be able to find the things and to fix them.

Rich: What is your criteria for giving someone commit access?

Marc: They need to have an account on SourceForge.

Rich: So … that’s the bar there?

Marc: Yeah.

Rich: And then they say, I want to contribute, and that’s it?

Marc: That’s right.

Rich: I wish that more projects were this generous. It's always concerning to me to look at how many projects we have that only have one committer. You worry about what happens when that person either loses interest, or moves into another occupation, doesn't have as much time, or, as in my case, has a baby and doesn't have the time for it anymore.

Marc: Let me just add to the previous answer. We also have three basic rules - what we call three rules. They are fairly general. We have them as criteria, but everybody who reads them says, that's perfect commonsense. But it's good to have them. So, our three rules are:

To respect the environment. To see software, and the community as an environment. What that means is not to commit something which doesn't have the proper license. A license which fits with the project. That's common sense, but we describe that. Also, if you're going to make major changes, and move things around a lot, well you should perhaps do that in an experimental branch, and let people look at it before you commit to trunk. So that's just basic common sense.

One thing that's interesting is that it used to be "preserve the environment" and it was changed to "respect the environment," and that's perhaps a translation thing, because the person that wrote these rules originally, that proposed, them was in French language, and sometimes in different languages, words mean different things. So in French, preserve is more like respect. But then when you convert it, some people can interpret it … well, I can't change anything. I must make changes, but not really. So when we change it to respect, it means you can make changes, but you have to take into account what's already there. So that's things like migration scripts, and for the data to follow.

So that's rule number one.

Rule number two is, **commit early, commit often**. This is very common in many Open Source projects. And this is to protect yourself because the project evolves so quickly. There's a new major release every six months. There's thousands of commits. There could be several thousand commits between releases. If you don't commit early, someone else is going to change that part of the code, and your patch is going to be worthless very quickly. So the best ways is to commit early. And even if you have a feature which is not quite finished yet, if the new feature is interesting, you can still put it in software, but just take it as experimental, so people know that there may be some major changes coming there. But at least it also signals to other people, hey, I'm working on this feature. If you're interested in something similar, we can join together.

The third part is **make it optional**. And that's it in the context of, we have lots of people using the software. What's perfect for one use case may not be for another use case. We can have arguments forever on what's good or bad but it really depends on your use case.

For example, by default, should anonymous users be allowed to see a page? Well, yes, if your use case is a public website, not if it's an intranet. There's no right or wrong. So what we do is to say make it optional, and then the debates can be on what should be a sensible default. At least everybody can have their way. And that's also very good way to avoid political issues, or conflicts, where people have strong disagreements on how it should be. They can both have it their way and make it optional. And since we're a wiki community, the collaboration spirit is very high, people want to collaborate, and will find ways to get everybody happy with the situation.

Rich: Of these 500 committers, how many of them are actually active at any given time?

Marc: There are 260 that have committed, out of the 500. So some people could participate, for example, in the documentation. They may not be committers, but we still put them on the commit list, because they may commit eventually. And some people just never committed. They came, they said I have an idea. Here's your commit access, And they disappeared. That's part of life.

How many active at a given time? I'd say about 50. It depends really how you define those things, Active? What's the definition of active? How many commits? But it's about 50 or 60.

Rich: Tell me some more about managing collaboration in a community this size.

Marc: Those are the three rules. Basically, people come and participate, and the big thing in a collaborative community, is, people have different objectives. How do you deal with that? Some people want frequent releases, they want new features. Other people want stability. So we found that the delicate balance is to have frequent releases - every six months there’s a new version. But then, what we do is, we have the long-term support versions. So every third version is a long-term support. That caters to the crowd that wants slower releases, but they still want their security patches.

Rich: Let’s finish up by talking a little about the project itself. What it does, and what your plans are for it in the future.

Marc: The Tiki project started in 2002. It started as an all-in-one system for collaboration, and online publishing. Features have been added ever since.

One of the things to know as well, which makes it different from other projects - It’s an all-in-one code base. There’s not the typical thing you have seen in a web CMS, where there’s a small core, and then there’s hundreds of thousands of extensions that you can pick from.

We encourage people to participate in the core, and they make things optional. One of the benefits of that is it makes it much easier to upgrade, because all the features are in the core. And let’s say there’s a feature that’s not quite ready, you just tag it as experimental, so people know what to expect.

The other thing that is a big benefit, and encourages collaboration, is that the inherent model - since there’s only one feature - let’s say, the discussion forum. There’s not three discussion forums in Tiki. There’s one. So if there’s three people that want to participate, they’re going to collaborate together. Not having any feature duplication is really good to avoid conflicts, and to avoid any waste.

It’s the Open Source web application with the most built-in features. So it does wiki, commerce, just about any feature under the sun. And as people need more things as the Web evolves, people get their commit access, and they just add more features. The project is sort of almost a Web operating system, and it will do whatever people who are participating will add as features.

One last thing that goes in the project is the concept of dog food. We use the software ourselves, to improve it. You could say that Tiki is a community management system that that were recursively using on ourselves, and that’s one thing that really helps a lot to keep it stable, and to improve.

I’d encourage anybody who is interested in the project, and in the model, to come around and ask questions, and hopefully to participate, and test our openness policy, if you have trouble believing it. It’s funny that people use to doubt the model, but now it’s really difficult to doubt. Some people would say, yeah, it works at a small scale, but it can’t work forever. Well, I’ve been hearing that for eight years, and I think they’re going to be wrong - I’m sure they’re going to be wrong in ten years. The project is just going to be better and stronger, because people can participate.

Rich: Thanks a lot.

Marc: My pleasure. And thanks to SourceForge for hosting us all these years.