with Eric S. Raymond

Interview

Linux and Open-Source Success

Eric S. Raymond is best known as an eloquent advocate of open-source software development, as well as for his support of the Linux operating system. Eric has received considerable attention in the press lately for his Internet publication of two annotated documents, "Halloween I" and "Halloween II"—internal strategy memoranda from Microsoft. Written by a Microsoft staff engineer named Vinod Valloppillil, "Halloween I" addresses open-source software development. "Halloween II," a related document written by Valloppillil and another staff engineer, focuses on Linux. Members of Microsoft management—including two program leads, the vice president in charge of NT development, and two members of the Microsoft executive council—reviewed and signed off on the documents.

W. Craig Trader is a software engineer at The Mitre Corp. and has been Raymond's personal friend for many years. The two spoke recently with Terry Bollinger, one of the guest editors of this issue and a member of IEEE Software*'s editorial board.*

How did you get hold of the Halloween documents, and how do you know they are authentic?

Raymond: "Halloween I" was provided to me by a source whose name I have not revealed, and will not reveal. I named it "Halloween I" for the date I received it, 31 October. The source said he thought it was interesting and worth publishing, but he was not in a position to do so himself. As for its authenticity, no hacker I know could duplicate its perspective, even as parody.

Three days after I published my annotated version of "Halloween I" on the Internet, two other sources independently sent me copies of a second memo that had been mentioned in "Halloween I." I named it "Halloween II" and proceeded to annotate it in the same way as the first one, while dealing with the media storm over my publication of the first memo. That was a pretty interesting week!

Has Microsoft publicly acknowledged authorship of the documents?

Raymond: They have. Within 48 hours of my

0740-7459/99/$10.00 © 1999 IEEE

publishing the first one, Microsoft held a press conference in which they acknowledged that the documents appeared to be authentic, but said these were low-level engineering studies that do not reflect Microsoft's policy. The reaction of knowledgeable observers was generally "Ha!"

Some news sources have suggested that Microsoft may have leaked the Halloween documents to help prove the existence of competition to NT in their US Department of Justice trial…

Raymond: I don't believe it.

Why not?

Raymond: Because there is stuff in those documents that could be viewed as evidence of anticompetitive practices. In particular, the language about "decommoditizing" protocols, as well as some other "dirty tricks" that were recommended, could be sufficiently damning in the hands of a federal prosecutor that I cannot believe anybody with any fiduciary responsibility to Microsoft would have let those documents out deliberately. There's been a lot of speculation about that, but I just don't believe it.

In your commentary you indicated that one of the most "insidious" strategies outlined in the documents was decommoditizing protocols. Why?

Raymond: That's right. This is a strategy we've seen Microsoft execute before. They tried it recently with Java and got slapped down in federal court. The tactic is to take an Internet protocol or a language specification like Java that is open and cross-platform, and add extensions to it under the cover of adding functionality. What they actually do is make their client software dependent on those extensions. Then they use their marketing power to disseminate those clients so widely that in effect the standard gets corrupted—it can't be used without the extensions. They did this with the point-to-point protocol, and tried to do it with Java. They managed to break the Java encryption handshake that is used to authenticate PPP connections. They are doing it with Microsoft mail format to drive out open-standard SMTP (simple mail transfer protocol), POP (post office protocol), and IMAP (Internet message access protocol) servers. When they succeed in getting that kind of extended, embraced lock on a protocol, then, even if it has the name of an open standard, it isn't one anymore. It has become a lock-in device to control Microsoft's customers.

What are some of the benefits of open protocols?

Raymond: You can have multiple vendors competing to produce clients and servers, all using a common protocol. That produces more consumer choice and lower prices. When a protocol is locked up, you are stuck with a monopoly supplier—and a monopoly supplier can raise prices at his or her pleasure. That's the world that Microsoft wants to live in.

The Halloween documents make several surprisingly favorable comments about the ability of open-source methods to produce new software. Do any of those comments stand out in your mind?

Raymond: Perhaps their most dramatic comment was that open-source evangelization efforts over the Internet seem to scale faster than those of Microsoft, which means we're out-marketing them. They also said that the ability of the open-source community to capture intelligence and use it to come up with good technical solutions rapidly is nothing short of amazing, which says we're outthinking them too.

Eric, you were quoted extensively in the Halloween documents, weren't you?

Raymond: Yes. Part of "Halloween I" was a reaction to a white paper I wrote called "The Cathedral and the Bazaar." It was partly because of those quotes that I felt it appropriate for me to respond to the documents on behalf of the open-source community.

The Cathedral and the Bazaar

Let's talk for a bit about that white paper, in which you first elucidated your principles of open-source software development. For readers who may not be familiar with open-source development, what are its principle themes as described in your paper?

Raymond: The principle theme that I described was a process of extremely rapid code evolution and massive independent peer review of code, which we have seen in practice in the development of the Internet and of Linux. These lead very quickly to technical solutions of extremely high quality. In fact, it appears to be a dramatically more efficient method of software development than the traditional closed, centralized development shop.

As the complexity of software requirements and software projects continues to escalate, we are passing out of the regime in which traditional closed-source development can be effective. We are now

entering a regime where only decentralized models, like the ones used to develop both the Internet and Linux, hold any plausible promise of software reliability and reasonable delivery times.

For me, what you just said represents one of the great paradoxes of open-source development. Traditional software development wisdom says that to build highly reliable software you must use an intensively controlled, thoroughly managed, and deeply "observed" software development process. Linux…

Raymond: …turns all that wisdom on its head!

Can you explain this reliability paradox? How can a software development process with essentially no management overhead, except for one or two part-time people, result in highly reliable software and rapid delivery times?

Raymond: The problem is that traditional methods of software engineering control and software quality assurance, such as the Capability Maturity Model or Personal Software Process, are all efforts to mechanize aspects of program generation and quality checking that fundamentally can't be mechanized. You just cannot get away from needing high-level intellection by a human mind to evaluate program quality. But human intellect is exactly what the Linux method gives you, in huge quantities. It gives you lots and lots of people looking at the code. The lesson we're seeing is that as sloppy and empirical and post facto as that sounds, it is in fact a more effective method of getting code quality than any of the a priori approaches people have tried in the past.

We'll always have a mixture of methods, because there are some kinds of software for which it makes economic, practical sense to be closed source. I don't think those cases are common, but they do exist. In general, the more critical a piece of software is and the more important it is to the computing infrastructure, the more its customers will insist on its being open source.

I suspect most software developers would have said that exactly the other way around. That is, the most critical pieces of software are the very ones businesses should try to control the most tightly. Why should a company hand over critical software to a development process that is only very weakly managed, lacks traditional configuration control, and is open for all to see?

Raymond: It comes back to the same basic issue: open source is the only way you get the kind of nine-nines reliability that is normal in other branches of engineering.

Through peer review?

Raymond: Yes. In most of software engineering, we do not institutionalize peer review of sources, and our reliability is terrible. But in the development of Internet technology we have routinely institutionalized peer review of sources—for 30 years now, I might add. The Internet is the largest system of cooperating programs in the world. It is multiplatform and multilingual, serving novice to expert users across many application domains. It does all these things that traditional software development says are really hard—and at nine-nines reliability.

So the pattern here is fairly clear. Coincidence? I don't think so!

Efficiency and Creativity

Tell us a little about the efficiency of the open-source method. Do you believe it makes better use of programmer resources?

Raymond: I don't have statistics on how many people are involved, or how we stack up against traditional projects. All I can see is that we make progress quickly—astonishingly quickly by traditional standards. Whether that's because there are a lot more of us working on the problem or comparable numbers working more efficiently, I can't really say.

From a slightly different angle, an a priori reason for believing that open source is more efficient is that it lets people work on problems they care about and code they love. It's a well-known fact that in any kind of creative work motivation is more than half the battle and if our people are nothing else, boy are they motivated!

Another reason to believe we are more efficient is our ability to build on earlier open-source software. In the world of closed source, they talk a lot about code reuse. But in the open-source world, we actually do it—routinely.

Turning the idea of sharing work the other way around, "Halloween I" asserted that the open-source community is unlikely or unable to do anything very creative, that all their products are based on work that was done before by others. Could you give some examples of tools that are innovative or new?

Raymond: A good one that I saw recently is the Squeak project, which involves radical work with extremely flexible and configurable graphics environments. It's being done by the some of the same people who did Smalltalk in the '70s and '80s. Also, there's Python, an absolutely terrific object-oriented scripting language that in many ways is even more attractive than Java. Python was developed entirely within the open-source community.

One "Halloween" criticism of Linux is that it provides poor support for the easy-to-use graphical interface needs of most computer users. Is that a valid criticism?

Raymond: Historically, yes, but that's changing. Projects like KDE (K Development Environment) and GNOME (GNU Network Object Model Environment) are addressing that area effectively. They're in alpha and beta now, so I think we'll see some pretty strong results from those projects over the next year or so.

Trader: In open-source development, people work on problems they care about, so the important parts tend to get fixed first. That is part of what happened in the case of Linux graphical user interfaces. The reliability, speed, and general usefulness of the operating system are there, but things like user interfaces that are not as critical to the ability of the operating system to perform its tasks have not progressed as quickly.

The Minimal Management Paradox

Let's go back to the lack of an explicit management structure in the Linux development approach. Without one, how do you make sure all the necessary development problems get solved?

Raymond: By letting people choose their own problems. There are enough people that eventually all the significant problems get solved—it's kind of a Chinese-army approach. Instead of taking a small number of people and saying, "OK, we say you're going to go here, because that's what our priorities are," you just create a framework. Within that framework, anybody who wants to solve a problem can bootstrap his own project and collect a community around it, and then use that community to solve the problem. You gain two things that way. One, you don't have to spend management resources trying to coerce or bribe people into doing what they really don't want to do. Two, you end up with work done by the people whom you most want doing it—the most motivated ones.

Trader: By way of contrast, a conspicuous recent trend in the closed-source industry is for companies to bring people in almost off the street, give them a six-week intensive course in Basic, Cobol, or Y2K fixes, and then turn them loose. As a result, you get programmers who know how to write code, but don't know how to debug code—or, for that matter, how to write good, solid code in the first place.

Raymond: Also, they don't understand problem analysis.

Trader: The danger is that these companies are depending upon process methodologies such as the CMM to succeed, whereas the real problem is that they don't have many creative people left in their processes. They may at best have some good high-level designers, but for the most part the organization is filled with people that can't take the "next step" in solving a hard problem. When they run into a brick wall, they don't understand that, gee, the bricks are not actually cemented together, so all you have to do to get through the wall is take the bricks down. Because they have no experience in digging deeper into the problem, they don't see such possibilities and instead are more likely to think, "It's not my problem." The open-source world is just the opposite: able and willing to explore as the need arises.

Here's an example. All of Linux's Ethernet drivers were developed by a guy in NASA, Donald Becker, because he wanted to have faster Ethernet drivers for a project he was working on. That project eventually became Beowulf, in which Linux PCs are linked together in parallel to create the 80th fastest supercomputer in the world. That was possible not because Don Becker was an expert on Ethernet drivers, but because he cared enough to become an expert and because open-source methods made it possible for him to take the next step.

Raymond: He's sure an expert in Ethernet drivers now!

Trader: Yes, and he shared that with the community, so the entire world benefited.

What Linus Torvalds Did Differently

Linus Torvalds is certainly not the only person ever to initiate a large, complex software project using free or open-source methods. Yet the Linux system he created seems to be achieving signifi-

cantly higher levels of growth and application diversity than most other free-source and open-source efforts. Why is that?

Raymond: Linus was both lucky and smart. He was lucky in that he started his development just as the Internet explosion was taking off. That let him make his initial Linux source available to an unprecedentedly large number of potential developers, all of whom he could then co-opt. Linus was also lucky in that he was operating at the tail end of 20 years of development of hacker tradition, which unconsciously informed his thinking about how to organize his development of Linux.

Now for the way Linus was smart. He saw development patterns, management dicta if you like, that were implicit in the hacker tradition, and he pushed them really hard, harder than anybody had ever done before. One such pattern that had always been part of the Internet culture was a tradition of rapid releases of prototypes, one after another. Another was decentralization of the development effort. Before Linus, it would have been thought utterly bizarre in operating system development to release a new kernel every 24 hours. But Linus not only did that, he actually made it work. He succeeded in amping up that feedback cycle to the point where he could get solid feedback and respond to it within a release cycle of only 24 hours.

This is all the more remarkable because he wasn't terrifically conscious about what he was doing. He didn't really analyze it all out. He just sort of had a feeling that if he pushed in this direction, he'd get good results. So he pushed and he got good results. And the rest of us weren't very conscious about it either. It wasn't until five years later that I looked at his behavior and said, "Wow, so this is what he was doing." That was what "The Cathedral and the Bazaar" was really all about: trying to understand explicitly what Linus had accomplished instinctively.

Linux as Multinational Software

What about international support for Linux? Does the international reach of the Internet translate into better accommodation of global users of Linux?

Raymond: An interesting consequence of the multinational nature of the Linux community is that we probably have better internationalization support than any other operating system. Access to the source code and use of developers around the world mean that in Linux you can edit text in Russian, Ethiopian, Thai, or any minority language that otherwise would not have enough wealth to generate a traditional market. You can support such groups with Linux, because the software is built and contributed by enthusiasts from those same groups.

I'm not saying we have perfection; you do have to go through a fair amount of tap dancing to do internationalization under Linux. But at least it's possible.

Trader: Linux didn't originally support French or German, let alone less well-known languages. The people who cared about their own languages were the ones who added such features to Linux.

Raymond: Here's an example. As an experiment in open-source methods, I coordinated the creation of a mail transport program called fetchmail. Some Brazilians recently sent me an e-mail asking if I'd be interested in internationalizing fetchmail so that they could have their messages in Portuguese. I said no, but that if they sent me a patch that looked clean, I would add it to the fetchmail distribution.

About a week and a half later they did. As a result, fetchmail can now emit all its messages in Portuguese. On top of that, because of the way they implemented it, it should be fairly trivial for other people to add language support for still more languages.

What about the overall impact of Linux in the developing world? How widely is it being used and deployed?

Raymond: Well, have you heard about Mexico's school system? The Mexican government recently announced that its entire elementary school system is going all Linux—as a teaching tool and for administrative support. This is a dramatic example of a third-world country finding that the incentives of open source are highly attractive: "Wow, sophisticated software and powerful engineering leverage—and at a cost we can afford." That's happening all over the world. The fact that Linux was built by a guy from Finland has also been a substantial driver of growth in the use of open-source software in Europe and Asia, where people fear domination of the software industry by US corporations and interests.

Trader: Linux and open-source software are the best economic aid packages that the world can give developing countries, because they help them become more efficient and self-sufficient over time. In effect, the open-source community is generating economic aid for the 21st century, giving it all away for free, and not even making any political noise about it. The world could use more of that sort of thing. ❖