Principles of Participation for Open-Information Communities
by Karl Fogel

Keynote given at Fujitsu Labs of America Technology Symposium (FLATS) 2007

http://producingoss.com/#principles

(Thanks to Daniel Ly for proofreading the transcription.)

[Introduction by David Marvit of Fujitsu Labs of America]

Karl is pretty well-know as an open source developer and author. He worked on CVS and wrote Open Source Development with CVS. After that, he went to CollabNet as a founder developer on the Subversion project. So those of you who are developers are probably deeply familiar with Subversion.

Based on that, he wrote a very well respected book actually called, Producing Open Source Software: How to Run a Successful Free Software Project. That was for O'Reilly in 2005. Recently he worked for Google as an Open Source Specialist; and he has since left to become editor of QuestionCopyright.org, which I view as another one of Karl's hats. When I'm free, I'd be happy to talk about offline or give you today if you have questions.

I just want to add that one of the pleasures of playing a role in setting up a conference like this one is that I get to invite people that I admire and respect and enjoy the company of; and I put Karl firmly in that category. So thank you for being here, and I look forward to hearing what you have to say.

[Applause]

[Karl Fogel]

Just testing the volume here. Can everybody hear me? Yeah, good thank you.

I was very very pleased and honored to be invited. My thanks to David and Fujitsu. I was especially pleased when I read the description that went out in this symposium. In particular the third paragraph, which you may have seen yourself, which I'll just read: "Corporations that want to contribute to this infrastructure typically want to generate a return on their investment, and yet the people providing the share content, the foundation these corporations need to build upon, are operating under a completely different set of economic rules and cultural norms – the social economy rather than a monetary one." That to me means that Fujitsu gets it, which I am very pleased to see.

What I'm going to talk about in this overview, open source realities is not economic reality. There are plenty of people who can do the numbers. Martin gave some very good breakdowns of where revenue comes from in my SQL. What I'm going to talk

about are the realities of how a for-profit corporation engages with an open source community.

Let me just give you a quick overview of my background in this. I've been developing open source software for 15 years or so. The most recent and probably most well-known project I've worked on is the Subversion project, which started very early 2000, even very late 1999. I worked on it full-time for six years at CollabNet – a company which started the project and is still heavily involved in it. And then I went briefly to Google; and now I work in a non-profit organization whose goal it is to bring the open source method of developing software to areas outside software for example, books, movies, songs. And a little bit later in this, I'll show you in practice how some of that actually works.

But to start with the Subversion project, how many of you are software developers? Okay, ten hands or so. How many of you are familiar enough with software development that you know the words "version control", "patch", "commit"? About half the audience, good okay. I'll try not to use overly technical language as I go; and I'll try to explain the terms as I go along.

If you ever want to look closely at the Subversion project, that's the website. I'm going to use the Subversion project for most of the examples in this talk, except for a couple where I'm going to use a book that I wrote which Dave mentioned – Producing Open Source Software. You can read the whole thing online. It's under an open copyright. The book itself is open source; and as you will see, that has had some interesting effects in how it is spreading.

Subversion. This is a chart of the growth in Subversion service on the Internet over five years or so. Naturally, it's very pleasing to me to see that. You can see we're getting close to 200,000 publicly visible servers. That doesn't count all the ones inside corporate Internet. What is more pleasing to me is this: This is the current list of global committers. That is people who have the right to change anything in Subversion; any part of code; and have their change be automatically included in the next release. This is not by any means the full number of people who have contributed something to Subversion. You'll see more about that later. But these are the people who have stayed involved long enough to be considered part of the family; they have a vote as it were; and there's some technical meanings to that.

This list started out with about 5 people in 2000; and now there are 43 there. But wait, that's not all. These are all the people who have full or partial commit rights to the code. So some of these people have the right to commit to certain areas, but they haven't yet earned the right to commit to other areas. So they can make some changes, but not others.

The big question for the companies that back Subversion; and the companies that depend on it in a big way is how do you work with a community like this? Most of these people have never met. I've met a lot of them because I've been going to conferences, but if you cross-linked everybody on that list with who's met each other, there wouldn't be very many lines. And you probably can't read their e-mail addresses, but they don't all work for the same company, by any means. A lot of them are completely independent developers.

If you were a company facing an open source project, how do you interact with them? How do you know what kind of people they are? How do you know what makes them tick? Don't read the slide. I'm going to go one-by-one through these, but there's some basic principles that a for-profit – or for that matter non-profit or any other kind of official organization can use – to interact in a sustainable and mutually beneficial way with an open source community.

Starting with the first one. One of the rules that we have in the Subversion project – and I think you will find this to be the case in most open source projects – is that officially, the people participating are individuals. That is to say if IBM has a developer who is working on a project; and then one day that developer leaves IBM or he – *[interruption]*. So one of the rules is if that developer leaves IBM, IBM does not get to nominate another developer to take her place. She earned her position in the project by doing things; by writing code, by whatever it is she did there. When she leaves IBM and goes to work for Yahoo, her position in the project moves with her. And that doesn't mean – as you'll see later – that everything she does in the project somehow represents Yahoo, but it is a basic principle that people working in the project are just the people; and who they happen to work for is not relevant. It might help fund their travel to a conference. It might help fund their ability to work on the project, but certainly there are many people in Subversion today who use their day job. Their full-time job is to work on Subversion. But as far as making decisions in the community goes, they're just people. They have one vote and have one voice. It doesn't matter whether the company they work for is a $2 million a year or a $5 million a year company.

I'll tell you a story about that. When CollabNet hired a developer to work on Subversion after we already had a few developers – so in the list here, I'm talking about that fellow, Mike Pilato. So he got hired for the sole purpose of working in the Subversion project to be in the open source community to write code; to participate in design discussions. Everyone in the community knew that CollabNet had hired him. And so he arrived with a lot of experience coding, but not much experience in open source.

On his first day he said, "Okay, how do I make changes? How do I commit my changes now?" And we said, "Oh, okay sit down. We have to have a talk. Welcome to CollabNet. You have to earn commit privileges in Subversion project. And just because we, CollabNet started the project, doesn't mean that we can give you commit privileges. You have to send your patches to the development mailing list, which is a public list." And you can go there today and you can see those patches that he submitted. They're still archived from six years or seven years ago.

When the other developers feel they've seen it on patches, they'll get together and have a private conversation and they'll vote. If anybody has concerns about your code, then you won't get commit access; and you'll never even know that there was a vote. And if nobody has concerns – boom – one day someone will send you mail saying we'd like to invite you to commit to the project. And CollabNet doesn't get to say when that happens. Of course, there are CollabNet developers on that existing committee; and they're looking out for the company's interest. When they see that you're ready, they'll nominate you. That's the way it works. It's not that your affiliation has no relevance, but it has no "official" relevance. And to his credit, Mike said, "Oh, I see." So he wrote some patches; and a week later he had commit access; and he's now one of the strongest developers in the project.

What I'm showing you highlighted now is all the people on the Subversion project who work for CollabNet currently. There are some other names on there for example, mine, which doesn't show a CollabNet address now that used to work for CollabNet. They've been there for a while and they've moved on. So you think well, okay CollabNet has a voting block of five people. How does that affect the project? Well that's all the people who work for Google. They actually outnumber the founder of the project now. But in practice, it doesn't matter because all of those people when they get into a discussion of what direction a project should take; they realize that their personal credibility is on the line. And so you will actually have cases – and I've seen this happen and have

participated in these discussions where you'll get voting blocks of different people from one company over here; and other people from the same company over here arguing with each other. And if it actually comes down to a vote, which is very rare, they will actually vote against each other. And their managers, because Google kind of understands open source, their managers don't order them to vote one way or another. They understand that they can't do that.

So from the developer's point of view, you're an individual. And from the company's point of view, you have to let people be individuals. You have to make them understand your corporate goals, but then leave the decision-making process up to the community in the community's own way.

So principles numbers two and three – I'm going to do them together. They're sort of the same thing, which is that in an open source project. A lot of people are volunteering their time. You can't afford in that situation to have a very high threshold of participation. However much investment and effort – oh. So when you convert to PowerPoint sometimes, you get funny effects. That mailbox is an arrow; and that envelope is an envelope going that way. *[Laughter]* I didn't catch that sorry.

When you invest effort, you cannot have a situation where a person invests effort, effort, effort; and then after months of trying – boom – all of a sudden they get some reward; and then it's back to the same thing.

There are many things in life that work according to step functions. Learning to play the guitar is a step function; learning how to write a compiler is largely a step function, I think. You don't get it 'til you get it. But you can't really afford that in the way your project works as a whole. You can't afford somebody to come in and start beating against your door with patches and code submissions and design comments; and you don't give them any reward until they've done it for six months because that person will just walk away and go to a project where somebody notices them. It has to be like this.

We have taken many very concrete steps in Subversion; and I've seen other projects to these too for example, we have a patch manager. His job is the watch the mailing lists and see when anybody posts a patch; and to wait. Two days go by, nobody responds to the patch. All this person does is he follows up and says, "Hey is anybody going to take a look at this? Shall I file it in the issue tracker? Are we not going to do it? Can we

please get back to this person?" All of this happens publicly. You can see it in the mailing list archives.

Most importantly, the person who originally posted the change sees it; and all the people in the community see it. And they feel, "Oh, we really let this person down. He at least deserves a response. Let's answer." And then you see people start following up. The patch manager is serving as a human alarm clock. And actually he's got some tools now that make his job easier. He probably needs more tools, but our automation is always lagging a bit behind our needs.

So another example of keeping it away from step functions and into a linear function is – this is typical in a lot of projects, but we've really got it on steroids, so to speak, in open source projects. A volunteer set up a build automation site. And then he sent out e-mails that would go everyday. So you can see that's my mail reader there. Yes, this is e-mails, I admit it. Pass, pass, pass – whoops – somebody committed a change; and the build fails. Well if you read the mail, you can see the URL. You can go there actually if you just go down in mail; you can also see the reasons. When you go to the URL and you page down, okay the blame list right here – those are the last two people to commit changes; and these are what the changes were. You go down and here's an overview of everything that's failing.

The point is that if you can save your volunteers one minute each of work a day and you have 50 volunteers like that, that's worth spending days setting up an automation system like that. And one of our volunteers did that. He's a good coder, but he took time off and set this up because to save all of us a little bit of time is much better for the project than for him to spend a few extra days coding.

As a corporation participating in open systems, that's largely your role. You spend a lot more time on community management and technical assistance than you would expect. I spent towards the end of Subversion at my time at CollabNet 60 percent of my time on community management; and 40 percent coding, probably. And other developers at CollabNet did the same thing.

Make Decisions in Public. That doesn't mean you can never have a private conversation. If you are a principle stakeholder in a project. You're Fujitsu and you have a very large financial interesting in a project succeeding, there are going to be times when your developers need to go into a conference room and discuss this situation. That's fine, but

any decision that gets made that affects the community that the community is going to find out about, that decision needs to be made in a public forum.

What do those public forums look like? Well this is a real-time chat. This is IRC; and in fact you can see right here, this r27220 (a revision number) there – in this chat room, we have a device that anytime somebody makes a change to the code, it puts a little notice in the chat room; and that becomes a topic of conversation. You can see the fellow down here is reacting to it, actually.

This is important for a couple of reasons. People can follow what's going on without making any extra effort. Again, linear function. Just save them as much as you can. The other thing it does is it builds a sense of community because people are sitting in there chatting away. You can see they're kind of, you know – they're just chatting about nothing. And then all of a sudden they see that Mike Pilato – that's the same guy from earlier – he's committed a change. He's actually did something with the code. All of a sudden everybody thinks, "Hmm, well it's been nice chatting, but I better go do something because I want to be in there too. I want my notices showing up." So it's like a constant low-level guilt trip *[laughter]*. That's actually useful. Guilt can be a constructive thing if it's not equated with shame. Making people feel like they should be doing more is something you need to do; and you can have technical needs in doing that. However, one of our rules is whatever conversation you have in this forum, it doesn't count. You can't make decisions here. *[Whoops, I think I just laser pointed somebody in the eye, sorry about that].*

This is the mailing list archive; and it's an official rule and I'll show you where we wrote it down later. This is sort of the official record; the congressional record, if you will, of the project. A decision is final when it's posted here and nobody disagrees with it. And if somebody does disagree, then there's a discussion; and it ends in consensus; and that's where the decision is. And later when you want to refer to that decision, you point to the URL in the mailing list archives. So it's very important to have a canonical source of decision records; and it's very important that that be public.

Not all decision making processes can be completely public, for example when you're considering somebody for commit access. That has to be private because their feelings could be hurt if they get turned down. So there are few things you do in private, but then the results of those decisions, they're public.

Principles of Participation for Open-Information Communities
by Karl Fogel

Keynote given at Fujitsu Labs of America Technology Symposium (FLATS) 2007

http://producingoss.com/#principles

(Thanks to Daniel Ly for proofreading the transcription.)

Document your Culture. Here's something that always amazes me when I look at it. Look at the tiny little scroll ball thumb up there. The Hackers Guide to Subversion is not a technical guide to the Subversion code. It does include technical facts about how the code is written, but it's mainly a guide to how you participate. If you want to become a Subversion developer, read this. This is how our community works. Well, watch that scroll bar. Instead of writing code, I and a few other people spent, what – a day, two days. I mean tens of hours at least writing out this huge document; and we add to it to this day. It's still being edited. And the reason is that when there's a potential new developer, what we want is for them to come to the website; realize their interest in the project – maybe they're already a Subversion user; and to think to themselves, "Gosh, I want to get involved. How do I do that?" We just want to be able to say, "There's the answer. Go read that; and we're here. Please ask us questions." And in fact, a lot of the time people have come to us and they have already read that because it's linked to from the front page.

The result for us is that people show up feeling like we are open to new developers; that we want them there, which is exactly what we want them to feel. And also when somebody does something wrong, you can point to something in the guide and say, "Well, that's a nice patch, but you're not meeting the following guidelines that are written up. And see, look, they've been here since 2003." That sort of gives a certain authority to the community's decisions, which is very useful.

If you can possibly avoid doing real work, that's good. It would be much better for you to amplify other people's work. Here is the most dramatic example of that I note. Another thing that I spent hours doing, instead of writing Subversion code. So this is a Subversion log message. When you make a change in Subversion, the change has a description associated with it; and that's this. It's also called a commit message. So that's the change number. That fellow is the person who wrote the change. His name is Hyrum Wright. He's also the patch manager, by the way – the fellow I was talking about earlier, but he also writes code. But the most interesting part to me is this part – the change was found by Barry Scott. In OpenOffice, that circle lines up, thank you. *[Laughter]*. So the above was found by a fellow named Barry Scott. And the patch, the word "patch" is obscured there – the change itself was written by Stefan Sperling, and Hyrum Wright – up here – only applied it. He didn't actually write this change. He just reviewed it and said, "Yes, it can go in."

(Thanks to Daniel Ly for proofreading the transcription.)

Well how do we use that information? Let's look at Barry Scott. We have a whole system that reads every hour or so – every couple of hours. It reads all of the Subversion logs; and it looks for those phrases "found by" and "patch by" and it parses them; and it remembers who did what. And you get this list. So you can see the top person on the list, Madanus – 51 patches and 19 other things. Like he found a bug, he's adjusted something. Bhuvan – 37 patches, etc. Well, there's Barry Scott. Click on it. It says he's got four things that they aren't patches, they're something else. If you click on that, you can see okay, there are the two things that he found. He reviewed this and suggested this; and there they are. And hey, there's our change – the one we were just looking at. And if you click on this right here, it takes you to the change itself.

And now if we look at Stefan, he's number 53 – 5 patches, 1 non-patch. There's the same change we were just looking at; and you can go to it again. The bottom of that list, 585 people. So that's the real total number of contributors to Subversion, half-thousand – a little bit more.

So why did we spend all this time doing this? The reason is not to make these people feel good. Although I've heard since then that people, when they see their name in this list, they do feel good; and they sometimes go visit it and use it on resumes. Like they'll actually put a link to their Contribulyzer entry. But that's not why we did it. We did it because we needed to save our own time. When we're evaluating somebody to give them commit access – that is to let them become one of the maintainers of Subversion, we were spending too much time trolling through the logs, you know searching by hand for somebody's name; and then we'd find the name; and then we'd take that and we'd have to go search for that elsewhere and other changes; and we'd go manually to the website that shows you what the change looks like; and we'd review the code.

Everybody was spending a lot of time on labor that could have been automated; and we actually realized it would be worth it. Instead of writing Subversion code for someone to go off – and write a tool to automate that entire process. So now, when somebody writes to the private list, which is rarely used, but it's used for this purpose – "Hey, I think that Stefan Sperling is ready to be a full committer now. We're ready to have him be a maintainer of Subversion." The other people who aren't so sure, they go to his web page, they click on his name, they follow all the changes, they go and look at the actual code. It's just a wonderful user experience for them; and it makes them more willing to spend time evaluating new contributors so we get more people so the project remains healthy.

Principles of Participation for Open-Information Communities
by Karl Fogel

Keynote given at Fujitsu Labs of America Technology Symposium (FLATS) 2007

http://producingoss.com/#principles

(Thanks to Daniel Ly for proofreading the transcription.)

CollabNet paid for my time to set up this system. It had no direct impact on Subversion, but it had a very large indirect impact. That's the sort of thing that you spend time doing if you are the backer of developers in an open source project. It's a lot of this meta-work, and you get the volunteers writing the code. Now I think Martin [Martin Mickos, the previous speaker] would probably say, "Well wait. We write a lot of my SQL coding." Of course at CollabNet we wrote a lot of Subversion code too, but if you want a wide external community, you have to provide technical and sort of meta infrastructure for them to come together and have an easy experience.

The last thing I'll point out is you should optimize your system in favor of action, rather than discussion. And this is especially true when you have audit trails. So Subversion and all other open source software is in a version control system. Raise your hand if you know that word – version control system? Okay, it's a system for remembering who made what change; comparing changes; distributing changes; and reverting changes – undoing work when necessary.

The fact that all the code is in one of these systems means that if something goes in that's a mistake, it's very easy to take it out. What that means is you can be very liberal with what you let people do. This is not specific to code.

The example I'm going to use is actually my own book. So I wrote this book, it's under open copyright. It's all online. It's a very very much more detailed, much more in-depth version of this talk that I'm giving you now. I assumed when I released it that I would have no trouble getting people to translate it. Well, here's what the website looked like the first few months after I released it. You know, here it is. You can read the whole thing online. There are reviews. You can order it from O'Reilly. You can download a PDF. You can download it in a Rocket Book even. But there's nothing on that web page that invites people to come translate the book. And as a result, for a few months I heard nothing but silence. I mean I got some nice reviews; and I got e-mails from people suggesting changes in the book, but I didn't get anyone saying, "Hey, I want to translate your book into Chinese." I was kind of disappointed. Then I woke up one morning and realized that I was stupid. I needed to invite them. So I put a notice on the front page, "Translations are underway. Click here. Find out how you can help."

Now the web page didn't look like this the first day, but it just said, "If you want to help out with the translation effort, send an e-mail to this address; and we'll set up something and it will start going." Instantly, I got e-mails from different people who spoke different

languages. Today – well there's the English. You can read that, I can't. By the way, this translation is maybe 60 to 70 percent done; and if any one of you want to help, talk to me afterwards *[laughter]* - that would be lovely. The German is almost done now. The Spanish is almost done. It has one of the largest translation teams, so it's very well reviewed. In total, I have about 30 to 35 translators. French – there are others, but I won't show them.

The rules that I had were anybody who shows up and asks to help with the translation, the answer is, "Yes". I don't need proof that you speak your language. I don't need proof that you can read English. I don't need proof that you're a human being. I don't need anything. If you ask, we say yes. And when I say "we", I mean me and the other translators because we're all on one mailing list; and we all govern the rules by which translations happen. Often they post and they ask me, "Hey, how should we solve this or that problem?" And usually my answer is, "I don't know. I'm the only one here not doing a translation. You tell me." So they're governing themselves. The important thing is that there's no harm in letting somebody in that turns out to be a bad translator.

There are two possibilities: (1) either they're the only person who could read that language in our group. In which case, it's better than nothing, right? We might as well have some translation, even if it's bad; or (2) there are other people there, in which case they'll fix the problem or they'll privately tell me that this person isn't working out and we'll solve it. We don't have to anticipate extreme problems. We can optimize for the common case, which is that people work reasonably well; they want to do good; and they want to do what they told you they wanted to do.

Among the conventions we have is that we want to make it easy for everyone to compare their progress. So we've got a rule that you trans– *[interruption]* this is the table of contents. You translate the chapter and section titles when you finish the section. So I'm scrolling down now – Chapter 3, Chapter 4 – oh, you can see he's now doing 5. The are actually two translators. But they did do 6; 7's not done.

So just by adopting a little convention like that – that is show your work in the table of contents so we know where you are; and the rule that anybody who wants to do something, the answer is, "Yes." You don't have to have a discussion. If you cause a problem, we'll solve it later. If the community is optimized for that kind of behavior, then you'll probably get results. All you have to have then is an interesting problem and the infrastructure to support people working on it.

I would summarize that as recovery is better than protection. Don't set up gates, just set up ambulances. And when you need to, you send them in.

That's everything. I'm ready for questions. I can talk in detail about various open source projects. I just happened to use Subversion as an example because I've worked on it recently. And anyone who is curious about the non-profit QuestionCopyright.org, the one down there, I didn't really focus on it in this talk, but I'd be happy to talk more about that privately during breaks or after this. So thank you!

[Audience applause]

Audience member: [says something too quiet to be heard]

Audience member: I was just very curious how you do the voting? The closed door voting?

Oh I'm so glad you asked. The question is, "How do we do voting?" I'm really glad you asked that because the first answer is we don't. That is, voting is a fallback mechanism. It is something you do as a last resort. In the entire history of the project, we've had two votes. You know what they were about? One was whether to call SUB or SVN for the short name for Subversion? And SVN won. I was on that side, I won. And the second one was, when we're formatting the source code, do we put a space between the function name and the parenthesis or not? Those were the only two votes. I lost that one. I wanted a space and they didn't.

But the answer is that list that I showed you earlier of all the full committers – the people who are official maintainers of Subversion? If there is ever is a vote, that's the electorate. Those are the people who get to vote. But we also try very hard never to have it come to a vote because what we want is consensus and basic remit on the course, not kind of one faction off here feeling dissatisfied because they might fork the project, right? Its open source, we can't stop them from fork. So voting is a mechanism to guarantee that problems get resolved, but it's not the way that you resolve them ideally.

Okay I see that the time is up. Thanks very much.

[Audience applause]