AN OPEN WORLD: STORIES FROM THE OPEN SOURCE COMMUNITY

A collection of essays and interviews from opensource.com

IN THIS BOOKLET

LIFE

BUSINESS

LAW

EDUCATION

HEALTH

GOVERNMENT

INTRODUCTION

Opensource.com launched January 25, 2010 as a platform for discussing the ways open source is changing the world. Since then, we've helped our community share hundreds of stories about the power of open source principles to spark radical change. Each of these stories is an inspiring testament to the wonderful—and often surprising—innovations a commitment to open source values can generate.

Every one has been a pleasure to tell. We've explored some unanticipated topics over the years, but we've never wavered from our original mission: to shine a light on the places where the open source way is magnifying ideas and multiplying effort. And we remain especially interested in topics beyond technology—developments in areas like government, education, business, health, law, and everyday life, where open source continues to grow.

We've noticed that once you become attuned to open source values—collaboration, sharing, meritocracy, transparency, participation, community, and rapid iteration—you start to see them everywhere. Eventually you might wonder—like we do—just how different our world could be if everyone embraced them.

This collection offers some of our most compelling stories—portraits of a world fashioned with a passion for open source. Here you'll find tales only our community members could tell. Stories from the trenches. From the library. From the cubicle. From the capitol. From the classroom and the boardroom and the courtroom.

Anywhere open source is making waves and turning heads.

While bringing these stories to the world is easy, selecting just twelve of them for inclusion in this inaugural "best of" collection was not. Our opensource.com moderators lobbied hard for their many favorites. The essays you're about to read are among the very best we have to offer from our first year and half of publication—dazzling gems that refract and intensify that light we shine on open source.

Read them, ponder them, and don't forget to share them.

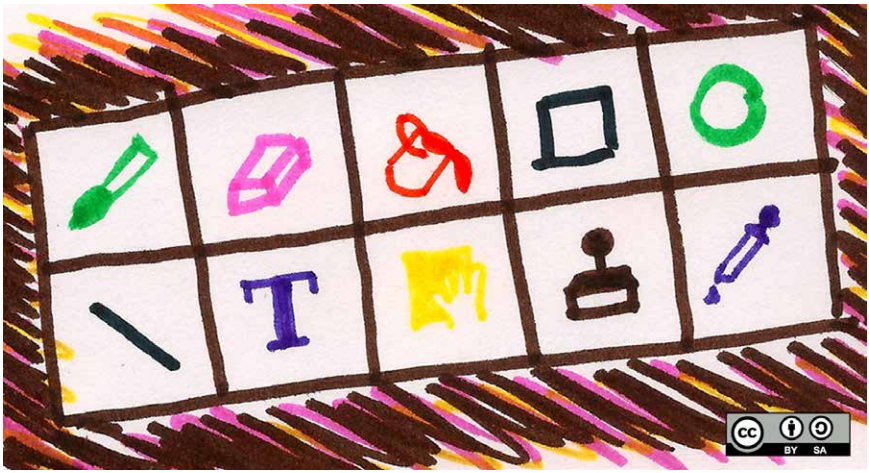
Then join the conversation at opensource.com. We have many more stories to tell together.

Perhaps one of them is yours.

AN OPEN WORLD: STORIES FROM THE OPEN SOURCE COMMUNITY

LIFE

opensource.com



THE DAY TUXPAINT BECAME CONTAGIOUS

Phil Shapiro (originally published May 2012)

I work at a public library with 28 Linux stations made publicly available in four separate rooms. The room in which I spend most of my time has 10 computers, and elementary and middle school students stop by daily after school to use them. About 90 percent of the children use the computers for games, and about 10 percent use them for doing homework. Very few use the computer for creative graphics applications. I'm bent on changing that.

Our computers run three very interesting, fun, and useful graphics programs. For young children, we have TuxPaint[1]. We also have the Inkscape[2] vector drawing program and the GNU Image Manipulation Program[3] —known as the GIMP. A few years ago our library offered a GIMP class for elementary school students. It was a joy seeing the students continue using GIMP after the class came to an end. Unfortunately, the GIMP contagion did not spread beyond the students in the class.

Last week, I was really happy to see a mother sitting at a computer with her 3-year-old son, with TuxPaint up on the screen. The child was squealing with delight as he used the various drawing tools in TuxPaint. Listening to him speak, I noticed he was highly verbal, too. "How old is your son?" I inquired politely.

The mom smiled back and said, "He's three."

I explained that TuxPaint was a free program and that the family could use it at home. Mom told me the family has a laptop, so I offered to install TuxPaint the next time they visited the library. (TuxPaint runs on all computer platforms—Linux, Macintosh, and Windows.)

I did not expect what happened next. Somehow, the word about TuxPaint spread

throughout our community. Older elementary school students started exploring it in our computer center. A few days later, a middle school boy asked how he could use TuxPaint on his computer. This is a boy who has spent hundreds of hours playing first-person shooting games.

This student went on to make a lovely drawing in TuxPaint. I commented, "You've got artistic talent."

He replied, "My teacher once asked me to draw a bunny rabbit for Easter and I drew a really excellent rabbit."

I offered to print his drawing on our color laser printer. That's when I noticed that TuxPaint was on most of the computers in our computer center. A TuxPaint epidemic was full-blown. Students of all ages were exploring different aspects of the program.

I showed the sixth-grade student's drawing to a fourth-grade girl who was enjoying TuxPaint.

"I'm going to try and make the same drawing," the fourth-grade girl said.

"Can she borrow your drawing for a little while?" I asked the sixth grade boy.

He said, "Fine!"

Within a span of 10 minutes, the computer center had transformed itself from a games-playing room to a room full of creative exploration. I can't explain how it happened, but I give a lot of credit to the programmers who created TuxPaint. For those of you who work with youth in outside-of-school settings, there is hope that students will voluntarily move themselves off a games-playing path and onto a creative exploration path.

TuxPaint, Inkscape, and GIMP are all free tools for creative exploration. It is possible to see these programs making their way into your community. I can tell you first-hand, it's a truly beautiful sight.

If you have ideas or tips for how to spur a creative epidemic with these and other FOSS programs, comment below or—even better—consider writing an article for opensource.com. This is the tip of the iceberg. Reveal to us some of the rest of the iceberg, won't you? ■

1. www.tuxpaint.org/
2. www.inkscape.org/
3. www.gimp.org/

INTRODUCING STUDENTS TO THE WORLD OF OPEN SOURCE: DAY 1

Asheesh Laroia (originally published November 2010)

From Blake Ross to Linus Torvalds, students are credited with major achievements in the open source community. But that's not the picture Yuvi Masory painted as he sat across the table from me at an OpenHatch meetup in Philadelphia.

"My lab is hiring," he explained. "We need students with programming experience and who can find answers to questions. But the students at Penn have never even heard of IRC. They've never contributed to open source."

Yuvi is a graduate student and staff programmer. He implored me to come to campus and give a one-hour talk to undergraduates about OpenHatch[1], my project to help new contributors find their way in the community.

"Give me a weekend instead," I said.

A weekend of immersion

We scheduled a planning session between Yuvi, myself, and Felice Ford, a Linux-loving classics major at Harvard who was visiting.

We settled on two days of rich interaction. Even though programming students can write code, most never see a bug tracker, and very few learn about version control. This creates a cultural rift where plenty of people bounce off[2] open source projects because of build problems or lack of community leadership. We wanted to be there to help students past problems like that.

We put up a website explaining the event[3]. For Saturday, we planned four one-hour sessions. Each session focused on a different topic and offered students some hands-on exercises. The second day was a more typical "hackathon," a project day where we helped students find their way in projects of their choosing.

To create a sense of commitment from students, and to ensure a tight student-teacher ratio, we limited Saturday to 20 students. To help us prioritize people who were particularly excited, and particularly new, we asked for:

- One sentence about how they discovered the event website
- One sentence about their current involvement in open source (e.g., "never heard of it", "run Ubuntu", "wrote most of the Linux kernel")
- One sentence about something they were excited about learning

Since our goal was to make more students aware of open source culture, we wanted to work directly with students to kickstart a local community. This strategy is unique, as far as we know. You may have read about the Professors' Open Source Summer Experience[4] that teaches professors so that they can run a semester-long class for students.

Will they come?

To get the word out, we asked the University of Pennsylvania's computer science program to email all 250 of its students. We also reached out to students at Swarthmore and other nearby colleges. Yuvi put up enigmatic index cards, and a friend of his put up flyers.

Within minutes of sending the announcement, the emails started rolling in. By the end, we heard from fifty-one applicants. Immediately overwhelmed, we switched from worrying about advertising to reading the applications.

What feels normal to us is apparently extremely exciting to these students. Reading the emails was like reading fan mail. Some wrote such gems as this:

- "My involvement with open source is primarily composed of unabashed admiration and adulation."
- "I'm most excited to learn how to initially get involved with a project, I've poked around before, but the initial learning curve has been too intimidating for me to take the plunge."
- "I have just read a lot about open source software. I'm really interested in how such collaborations create innovative and effective products (Firefox!), as well as the business perspective of how these collaborations are organised and run efficiently."
- "I'd be excited about learning pretty much anything that has to do with open source software, the communities that create it, and its social impact."
- "I hope I make it! This sounds terribly interesting!"

I just sat at my computer, reading and re-reading, saying aloud: "This is so exciting! These people are so excited!"

The students came from a wide variety of ethnic and cultural backgrounds. More than a third of the applicants were women, a (slightly) more even ratio than the Penn CIS program itself. Yuvi and I selected the thirty most excited students and told them to meet us on Saturday.

Now we needed teachers. Felice had created #penn on Freenode as a chat room for our burgeoning community. To our luck,

a GNOME hacker named Zach Goldberg[5] appeared in the channel, and we convinced him to take a bus from New York to his alma mater. My friends Jonathan Simpson[6] and John Stumpo[7] rounded out the teaching team.

We spent Friday afternoon and evening nailing down logistics. Around 1 a.m., Yuvi and I decided to switch which topics we would teach. Felice organized the students into groups based on their level of experience. Finally, we could all go to sleep.

One busy Saturday

The teachers all arrived by 10 a.m., and we set up wifi for the students to use. Teachers taught the same topic four times in a row, so we had the chance to improve our curriculum. Students switched between teachers and took a whirlwind tour of four areas within the open source community:

- Communication technologies, like IRC and mailing lists
- How to get, build, and modify open source code
- Project organization, including version control, bug trackers, and individual roles within a project
- Linux and command line skills

We broke once for lunch, and another time to discuss free software ethics in general and licensing specifically. Finally, after students had visited all four modules, we had an open discussion to wrap up the day. About twenty of our thirty students stuck through until the end.

Much feedback was positive. One student said about contributing to open source, "You made it feel a lot more doable." He continued, "You don't have to be a pro programmer to help or contribute something." Another enjoyed the variety of teachers and their "different personalities, the different take on things."

One student was particularly taken by the discussion of principles and ethics behind the free software movement. "It puts everything in a different perspective," she said definitively. However, she found our use of the term "hacker" a jarring distraction.

Toward the end of wrap-up, a student asked us when we would be running another event. At that moment, Yuvi and I looked at each other in disbelief.

Overall, students enjoyed the down-to-earth nature of the event. One student enjoyed our "conversational tone" and explained, "It was good to learn that open source people aren't cyborgs." Another called upon us to "Continue to keep it free [of charge]!"

What's next?

Read about day 2 and holding your own event[8].

Check out our photo gallery[9], snapped Saturday and Sunday. ■

1. http://opensource.com/life/10/8/ready-be-open-source-contributor-dont-know-where-start
2. http://jonoscript.wordpress.com/2010/09/28/improving-the-discovery-path-for-new-contributors/
3. www.penn.openhatch.org/old-index/
4. www.opensource.com/education/10/9/open-source-education-educators
5. www.zachgoldberg.com/
6. www.sogeekithurts.com/
7. www.jstump.com/
8. http://opensource.com/life/10/11/introducing-students-world-open-source-day-2
9. http://openhatch.org/blog/2010/photos-from-penn/

AN OPEN WORLD: STORIES FROM THE OPEN SOURCE COMMUNITY

BUSINESS

opensource.com

THE FOUR CAPITAL MISTAKES OF OPEN SOURCE

Nicolas Pujol (originally published February 2011)

How do you develop a successful open source business that lasts? Of the more than 250,000 open source projects on SourceForge, few will be successful at that goal. But one way they might think about how to do it is by doing it in reverse: What should an open source project or business not do?

The negative advice has existed since ancient times, from one religion to another. The Ten Commandments are for the most part written as what not to do. We can go for a short walk or drive around our neighborhood: road signs give us, in very short messages we can read while driving, negative advice. Ask Warren Buffett about finance. He'll tell you "Rule #1 is 'Don't lose money,' and Rule #2 is… 'Don't lose money.'"

Open source can also be better understood through negative advice. The latter can be back-tested and endure the test of time. By following a positive framework (but without falling into platonicity), one can slightly increase the chances of success. But by ignoring a negative one, you will most certainly fail.

First negative rule: Reflexivity

Don't try to sell the same product you are giving away for the same use case.

As a business, open source is built on sequential sets of events. Free software and openness create an economy based on non-monetary transactions. Instead of money, people trade their time and, generally, their mind share in exchange for value.

It is the Mind Share Market. As this happens, another economy takes shape that follows the more common path of transactions using money: the commercial market. In order for the model to work, what is free and paid must necessarily be complementary, therefore different. Differentiation is at the core of all open source businesses, and its opposite, reflexivity, is where the business tries to sell the same good that it is giving away for free. Reflexivity is destructive, as it starves the provider and prevents the business from developing financially[i].

Second negative rule: Coercion

Artificial fences are self-defeating.

One of the key reasons customers choose open source is freedom. Coercion is the opposite and relies on forcing third parties to behave in a certain way. At its roots, open source exists because customers do not want to be forced. The practice is hence self-defeating, even if it can work on the commercial market in the short run. Coercion is viral: it can over time tarnish the broad perception of open source as a deceiving scheme and may invite others to do so if temporarily successful. Barriers to entry and exit are necessary, but in a Peter Drucker style that seeks customer respect.

Let others deal with legally acceptable deception.

Third negative rule: Isolationism

What works in some contexts doesn't work in open source.

Ecosystems thrive on extensibility and die of bureaucracy. The ability to access code, to re-distribute it in certain scenarios, and to enable interactions with other components gives open source an advantage not readily available in many other business models. Hundreds of thousands of engineers (potentially one day, billions of people) working together and contributing value can outcompete a large corporation with the same number of engineers on its payroll. But for this to happen, collaboration must be extremely simple. Observe technologies like Linux, Firefox, WordPress, MySQL, Android or Wikipedia: they make it easy for others to extend their platforms from the periphery to the core; almost invasively. Isolationism blocks collaboration, partnerships, application programming interfaces (APIs), and defeats the purpose of being open.

Fourth negative rule: The salary addiction

Don't do anything only for money—especially open source.

The last capital mistake requires some context. There are situations where a job and a salary must take absolute precedence over purpose. A job may be "just a job" to support a family.

In other situations people end up in roles they didn't have to accept, but did so only for financial reasons. Phoniness is the last capital mistake of open source: it is not only immoral, but often counterproductive. People with a sense of purpose would do what they do for free, regardless of incentive. The latter exists, but cannot be the primary driver of action. Matt Mullenweg likes to say that *code is poetry*[ii]. Poetry is not created on a mechanical assembly line. Passion does not always translate into business momentum. Revenues do matter. But if you see open source as only business you will never understand it. ■

i. Even Wikipedia, a nonprofit with nothing for sale, does not give everything away. It retains its brand, infrastructure and ad space (used today for donations).

ii. This applies to code and to any other value generation and collaborative work; you are reading this article on opensource.com.

RETHINKING OFFICE DESIGN

Rebecca Fernandez (originally published May 2010)

First, a confession. Despite the hip corporate persona of Red Hat, when I first joined the company everyone had typical cubicle farm workspaces. Sure, there were hints that the company aspired to Google-like coolness: a foosball table, a game room, lots of free junk food. But in our daily office-worker lives, we were holed up in a standard maze of shared cubicles. Our idea of "open office design" was to persuade our cubemates to leave the sliding doors open.

For six months, I labored happily in my gray box, content to talk only with my supervisor and my cubemate. So when the department director announced that after the Christmas holiday week, we'd be moving to a new "open" space downstairs, I groaned inwardly. The cubicle walls were being removed; the department VP and managers would work in the same area as everyone else; and the new space would include lots of nooks and rooms for impromptu collaboration and scheduled design-thinking sessions. As the lone quiet, left-brained web developer among a host of creatives, I was certain this sudden push for collaboration meant I'd never get any work done.

I was mistaken.

According to the 2001 office design study, Offices That Work: Balancing Communication, Flexibility and Cost (pdf)[1], "the major reason for an office today is to bring people together: to socialize and share information; to inspire and inform each other; to provide guidance and feedback. Relatively little of the work of most office workers requires deep, individual concentration for hours at a time."

As a computer programmer, I was not exempt:

As the literature on computer engineers shows, this is true even for the prototypical job function requiring deep concentration. There do need to be times and places for such work in the office, but whether such places need to be assigned to one person for his or her exclusive use, or requires complete physical separation from others doing the same work, has been challenged by many corporations over the past decade.

Within a month in the new workspace, I knew more about every colleague in my department than I'd learned over the prior half-year. My own role deepened from being a ticket-resolving web monkey to a full-fledged knowledge worker and vital part of the team.

My fears about moving out of my cubicle

1. Without cubicle walls to hide behind, interruptions would be endless.

In one sense, there are more interruptions. Communication is abundant—and more frequent—when you can see your team members. But the rapid flow of information throughout the office actually reduces the email, phone calls, and traditional scheduled meetings needed, according to the study linked earlier. Surprisingly, increased visual contact actually contributes to fewer unwanted interactions. When you can glance at a coworker and see that they look engaged in a problem or irritated by a phone call, you're more likely to ask your question later than if you had walked down the hall and already poked your head into their office.

The study also notes:

Our data suggest that individual performance or productivity may be reduced in a given unit of time, while both individual performance and that of their team benefit over the life of the project. In other words, this minute's interruption can be annoying, but over the life of the project such "interruptions" tend to be seen as contributing to overall success.

2. In an open office design, there would be nowhere to go when I needed to hold a private conversation or think intently without interruption.

A well designed open layout includes places for these tasks. When Cisco redesigned their offices[2] to be more collaboration-friendly and reflect modern work habits, the company opted for a highly flexible design. Only administrative assistants were assigned longterm office desks; no one else has ownership over a particular workspace. Instead they choose the type of workspace they need for a few minutes, hours, or all day:

Cisco employees are increasingly mobile—and less and less working at a particular desk ... Throughout the day, employees [select] an appropriate environment to accomplish the task at hand: meeting in a group, participating in a conference call, or working alone on a spreadsheet or project plan.

The Cisco plan includes a quiet area deemed "the library" for work requiring intense concentration and quiet, as well as an etiquette policy, developed by employees along the way, which frames the use of different areas: non-private meetings with one other person should take place in smaller, open seating areas, not a closed conference room, for example.

The decision to change the Cisco office design was made after considerable thought:

Like most companies, Cisco designed its office space under the traditional assumption that employees would work in their own cubicles during regular work hours and would need assigned work spaces with their own desks, PCs, and phones. The result was that meeting rooms were often in short supply, while offices and cubicles remained vacant 65 percent of the time on average.

Nobody would consider building a manufacturing facility that they intended to use just one-third of the time," says Mark Golan, Cisco vice president for WPR. "And yet that's what we routinely do with workspace. We realized that assigning resources based on utilization would significantly reduce Cisco real estate costs." [emphasis added]

3. With an open design, my superiors and coworkers would be constantly scrutinizing my activity. I'd be self-conscious as I went about my work.

When we moved to the open floor plan, I found that I actually had more privacy than before—when I wanted it. Within cubicles, there is a sense of "pseudo-privacy," where your neighbors pretend not to hear your phone conversations and feel awkward speaking up if they have information that would benefit you. But in an open office space, you know who is hearing your conversations, and your coworkers feel free to provide input. If you want privacy, you know to hold the conversation in a place designed for it.

In addition, I had not given ample consideration to the value of making eye-contact with colleagues. When you notice someone approaching your desk, you can gauge whether they mean to speak with you or someone else. You have the opportunity to jot down a final thought or finish a line of code, because you have an extra moment's notice. And when you're discussing a problem with a coworker, you can invite others with a glance to join the conversation.

4. With an open office, my coworkers' annoying habits would be magnified.

Anyone who has worked for a few years has shared cube walls with coworkers with not-so-endearing habits. The one who checks his voicemail on speakerphone. Or chatters loudly and nonstop on her cell phone. Or sings gospel songs. Or paints fingernails. So you can imagine my trepidation at the prospect of the removal of those (somewhat) protective barriers.

What's interesting is that when people can see their office neighbors, they are far more self-aware. But if your coworker sings in the conference room during team meetings, you may want to lobby for a desk at the opposite end of the room.

Unexpected problems

While none of my fears materialized, other problems did surface in our space.

1. Moving day, again?

Now that our department head and other managers could watch the interaction between different coworkers, moving us from one desk to another became an irresistible urge. While my own desk only moved thrice in two years, others seemed to be packing up again just as soon as they'd settled in. Initially we benefited from the new chemistry and collaboration. After several moves, the cons of instability took over. Perhaps we should have opted for a fully flexible, choose-your-workspace environment like Cisco.

2. Added mobility requires new technology

Cisco discovered that the needs of mobile knowledge workers are different from stationary employees. Most, if not all workplaces need power outlets to compensate for the short battery life of laptops. The company tried to provide uninterruptible power supplies throughout the building, but as the units beeped after an hour to signal low power, they were highly disruptive. Cisco is considering a pilot program allowing employees to swap out dying batteries at exchange and recharge stations.

In addition, Cisco used wireless and hard-wired phone technologies to give workers

the ability to check voicemail and make phone calls from any workstation.

3. Limited number of collaboration areas

We didn't anticipate the culture shift that accompanied moving into a new space would require more spaces for collaboration. Smaller areas for non-private meetings and a second closed-door conference room would have made our space a bit more usable.

4. Neighbor immigration

Our department, Brand Communications + Design, was the first to receive permission and funds for an open office design. That space included a large, open meeting area with several whiteboards and comfy chairs. As employees from other departments were invited to meet with us, they quickly noticed what vibrant and collaborative meetings sprung from the space. "Let's meet over in the Brand Comm space" became a common refrain for anyone looking to hold an informal and insightful meeting. Unfortunately, our space was not designed to host meetings for multiple departments, and creating similar spaces in those departments would have been a valued decision.

5. Shifting requirements

An open office design must be regarded as a work-in-progress. As new needs emerge, the space must be able to accommodate. At Cisco, this meant adding personal lockers for purses or lunches, and larger filing cabinets for employees whose jobs required them to store forms or records. Within the Brand Communications + Design space at Red Hat, the function of several closed-door rooms has changed over the years, serving as everything from a video recording studio to a library to a temporary office.

Real-world examples

So what does the open office look like? And how does a business—without the budget of a Google or an IDEO—build an equally collaborative environment?

The Cisco case study shows that open office environments are actually more cost-effective than more traditional types. A building with large, closed-door office rooms could convert those private rooms into door-less, team "bullpen" rooms, where several colleagues work together. A department with cubicles could remove the walls and replace them with interconnected desks and smaller meeting areas. The ideal open office project would include its future inhabitants in the design process.

There is a lot of inspiration to be found at www.officesnapshots.com[3], with pictures of office spaces from Microsoft to Apple, Twitter to Facebook, and plenty of smaller businesses as well.

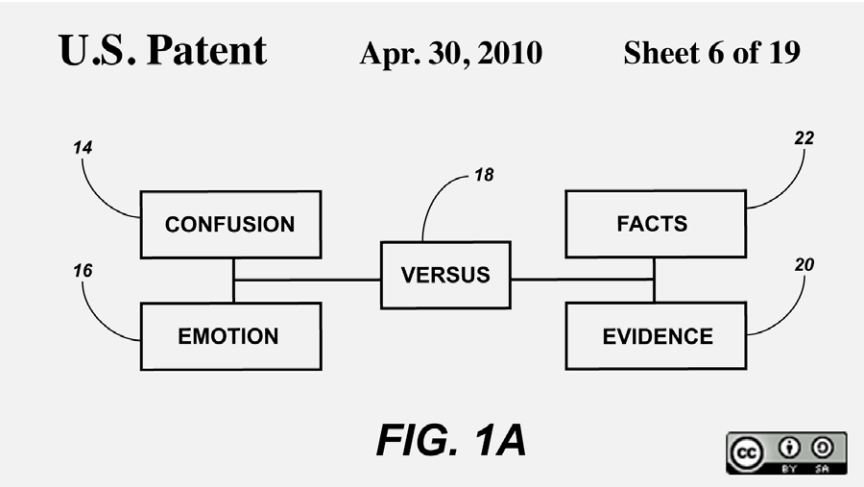
Articles like "Why Office Design Matters"[4] from Harvard Business Review, and BNet's "Three New Designs for Optimizing Collaboration"[5] provide additional ideas and case studies.

But more valuable may be talking to people who work in open environments about their experiences. ■

1. http://tinyurl.com/8tfn4es
2. www.cisco.com/web/about/ciscoitatwork/collaboration/connected_workplace.html
3. www.officesnapshots.com/
4. www.hbswk.hbs.edu/archive/4991.html
5. www.bnet.com/2403-13056_23-190685.html

AN OPEN WORLD: STORIES FROM THE OPEN SOURCE COMMUNITY

LAW



TOTAL VICTORY FOR OPEN SOURCE SOFTWARE IN A PATENT LAWSUIT

Rob Tiller (originally published May 2010)

The jury verdict last Friday in favor of Red Hat and Novell in a case based on bad software patents owned by "non-practicing entities" is an important victory for the open source community. Those in the business of acquiring bad software patents to coerce payments or bring lawsuits should be worried. Two such businesses were plaintiffs in our case, and they did their best to confuse the jury in one of their favorite locales, eastern Texas. But it didn't work. The jury unanimously found that the patents were not infringed, and, even worse for the plaintiffs, that the patents were invalid.

The case was about allegations by IP Innovation, L.L.C. (a subsidiary of Acacia Technologies), along with Technology Licensing Corporation that Red Hat and Novell infringed four claims from U.S. Patents 5,072,412, 5,394,521, and 5,533,183. The patents share a common disclosure and are all titled "User interface with multiple workspaces for sharing display system objects." The patents relate to a computer-implemented system and method for providing a graphical user interface with multiple workspaces.

Like most patent cases, this one involved technical subject matter and terminology. However, the plaintiffs came forward with minimal evidence to support their argument of infringement. They also faced abundant evidence showing that the patents were

invalid based on prior art. In other words, there was nothing new in these "inventions" sufficient for a patent.

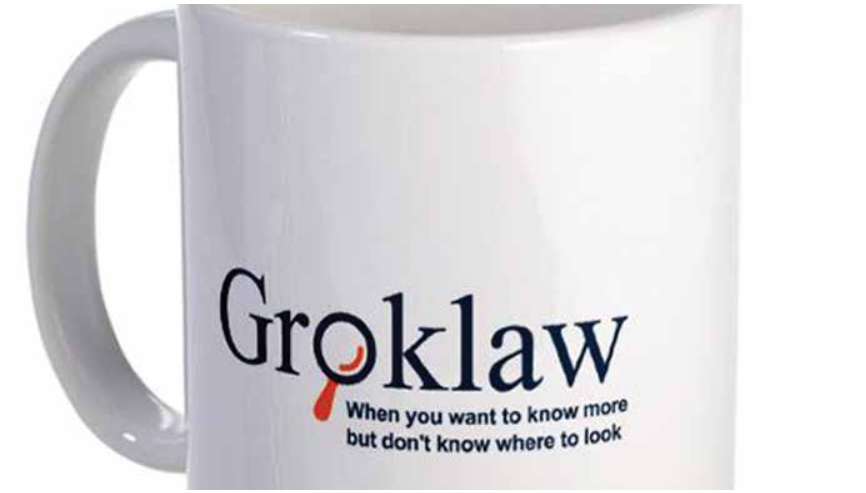
In these circumstances, you might suppose that a rational patent plaintiff would dismiss the case, perhaps in return for a token payment. Instead, the plaintiffs decided to ask the jury for millions of dollars. Their theory appeared to be that the jury might be confused by the technical terms and unsympathetic to out-of-state businesses with creative business models.

With that end apparently in view, the plaintiffs' counsel launched an attack on the theory and practice of open source software. It was clear during jury selection that our jurors had no prior knowledge of, or experience with, open source. Plaintiffs attempted to exploit this inexperience by arguing that open source software involved behavior that was, if not downright illegal, at least ethically dubious. They promoted the fallacy that open source distributors unfairly take the property of others and thereby unfairly profit. They also suggested that Red Hat's public criticisms of the U.S. patent system as it relates to software and related calls for legal reform were un-American and indicated a secret fondness for the writings of Karl Marx. I kid you not! As absurd as this argument sounds, after many hours of sitting on a hard courtroom bench, I briefly wondered whether the jury might fall for this version of the classic FUD strategy and be so fearful and confused as to find for the plaintiffs.

It turned out that there was no cause for concern. Michael Tiemann, Red Hat's vice president of open source affairs, explained the fundamentals of open source so as to make them clear, and even inspiring. He explained that open source software is about voluntary collaboration, not involuntary expropriation. He also made plain that Red Hat's legitimate criticisms of the existing patent system in no way shows a proclivity to infringe patents or indifference to patent claims, and that Red Hat respects and abides by the law.

Our side took the opposite approach from the plaintiffs, basing our case on facts and evidence, rather than emotion and confusion. Our experts carefully showed that our products were noninfringing and demonstrated specific examples of prior art. In the end, the jury saw through and quickly rejected plaintiffs' FUD. The jurors took a bit more than two hours to find every one of 23 issues in favor of Red Hat and Novell.

We learned many things from this experience, but I'll note just three here. We now know for certain that those in the business of bringing software patent lawsuits are not invincible, even in the supposedly patent-friendly jurisdiction of the Eastern District of Texas. We know that Texas juries are willing to reject bogus infringement claims and invalidate bad software patents. And we know that attacks on open source based on FUD will not stand up when subjected to the light of truth. ■



INTERVIEW: PJ ON THE BEGINNING, ENDING, AND FUTURE OF GROKLAW

Ruth Suehle (originally published May 2011)

Over the last eight years, Pamela Jones, known as "PJ," wrote volumes at Groklaw[1]—first as a blog about the holes in SCO's claims, then increasingly as a place for wider commentary on the legal issues facing Linux and open source. To summarize the site's mission statement[2], Groklaw was a full legal news resource, "acknowledged and used by all the parties, including SCO." But it was also a community—a place for open source believers to gather, learn, and share.

Last month PJ announced that because SCO as we knew it is no more, she would stop publishing new articles today[3], May 16, Groklaw's anniversary. Now she's handing the reins over to Mark Webbink[4], former general counsel at Red Hat, law professor, and board member at the Software Freedom Law Center, to create "Groklaw 2.0."

Here's what she had to say about Groklaw's past and her future.

What inspired you to start Groklaw? Did you anticipate the audience it came to appreciate?

When I started, I was literally just practicing for a job interview. I had no knowledge of the Internet, obviously, so I didn't know the whole world could see what I was doing. When people showed up, it was a shock, and the numbers—it was hundreds of people all of a sudden, then thousands, until we finally

had to move to larger quarters. After I got more used to it, it was exciting too. Because when I saw the level of technical knowledge my readers had and how much they wanted to learn how the legal process works, I realized what it could mean, what we could do, if I could learn to ride the wave.

Where do you think Groklaw has been most informative and influential?

It's hard to praise oneself without feeling idiotic. But as a group, what we showed is that if the FOSS community gets behind an effort to do legal research, no single law firm can beat them. The community we built lived computer history. The gray beards are still among us, after all. So we have UNIX guys and we have Linux guys, the very people whose code was being fought over by corporate interests.

So we were a voice, a way for the community to point out what was not true, and they could point to the evidence that it was not true. Law firms don't have that, and you could see the difference. They might have an expert, but that person can't compete with a community like Groklaw's. They'd file a document with the court and within hours the community would have taken it apart and shredded it, and they were right, over and over and over.

What I am most proud of is our trial coverage in SCO v. Novell, the jury trial. That and being the ones to first publish the previously sealed settlement agreement from the BSDi litigation. I'm proud of the fact that the community we built is still strong, still ready to do whatever needs doing. Building and maintaining a community isn't as easy as it looks. Over the years, some thought they could do a better Groklaw, and they did try, but none of them continued or ever really took off.

If you were starting Groklaw again today, with the benefit of the experience you've had, would you do anything differently?

I was naive in the beginning. I didn't know people as venal as I was about to be writing about. And I didn't know anyone personally, except for one relative, who lied without any apparent pangs of conscience. So at first, whoever showed up to help was accepted at face value. Later, I realized that some were operatives working to destroy from within. It was a sad and creepy lesson to have to learn. If I were starting it up now, I would factor that knowledge into every part of what I built.

Why did you decide to discontinue working on Groklaw?

I'll still be working on it, just not doing articles. I want to finish the Comes v. Microsoft exhibit collection and fix some other loose strings, so the work stands the test of time and is truly useful to historians and lawyers.

I can't do that and write articles every day. And I have a number of personal and other work projects that I shoved to the back burner in order to do Groklaw, and now that the emergency for Linux is handled, it's time to prioritize in a more normal way. We won, the emergency is over, and I get to relax a bit now.

So that is part of it. But the most important consideration was this: I was born to write Groklaw, about SCO and the Linux kernel and copyright litigation. But the battlefield now has shifted to mobiles and patents. I thought seriously about that, and I recognized that I am not the right person to take the lead on that. I always hated patent law, and nothing I've seen in the last 8 years has altered my feelings. I hate software patents with a passion, I think they are destroying innovation in the US, and that they particularly threaten FOSS, the open development model being

opposed to patents. I think software and patents need to get a divorce.

I consider that a serious enough matter that I thought modesty needed to inform me to stop, that others could fill the role and would if I did. Then when I announced I would stop, I was flooded with requests to find someone to continue, and I realized the community was right. It was irresponsible if I didn't try to maintain the community, their skills, in one place. And happily, we found someone. I think Groklaw will end up more important than it's been, actually, because Mark Webbink is lawyer, a FOSS lawyer, and a law professor. With him taking the lead, and his law students —and we hope eventually others at other law schools—joining the community, it can grow in the direction that is needed now. They can explain the law, and the community at Groklaw can help them understand the tech. It's what Groklaw is for, what I dreamed it should be—a place where the two communities can teach each other, so they can together hopefully help judges to understand the tech so they can reach better decisions, ones based on technical realities. So this is organic, part of what Groklaw is supposed to be, just the next step.

Part of Groklaw's success was realizing that we could contribute just as we are, without trying to be more than we were. But that means also remaining modest and aware of what we were not qualified to do. I always said the only legal advice I ever give is, Ask your lawyer. Well, now Groklaw is going to follow that advice and get a lawyer. It's a natural progression. And it's the right time, given Microsoft's rather obvious strategy of using patents against GNU/Linux.

How would you describe the relationship between Groklaw and open source?

Groklaw is an application of Open Source ideas to legal research. But Open Source doesn't mean a free for all. With the Linux kernel, Linus and his maintainers rule ultimately. Everyone can contribute freely, but as you go up the chain, there is an editorial process, so that the best get the most responsibility and the final say belongs to Linus. Same with Groklaw.

After there were threats and harassment, we had to be less open to the world about certain things, to protect everyone. That's not something open source software projects have to deal with, so the differences that sometimes people comment on are due to that distinguishing factor. For example, at first I'd ask people if they wanted public credit for their work. Lots did. Later, nobody did, but they still worked just as hard. So, internally we knew who deserved the credit and who should get more responsibility, but outside it was not apparent. Like a pool that looks peaceful on the surface but below there are currents flowing in all kinds of ways at once. Groklaw is like that. And it's proof to me that people don't volunteer for such projects out of ambition or a desire for credit. The community continued to work just as hard as before, and for absolutely nothing in return, just to make a difference if we could. Kind of like you see in communities threatened by a flood and they all go out and fill bags with sand.

I sometimes say that if the whole world was like the FOSS community, everything would be better. And I mean it.

What do you think are the lessons that Groklaw holds for open source and collaborative communications efforts in other areas?

That it works just as well for legal research as for software development, so long as you have an editorial process to decide what is accepted and what isn't and as long as you approach your particular task in a pragmatic

way, recognizing that software development isn't like many other types of projects. But what is key is the ability to put together thousands of people all over the world and get them to work unitedly toward a common goal. It's a remarkable thing. I wouldn't have missed it for anything in the world, and I'll never forget it. When Groklaw would win awards, I'd always credit the group, and sometimes people would act like that was just pro forma. It was not. I certainly and absolutely could never have done Groklaw alone. There is a kind of dynamic to a large group that is as powerful as a tornado but in a positive way—when you let people show initiative and they send you their ideas and materials and evidence and personal experience and let them try things. All you have to do is provide a little direction. Sometimes it works, and sometimes it doesn't, but when it works, you can change a little bit of the world. Groklaw indubitably did.

Do you have any future projects, particularly relating to open source or technology, in the works?

My fervent desire is to leave the limelight behind and live a private life again. I always wanted that. Since I never planned for Groklaw to become Groklaw, it was a mixed blessing when it happened. It was fun, it was creatively exciting, and ultimately it was fulfilling in a way that I can't even put into words. Maybe this: I know something I did in this world actually mattered. It's quite a feeling. But as I said, it wasn't a plan, and I certainly have never been ambitious, and I didn't want anything from Groklaw except to be effective. Now that it is, I'm happy and satisfied. I never wanted to be "somebody" and fame repels me, frankly, and I've avoided it. Now, I have an opportunity to go back to my previous personal life, happy in the knowledge that we did what we set out to do. I'll be around in the sense that I'll be in the background until I finish the transition, training the new people, and finishing up the polishing of Groklaw's records. Then, it'll be me on my porch, waving at cars as they go by, and just living a relaxed and normal life again. I've never worked so hard in my life as I did on Groklaw, and I need, really need, to rest up a bit. ■

1. www.groklaw.net/
2. www.groklaw.net/staticpages/indexphp?page=20040923045054130
3. www.groklaw.net/article.php?story=20110409161444432
4. www.groklaw.net/article.php?story=20110515173831922

AN OPEN WORLD: STORIES FROM THE OPEN SOURCE COMMUNITY

EDUCATION

STUDENT PARTICIPATION IN OPEN SOURCE PROJECTS (A PROFESSOR'S PERSPECTIVE)

Heidi Ellis (originally published December 2010)

I must start by thanking Mel Chua[1] for visiting us in Connecticut and for prompting/prodding me to think more deeply about how open source and academia work together to accomplish education. I believe I now have a better picture of student and academic participation in open source projects.

At first look, student participation in open source projects seems like it should be relatively easy to accomplish. Sure, from a teaching perspective there are issues related to selecting a project, learning curve for the project, finding a mentor, identifying ways that students can participate, figuring out how to grade things, and more. But these things are surmountable.

But in recent years, some rocks in the river have appeared that make navigating the current of open source involvement trickier than it first seemed.

When two groups collaborate, they typically do so to accomplish common goals or to work together towards goals for both groups. In this case, the goals of the two environments differ. The open source environment seeks to create a product that meets user needs. The academic environment seeks to produce students with a certain knowledge and skill set. These differences need to be understood in order for academic and open source project collaboration to be successful.

Open source communities would like to see larger numbers of developers contributing to their projects (as would I). And some in the open source community view students as a possible source of future development

(I happen to agree). Academia sees open source as an opportunity for students to gain real-world experience, learn professionalism, and have some evidence of software proficiency that they can demonstrate to potential employers. Making a contribution is helpful, but not essential.

Can open source and academic collaborations accomplish the goals of both groups? I think so, but there are some differences in the environments which present... well, we'll call them "learning opportunities." In order for a particular collaboration to be successful, it helps if both groups understand these differences.

While talking with Mel, it became clear how much the two environments differ with respect to pace, planning, and constraints. The open source way is very opportunistic and flexible, while academia is very planned and structured. The open source way emphasizes short-term optimization and taking immediate advantage of resources (for example, developer expertise, time, or funding). Resources can appear and disappear relatively quickly in the open source environment. With the fluid nature of both resources and participants, it is difficult to estimate long-range (one or more years) results. This is not to say that open source projects do not do long-term planning, but that the development process is sufficiently flexible to allow projects to change paths or goals as new opportunities open up.

Academia is built around concepts of long-term optimization and resources allocated over time. Academics have a fairly fixed set of resources (for instance: time, instructors, students) which vary little over the long term (several to many years). In addition, academics operate under a series of constraints. Academic resources are bound by time limitations, such as semester schedules and class hours. They are bound—obligated—to syllabi, learning outcomes, and grading. These things cannot typically be changed within a three-to-four-month timeframe, and sometimes not within a year. This limits the ability of academia to take advantage of the opportunities that arise spontaneously from open source.

The pace of the two environments is also different. The open source environment tends to be fast-paced and less predictable with an intermittent pattern of effort as people have more or less available time to contribute. Academia has a much slower pace (some might say glacial) with higher predictability.

The academic schedule is certainly predictable. Class schedules are often created six to eight months before the term starts. In addition, curricula plans encompass all four years of a student's stay at an institution. Therefore, classes must be supplied to meet the curricula that was in place at the time that a student entered the institution. In addition, changes in curricula are typically phased-in over a four-year period.

Clearly, there are also large differences in culture. But I think that collaboration between open source and academic realms can work, as there are also some strong commonalities between the groups. The open source and academic environments both share the desire to create something, to produce a product that people will use. Both groups have a love of learning and both groups are based on the idea that something (whether it is knowledge or software) should be accessible to everyone. Both groups have a desire to belong to a professional group, to be interacting as professionals and participating in ongoing professional activity. And interestingly, I think both groups share the

desire to be self-directed and to have control over what they do.

So what else have I learned, as a professor trying to get more students involved in open source? Lots!

Participation in open source definitely benefits students. I have watched students gain invaluable professional knowledge and experience, growing skills and forming professional networks through participation in open source. Many students are motivated by participation in open source projects in a way they aren't in a traditional classroom. They have a better understanding of how the seemingly esoteric things they've learned in their courses matter.

Setting expectations is important. Expectations are important—for both the student and for the open source community. The differences in cultures identified above must be understood by both groups in order to support a successful collaboration. The actual methods and manners of participation in the project may look very different from the academic and open source perspectives.

I can be more opportunistic. My preferred approach is to plan things out well in advance. Talking to Mel made me realize that there were lots of opportunities that occur spontaneously. With little effort, I could take advantage of these opportunities if I'm willing to alter—or abandon—my plan.

For instance, with two days notice, Mel and I set up a Hack Share[2] where we invited Sebastian Dziallas[3] to come hack (live and in-person) and teach students how to package an application. I would not have attempted this on my own, assuming that I would need lead time to advertise, get resources, secure a location—all the details. However, Sebastian's talk was very well attended and a huge success on a small scale.

Could I have gotten a larger attendance? Sure! But not in my window of opportunity. With little time to plan, the Hack Share reached only a small number of people. But if I refused to try because of the immediacy of the opportunity, the event might not have occurred at all. The trade-off is to reach fewer people in smaller ways, but with a larger number of experiences. The conversations I've had with Mel—and the success we had with this quickly formed event—encourage me to take advantage of opportunities that arise.

Academia needs to be sure to give back to the open source community. One very real danger of student participation in open source software development is that students will learn from the community, gain from the community, and then not provide anything back to that community. This violates the open source way and could easily break up open source/academic collaborations. In my opinion, the onus is on professors to find a way to provide some return value to the open source community. This value does not necessarily need to be in the form of code, and could easily take the form of documentation, wiki gardening, or other needed tasks.

I believe that our efforts involving students in open source projects will pay off for the open source community—in the long run . It may be many years before these benefits will be reaped. I say this for several reasons. First, most students are focused primarily on their degree and then on getting a job. These are folks who are (rightly so) spending most of their energy on establishing careers. This means that for at least a year (perhaps longer) after graduation, these folks may not have time to contribute to open source projects.

Second, I believe that students will carry the banner for open source, but that it will take time for the idea to spread. Remember that students are not professionals and they are learning how to participate openly, in addition to the material in all their other classes. They typically have a much longer entry timeframe into open source than an experienced developer.

Lastly, academia moves at a snail's pace compared to the open source world. It will take time for professors to understand the opportunities offered—and the social obligations necessitated—by involving students in open source. And it will take them even longer to change their own classes to include open source; longer still to have open source integrated across a curriculum.

These observations have both positive and negative repercussions for the open source community. The bad news is that there is not likely to be a huge influx of new open source developers—graduating college students familiar with the open source way—in the near future. This is compounded by the fact that the number of computing students has not yet recovered from the steep decline in numbers that occurred in the 2000s.

The good news is that there is likely to be a trickle of these university-taught developers and that this small stream is likely to continue for many years. It is my hope that the stream will grow as word spreads and as more professors adopt approaches involving students in open source projects.

One significant advantage in our efforts to make open source more prevalent on college campuses? The already-growing awareness of open source within the computing student population and beyond. Students are excited by participating in open source, no matter how it's introduced. Hopefully this excitement will catch fire in academia—in the classrooms and beyond. ■

1. www.opensource.com/users/mchua
2. http://opensource.com/education/10/11/open-source-and-student-engagement-explained-5-minutes
3. www.opensource.com/users/sdziallas

THREE UNSPOKEN BLOCKERS THAT PREVENT PROFESSORS FROM TEACHING OPEN SOURCE COMMUNITY PARTICIPATION

Mel Chua (originally published November 2010)

One of the hardest things about trying to bridge two worlds—for instance, open source communities and academic institutions—is all the stuff you don't hear on a daily basis when you're working remotely. Sometimes it takes several rounds of garlic bread and pasta for people to begin articulating what's blocking them from teaching their students how to participate in FOSS communities. Sebastian Dziallas[1] and I sat down last weekend at the 2010 Frontiers in Education conference[2] with a group of professors from the Teaching Open Source community[3]. "What are the biggest blockers that you're facing in doing this," we asked, "that people in the open source world just don't know about or understand?" Here are their answers.

Blocker #1: Intellectual property policies, aka "No, you can't release that under an open license."

At some schools, if you make it on campus, for campus, or with resources from campus,

guess who owns it? Yep: campus. One way colleges and universities make money is "technology transfer," a form of intellectual serfhood—if you're a professor, a student, or a lab, you get resources (students, classes, space, equipment) from the school, but all the IP you produce is owned by the school, so the school takes care of licensing that IP out to companies that want to use it... and keeps the cash.

If you're a school, this arrangement works out in your favor, so you put policies in place specifically preventing students and professors from giving away their "schoolwork" for free, because... well, that's how you make money. The concept of open licensing as a benefit (free marketing!) to the university instead of a drain (giving away precious IP we'd otherwise sell at a profit!) is new to many places, and when you're trying to get a project started for a ten-week class, you can't afford to spend all ten weeks patiently educating university administration about the benefits of licensing (while you simultaneously try to learn data structures in Java).

So that's one bug.

Blocker #2: Student privacy, aka "We're going to make your students fill out forms now before they can release their work for class."

Even if professors (and students) think it would be beneficial for student work and professor feedback on that work to be out in the open where more people can see and comment on and benefit from it, clearance has to be specifically sought because of federal regulations like the United States' Family Educational Rights and Privacy Act (FERPA). These are designed to keep sensitive data about students (read: grades) under their own control. But it's a fine line to walk—can you require people to upload graded classwork to a public server? Can you do your comments and evaluations there? Can you require them to list their names? To work and interact with a community they may not want to work with (for instance, if your class is a requirement, and students aren't there voluntarily)?

Different institutions have different policies, and some professors may not have the time, the legal expertise, the political capital, or the ability to take the risk and step forth for the advocacy this might take at their particular school. When you're at a school to teach students, you want to spend time teaching them, not responding to letters from administrators concerned about families complaining that you're broadcasting their children's private data.

Blocker #3: IT support, or the lack thereof.

People from the open source world are used to the following workflow when they want to show others a new piece of (open source) software:

1. Go to the computer sitting on your desk.
2. Download and install the software.
3. Email your friend the link to your web server.

Professors can do the same thing, but once they want to make that resource available to the students in their classes, they may have to first:

1. Ask IT for an internally hosted box.
2. Wait a while.
3. Try asking, "When can my TA and I have an account on a server? Any server! Any server at all!"
4. Offer, "Yes, yes, I'll administer it myself (in my nonexistent free time)."
5. Fill out more forms.
6. Worry that half the semester is already over.
7. Wonder how much longer this is going to take.
8. ...and so on.

Even if you get IT's permission to try out something, or persuade your students to try out some open source applications on their own, the question then becomes one of support. If your students install Linux and tinker around and crash their computers, IT isn't going to fix it. Students know this and often don't want to take the risk. If they do, and things break, they'll come to you—and so in addition to being a professor, you now get to provide technical support for your entire class for applications you are probably not familiar with debugging.

How can we help?

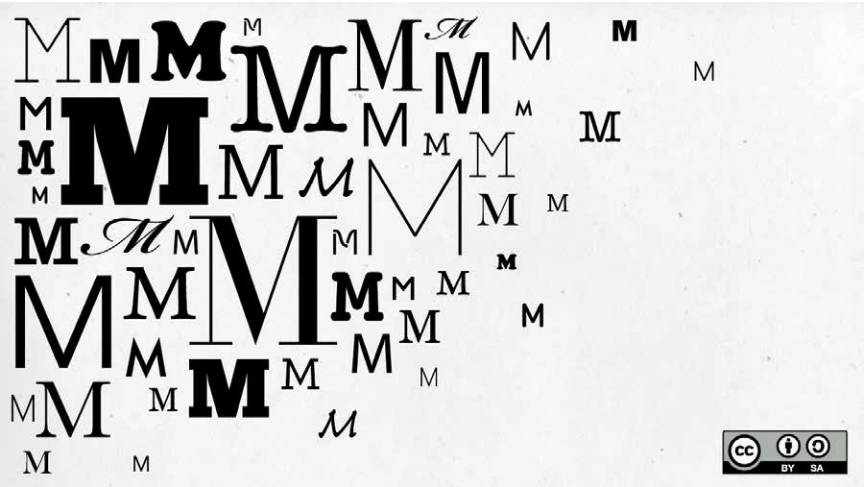
Remember, these comments came from professors who have already fought through whatever they needed to figure out in order to start getting their students involved. These are the people who are already clearing out these blockers—often working for several years to even be able to start to teach their students about FOSS. These professors are still few in number, and the first of their kind, oftentimes standing as the only faculty member in their institution who doesn't think the idea of teaching FOSS is crazy. These people are our allies. How can we help them get past the "community participation bugs" that are stumping them?

Thanks to Heidi Ellis (Western New England College), Matthew Burke (George Washington University), Clif Kussmaul (Muhlenberg College), Greg Hislop (Drexel University), Mihaela Sabin (University of New Hampshire), and Steve Jacobs (Rochester Institute of Technology)—for the discussion that led to these notes, and to Sebastian Dziallas (Olin College) for helping me write them up into this article. ■

1. www.opensource.com/users/sdziallas
2. www.fie-conference.org/fie2010/
3. www.teachingopensource.org/

AN OPEN WORLD: STORIES FROM THE OPEN SOURCE COMMUNITY

HEALTH



JOIN THE M REVOLUTION—GET YOUR TOOLS

Luis Ibanez (originally published March 2012)

The M programming language is also known as MUMPS. Which stands for Massachusetts General Hospital Utility Multi-Programming System. Read my earlier post[1] introducing the multi-user, strongly imperative language designed to manipulate and control massive databases. Then get started using it with this tutorial.

Two main software environments are available today for programming in M:

- Intersystems Caché[2]
- FIS GT.M[3]

You can download an evaluation version of Intersystems Caché[4], but because FIS GT.M is free and open source, we will use it here as the reference system for this tutorial.

The M language has a well defined set of standards:

- ISO/IEC 11756:1999[5]
- ISO/IEC 15851:1999[6]

We will stick to the M-standard in the exercises of this tutorial, therefore the source code examples should work in both GT.M and Caché environments.

Let's focus now on installing GT.M and getting it to work in your favorite Linux installation.

"Every tool is a weapon—if you hold it right."
—Ani DiFranco

Step 1: Download and install GT.M

Download and install GT.M:

```
$ wget http://download.sourceforge.net/project/fis-gtm/GT.M%20Installer/v0.11/gtminstall[7]
```

```
$ chmod +x gtminstall
```

```
$ sudo ./gtminstall —utf8 default
```

Work is in progress to create Debian packages for GT.M (debian-med fis-gtm)[8], and they should be available soon. In the meantime, the instructions above are the most straightforward way to install GT.M in your Linux environment. Note that this installation will use easy defaults. Such an environment will be good for trying out GT.M and for running through the exercises of this tutorial, but it may not be good enough for a production system. Consider this installation a safe sandbox for learning M.

The executables of the installation will, by default, go to one of the following directories (depending on whether you are in a 32-bit or 64-bit architecture):

```
/usr/lib/fis-gtm/V5.5-000_x86/
/usr/lib/fis-gtm/V5.5-000_x86_64/
```

Now we set up the environment variables for GT.M by sourcing the gtmprofile file.

From your shell, do the following:

```
$ source /usr/lib/fis-gtm/V5.5-000_x86/gtmprofile
```

You will see output similar to:

```
%GDE-I-GDUSEDEFS, Using defaults for Global Directory

 /home/ibanez/.fis-gtm/V5.5-000_x86_64/g/gtm.gld

 GDE>

%GDE-I-EXECOM, Executing command file /usr/lib/fis-gtm/V5.5-000_x86_64/gdedefaults

GDE>

%GDE-I-VERIFY, Verification OK

%GDE-I-GDCREATE, Creating Global Directory file

/home/ibanez/.fis-gtm/V5.5-000_x86_64/g/gtm.gld

Created file /home/ibanez/.fis-gtm/V5.5-000_x86_64/g/gtm.dat

%GTM-I-JNLCREATE, Journal file /home/ibanez/.fis-gtm/V5.5-000_x86_64/g/gtm.mjl created for region DEFAULT with BEFORE_IMAGES

%GTM-I-JNLSTATE, Journaling state for region DEFAULT is now ON
```

For the long term, it is convenient to do this from the initialization file of your favorite shell. For example, in bash, add the following lines to your $HOME/.bashrc file:

```
# Set up GT.M environment.

source /usr/lib/fis-gtm/V5.5-000_x86/gtmprofile
```

This adds a set of GT.M-related variables to your environment, and also adds the GT.M executables to your PATH. If you are curious, you may want to take a look at those changes by doing the following in the prompt of your bash shell:

```
$ env | grep gtm
```

Now you can run GT.M for the first time by simply typing gtm at the shell prompt.

```
$ gtm
```

This should open the GT.M prompt:

```
GTM>
```

At this point you can type a couple of verification commands. For example:

```
GTM>write $zversion
GT.M V5.5-000 Linux x86
GTM>halt
```

The "intrinsic special variable" $zversion[9] returns the version of the installed M environment. The halt[10] command stops the gtm interpreter and returns control to the operating system, so you will be back at your shell's prompt.

The initialization process creates a local installation in your home directory under:

```
$HOME/.fis-gtm
```

with the subdirectories:

```
$HOME/.fis-gtm/r
```

```
$HOME/.fis-gtm/V5.5-000_x86
(if in a 32bits architecture)
```

```
$HOME/.fis-gtm/V5.5-000_x86_64
(if in a 64bits architecture)
```

As we write code examples, these are the directories where the code will go.

This is a good point to note that M/MUMPS is a combination of a programming language and a database (as was kindly pointed out by one of the first commenters to our previous post)[11]. We will try to be more explicit going forwards when we are referring to the language versus when we are referring to the database.

Step 2: Testing the installation

We can now write a "hello world" program.

First, set the path to your favorite editor in the "EDITOR" environment variable of your shell. For example in bash:

```
EDITOR=/usr/bin/emacs
```

or

```
EDITOR=/usr/bin/gvim
```

Then from the same shell, invoke gtm, and at the prompt, request to edit the "Hello.m" file:

```
GTM>ZEDIT "Hello.m"
```

This should open the editor program that you just set up in the EDITOR environment

variable, and now you can type in it the following M code:

```
MYLABEL ; This is a comment
WRITE !,"Hello World"
QUIT
```

Note that the second two lines leave one blank space in the first column, while the first line (containing a label) starts in the first column.

Then save the file and quit the editor. Once back at the gtm prompt, type:

```
GTM>ZLINK "Hello"
```

and execute the program by using the DO[12] command:

```
GTM>DO MYLABEL^Hello
Hello World
```

Let's now edit the program again by typing:

```
GTM>ZEDIT "Hello.m"
```

and once in the editor, let's insert another line:

```
MYLABEL ; This is a comment
WRITE !,"Hello World"
WRITE !,$HOROLOG
QUIT
```

Then save the file and link it again with the command:

```
GTM>ZLINK "Hello"
```

It is important to call ZLINK[12] every time that your modify the source code, since it will recompile it and will replace the previous code in the current environment. Now you can execute the new version with:

```
GTM>DO MYLABEL^Hello
Hello World
62520,56765
```

The $HOROLOG[13] special variable returns the date and time as a string value specifying the number of days since December 31, 1840 and the number of seconds since midnight of the current day. (Read why that date was chosen.)[14]

Step 3: Looking under the hood

You may find it interesting to see where the source code and compiled versions of your routines are going. Take a look at the directories:

```
$HOME/.fis-gtm/
V5.5-000_x86_64/r/
```

```
$HOME/.fis-gtm/
V5.5-000_x86_64/o/
```

where you will find the files:

```
$HOME/.fis-gtm/
V5.5-000_x86_64/r/Hello.m
```

```
$HOME/.fis-gtm/
V5.5-000_x86_64/o/Hello.o
```

Acknowledgments

Many thanks to K.S. Bhaskar (Development Director at Fidelity National Information Services, Inc.) for his guidance on fis-gtm and for providing the large majority of the materials presented in this tutorial. All errors that may have slipped above, of course, are mine alone.

References

The complete reference to the M language is available at:

www.tinco.pair.com/bhaskar/gtm/doc/books/pg/UNIX_manual/index.html[15]

The pocket guide to MUMPS is available at:

www.vistaexpertise.net/docs/pocket_guide.pdf[16] ■

1. www.opensource.com/health/12/2/join-m-revolution
2. www.intersystems.com/cache/index.html
3. www.fis-gtm.com/
4. www.intersystems.com/cache/downloads/index.html%20
5. www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29268
6. www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29269
7. download.sourceforge.net/project/fis-gtm/GT.M%20Installer/v0.11/gtminstall
8. www.debian-med.debian.net/tasks/his.fr.html
9. www.tinco.pair.com/bhaskar/gtm/doc/books/pg/UNIX_manual/ch08s49.html
10. www.tinco.pair.com/bhaskar/gtm/doc/books/pg/UNIX_manual/ch06s07.html
11. www.opensource.com/health/12/2/join-m-revolution#comment-9013
12. www.tinco.pair.com/bhaskar/gtm/doc/books/pg/UNIX_manual/ch06s38.html
13. www.tinco.pair.com/bhaskar/gtm/doc/books/pg/UNIX_manual/ch08s05.html
14. www.en.wikipedia.org/wiki/MUMPS#Epoch_choice
15. www.tinco.pair.com/bhaskar/gtm/doc/books/pg/UNIX_manual/index.html
16. www.vistaexpertise.net/docs/pocket_guide.pdf

OPEN SOURCE CANCER RESEARCH

Lori Mehen (originally published December 2011)

When it comes to treating, curing, and preventing cancer, modern medicine has largely failed. You could argue that cancer is far too complicated to unravel in the few millenia we have been documenting it. Or that the billions we spend annually on research is far too little. Established incentives and policies that perpetuate research silos certainly seem to slow success.

Medical researchers have been trained in a professional culture where secrecy reigns, where they must protect their own interests. The dominant culture discourages sharing research findings and collaborating on projects. It has become more important to protect vested interests than to take advantage of the huge collaborative network that is available in academia.

This mode of thinking is a bitter pill to swallow for the quarter of our population that will die of cancer. According to the World Health Organization[1], one in every four deaths is attributable to cancer.

What would happen if cancer researchers were able to adopt an open and collaborative approach like the one that has—for the last two decades—revolutionized software development? What if cancer research could be open source?

Linux has been successful because a large group of people recognized a need and agreed on a process for meeting that need. The brilliance of the open source approach is in the sheer amount of participating brainpower. The open source community shows that the collective intelligence of a network is greater than any single contributor.

While the term is attributed to software development, the idea is not. In fact, some medical research does use this methodology in the same way that Linus Torvalds and others develop open source operating systems. The Human Genome Project[2], for example, very successfully distributed gene-mapping in efforts to speed up the sequencing of the genome. The HGP teams published their data openly, on the Internet.

More recently, a team of Harvard[3] researchers discovered the power of distributed research. A team led by Jay Bradner[4] at the Dana Farber Cancer Institute[5] discovered a small-molecule inhibitor that showed promise in its ability to interrupt the aggressive growth of cancer cells. The small-molecule inhibitor, called JQ1—after Jun Qi, the chemist who made the discovery—works by suppressing a protein (bromodomain-containing 4, or Brd4) necessary for the expression of the Myc regulator gene. It is a mutated Myc gene that is believed to be at the root of many cancers. Without Brd4, Myc remains inactive. Inhibiting Myc could be part of the key to successful cancer treatments.

With the cells from an affected patient, Bradner's group successfully grew the cancer in mice and discovered that the mice with the cancer who received the compound lived, while the mice with the cancer who didn't receive the compound rapidly perished.

Instead of operating in secrecy and guarding their work, Bradner's group shared it. They simply started mailing it to friends. They sent it to Oxford crystallographers, who sent back an informative picture that helped Dr. Bradners team to understand better how the small-molecule inhibitor works so potently against Brd4.

They mailed samples to 40 labs in the US and 30 more in Europe, encouraging these labs to use it, build upon it, and share their findings in return. As a result of this open source approach, Dr Bradner's team has learned—in less than a year—that JQ1 small-molecule inhibitor prevents the growth of leukemia, making affected cells behave like normal white blood cells. Another group reported back that multiple myeloma cells respond dramatically to JQ1. Still another found that the inhibitor prevents adipose cells from storing fat, thus preventing fatty liver disease.

Bradner has published his findings. He has released the chemical identity of the compound, told researchers how to make it, and even offered to provide free samples to anyone in the medical research community. (If you're a researcher who'd like a sample of the JQ1 molecule, you can even contact Bradner's Lab via twitter @jaybradner[6].)

Bradner feels his early successes are due not only to the science, but also to the strategy. Using an open source approach, sharing the information about this molecule, and crowd-sourcing the research and the testing illustrates the opportunities that an open methodology can bring to the difficult challenges of medical research and prototype drug discovery.

In his recently released TED talk video, Dr. Bradner explains that he firmly believes that making a drug prototype freely available among researchers will help accelerate the delivery of effective cancer drugs to affected patients.

With more practice—and more familiarity with each other and this kind of collaborative research—scientists can break large, complex, time-expensive projects into smaller, achievable portions. By spreading

out those small tasks among many groups, much more work can be accomplished in a vastly reduced amount of time.

Using the old research models, Bradner's team might have learned that JQ1 affects AML cells in the first year. But it might have been next year before they got to leukemia, and years after that before they realized it also could affect fatty liver. How many years do you think the old approach adds to the development of drugs we need today?

It is time to seriously consider a different model for scientific research—one that directly engages and benefits society, encourages open access and the free exchange of scientific information. The benefit to patients would be enormous. ■

1. http://www.who.int/mediacentre/news/releases/2003/pr27/en/
2. www.ornl.gov/sci/techresources/Human_Genome/project/about.shtml
3. www.hms.harvard.edu/hms/home.asp
4. www.bradner.dfci.harvard.edu/
5. www.dana-farber.org/
6. https://twitter.com/jaybradner

AN OPEN WORLD: STORIES FROM THE OPEN SOURCE COMMUNITY

GOVERNMENT

HISTORY OF OPEN SOURCE IN GOVERNMENT

Gunnar Hellekson (originally published May 2012)

It is difficult to imagine the Federal government moving in one well-coordinated direction on any matter, and so it has been with the adoption of open source software. Some agencies were early adopters, especially the academic and research communities. As it did in universities, open source adoption in the US government originated in research settings, where sharing and collaboration were already part of the culture of pedagogy. In this way, the government had been using and creating open source software even before it was called "open source." Other agencies and departments have been more conservative, for a variety of reasons, and are only just now bringing open source software into their operations. With this in mind, the history of open source in the US government is best understood as a series of individual stories that have collectively led to the pervasive adoption of open source we see today.

It was in 1997 that open source as an enterprise computing trend emerged, and the US government was there. While Eric Raymond was writing his seminal treatise on open source, "The Cathedral and the Bazaar[1]," a Major in the US Air Force named Justin Seiferth published "Intranet Hallways Systems Based on Linux[2]" in the Linux Gazette. This article described a simple web-based explorer for Windows file servers built on the Linux operating system. This may be the first public acknowledgment of the US

Government's use of open source software as we know it today.

For the next several years, advocates in the private sector and cautious staff in government began to engage the questions that still confront open source today: Is it ready? Is it secure? How do we use it? In 1999, Mitch Stoltz of NetAction wrote the first persuasive essay on the topic, "The Case for Government Promotion of Open Source Software[3]." Stoltz invokes many arguments that are still being used today: lower cost, increased flexibility, and better security. That same year, the President's National Coordinator for Security, Infrastructure Protection, and Counter-Terrorism convened a multi-agency working group to produce "Open Source Code and the Security of Federal Systems." That report is the first official study of open source by the federal government.

While at the Air Command and Staff College, Major Seiferth returns to our history again, this time publishing a research report on the potential benefits of open source specifically in the DOD. Seiferth notes ironically that the US Government is at once reluctant to use open source, and a great creator of open source projects[4]:

"Within the Department of Defense, the National Laboratories and Defense Advanced Research Agency have been the most visible users and producers of open licensed systems. They've released such advances as the original firewall and network security toolkits. As a more recent example, within the last year the National Air and Space Agency has debuted several inexpensive supercomputers. Open licensed operating systems and applications allowed the scaling of inexpensive pentium-based machines into an integrated hardware/software system. In addition to being inexpensive, these machines are among the most powerful available."

Seiferth, like Stoltz, makes a number of familiar arguments for open source, but his greatest insight is that open source is "Commercial Off-the-Shelf" (COTS) software. This is significant, because it means that open source would be able to use the existing policy and regulations that had already been created for software more generally, rather than being treated as a special case and thus hampering its adoption. This will later become the explicit policy of the Office of Management and Budget, as well as the Department of Defense.

The very next year brings an explosion of open source activity in government. In the private sector, IBM announced that they are investing one billion dollars[5] in the Linux project. The Open Source Software Institute[6] was founded to aid the adoption of open source in the Federal government.

Meanwhile, government adoption continues apace. We begin to see the procurement apparatus wrestle with open source licensing in procurements. The US Air Force Scientific Advisory Board's "Ensuring Successful Implementation of Commercial Items in Air Force Systems[7]" is the first procurement guidance to explicitly mention open source.

Some agencies aren't waiting, though. The National Security Agency — to the astonishment of its peers and the open source community – releases SELinux[8], which provided a set of strong security controls to the Linux operating system. In doing so, the NSA was taking technology that had been useful to a very small set of customers, and was therefore very expensive, and made it freely available to the general public. Innovation quickened, the software improved, and SELinux is still used in Linux today. Most recently, SELinux was ported to the Android system[9], where it provides mobile phone us-

ers protections against hostile applications. This wasn't the first time the US government has released software, but it made headlines because it was an implicit endorsement of the open source process by arguably the most security-conscious intelligence agency.

This flurry of activity continues into 2001, with MITRE releasing "Making the Business Case for Open Source Software[10]." This document, the most comprehensive treatment of open source to that point, was published as part of the larger "Open Source Software in Military Systems" study which the US Army had commissioned from MITRE. The report concludes: "Open source will benefit the government by improving interoperability, long term access to data, and the ability to incorporate new technology." Here, we see the US Army, who is later to become one of the largest open source users in the world, taking its first exploratory steps.

The next major milestone is in 2003, with the release of the "Stenbit Memo[11]." On May 28, the DOD CIO John Stenbit released the first DOD-wide guidance on open source software, which implicitly permits its acquisition, development, and use. Meanwhile, the Army begins to deploy the "Blue Force Tracker," running on open source software, to over 80,000 tactical vehicles. Famously, General Nicholas Justice proclaims, "When we rolled into Baghdad, we did it using open source." Nine months later, in July of 2004, the OMB issues a memo similar to the Stenbit Memo that covers the government as a whole. At approximately the same time, NASA releases the very popular World Wind[12] geospatial visualization project under the newly-minted "NASA Open Source Agreement[13]." Six months later, Red Hat, the world's largest open source company at the time, creates a US Government division[14] and the first Government Open Source Conference (GOSCON)[15] is held in Portland, Oregon.

In 2006, Sue Peyton, the Air Force Assistant Secretary of Defense for Acquisition, commissioned the "Open Technology Development Roadmap[16]," which goes beyond the simple benefits of open source, and describes how it can be put to productive use in the context of the DOD's Net-Centric doctrine, which was in fashion at the time. This is the first effort to align the principles of open source with an overall agency strategy, demonstrating how savvy open source advocates inside the government have become.

In 2007, the US Navy commissioned Raytheon, IBM, and Red Hat to add "real-time" features to the Linux kernel[17], which it required for the new destroyer it was building. Significantly, the Navy ensured that the software is released into the open source community. Shortly thereafter, the US Navy CIO Robert Carey releases the Navy Open Source Memo[18], which explicitly classifies open source as COTS software. This is a significant change in tone from the Stenbit memo and OMB memos of 2004, which only implicitly provide this same guidance.

Open source use subsequently explodes. By September of 2008, the Microsoft-funded Open Source Census[19] was reporting that open source use in government was higher than any other industry. The Federal Open Source Alliance's Federal Open Source Referendum[20] study reported that, 71% of agency executives believed they could benefit from open source and 58% said they were likely to consider open source.

The Obama Administration's first act on taking office was to issue the Open Government Memo[21], which articulated a general policy of

"transparency, collaboration, and participation." Subsequent agency initiatives prominently featured open source software as a means to achieve those goals. Open source policies began to pour out of governments at the federal, state and local level. NASA, in particular, made open source software and the open source development process a cornerstone of their open government plan[22]. In the private sector, Open Source for America[23] was founded. This coalition of industry, advocates, and individuals is meant to be a central resource for advocates of open source software in government. That August, Macon Phillips, the White House New Media Director who would later release portions of the software for whitehouse.gov, called open source "...the most concrete form of civic participation[24]." Clearly, open source and open government became inextricably related.

In October of 2009, the "DOD Open Source Memo[25]" is released by David Wennergren, the DOD CIO. This memo got headlines around the world, and remains the single most influential government policy document on open source today. The memo itself is simple, and following the Navy's declaration two years earlier, reminds procurement officials that open source software is COTS. The appendices to the memo, however, go into much more detail about the potential advantages and risks of open source software. The memo specifically encourages the DOD to take advantage of its ability to modify software to suit a mission's need.

Later in 2009, CENDI, an organization of government managers, issues a FAQ[26] on copyright and open source to help agency lawyers understand open source licensing and the sometimes confusing intellectual property questions that they pose. A few months later, for the first time since 2004, OMB refreshes its open source guidance with the "Technology Neutrality[27]" memo, reminding agencies that competition in software is important, and that they are forbidden from discriminating against software based on its development method. Once this memo was published, most of the barriers to open source adoption had been diminished or eliminated in the US government.

Unburdened, open source continued its growth in 2011. Sue Peyton's Open Technology Development Roadmap from 2006 receives a "Lessons Learned[28]" sequel, which makes recommendations to DOD programs interested in releasing their own software. Eben Moglen, one of the most prominent open source lawyers in the country, and head of the Software Freedom Law Center[29], releases "Government Computer Software Acquisition and the GNU General Public License[30]," which explains the provisions of that very popular open source license in the context of government procurement regulations. Clearly, the government's understanding of open source had grown more sophisticated since its first tentative forays a decade before. A survey conducted by Lockheed Martin[31] at this time found that 69% of government contractors and 40% of federal agency respondents were already using open source. The survey also found that 66% of all respondents said that they would be using more open source in the next 12-18 months.

With this increased comfort, 2011 also saw the release of more open source software from the government than ever before. The White House released portions of the code for whitehouse.gov, the code for the Federal CIO's IT Dashboard, and the data.gov platform. At the end of 2011, the Federal

CIO announced a draft "Shared First" policy, which mandates re-use and sharing of IT resources amongst civilian agencies, and specifically mentions that agencies should collaborate on software development[32]. Also, NASA releases code.nasa.gov, a landmark project to centralize all the source code released by NASA in one citizen-friendly web site[33].

So we see the adoption of open source in the Federal government as an evolution: the first furtive steps in the late 1990s and early 2000s, manifested in persuasive essays and studies. From there, certain organizations like NASA and the Army take leadership roles in open source adoptions. From 2003 to 2009, a series of policies institutionalize its use throughout the government. By the close of the first decade, the White House, NASA, the Office of Management and Budget, and other agencies are not just using open source, but creating and releasing open source software of their own.

Did I miss a major event? A major code release? Let me know in the comments.

[This is a writeup I did as a companion to the History of Open Source in Government Timeline[34]. Karl Fogel[35] and I will be presenting more findings[36] from the timeline at OSCON[37] this year.] ■

1. www.catb.org/%7Eesr/writings/homesteading/cathedral-bazaar/
2. www.linuxgazette.net/issue19/hallways.html
3. www.netaction.org/opensrc/oss-report.html
4. www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA398898
5. www.news.cnet.com/2100-1001-249750.html
6. www.oss-institute.org/
7. www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA411926
8. www.selinuxproject.org/
9. www.selinuxproject.org/page/SEAndroid
10. www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/kenwood_software.pdf
11. www.terrybollinger.com/stenbitmemo/stenbitmemo_png/index.html
12. www.worldwind.arc.nasa.gov/
13. www.opensource.gsfc.nasa.gov/nosa.php
14. www.gcn.com/articles/2005/01/21/red-hat-pushes-for-linux-in-federal-market.aspx
15. www.goscon.org/
16. www.acq.osd.mil/jctd/articles/OTDRoadmapFinal.pdf
17. www-03.ibm.com/press/us/en/pressrelease/21033.wss
18. www.doncio.navy.mil/ContentView.aspx?ID=312
19. www.lmaugustin.typepad.com/lma/2008/09/open-source-census-more-numbers-on-open-source-adoption.html
20. www.blogs.the451group.com/open-

source/2008/10/22/goscon-gives-government-good-open-source-ideas/

21. www.whitehouse.gov/the_press_office/TransparencyandOpenGovernment
22. www.nasa.gov/open/plan/
23. www.opensourceforamerica.org/
24. www.dailymotion.com/video/xgh1i3_obama-s-new-media-director-backs-open-source-government_news
25. dodcio.defense.gov/Portals/0/Documents/FOSS/2009OSS.pdf
26. www.cendi.gov/publications/09-1FAQ_OpenSourceSoftware_FINAL_110109.pdf
27. www.cio.gov/documents/Technology-Neutrality.pdf
28. dodcio.defense.gov/Portals/0/Documents/FOSS/OTD-lessons-learned-military-signed.pdf
29. www.softwarefreedom.org/
30. www.acc.dau.mil/adl/en-US/475584/file/60698/OSS%20White%20Paper%2010-11.pdf
31. www.marketconnectionsinc.com/Reports/intersection-of-open-source-and-the-cloud.html
32. www.cio.gov/documents/Shared_Services_Strategy.pdf
33. www.code.nasa.gov/
34. www.atechnologyjobisnoexcuse.com/2011/12/building-a-timeline-of-open-source-in-the-us-government/
35. www.red-bean.com/kfogel/
36. www.atechnologyjobisnoexcuse.com/event/oscon-2012/
37. www.oscon.com/

DOCUMENT FREEDOM DAY: PASSION AND POLITICS

Karsten Gerloff (originally published March 2010)

The battle for open standards in Europe

Today, people and groups around the world are celebrating Document Freedom Day[1]. This is an international day to raise awareness of Open Standards and free document formats. As the event takes place for the third time, the previous focus on the OpenDocument Format[2] (ODF) is broadening to include other free formats such as Ogg Vorbis, and Open Standards[3] in general.

Standards have the reputation of being a dry topic. But Document Freedom Day is inspiring lots of passion and creativity around the world. Volunteer groups from the Free Software scene are using this international day to draw their communities' attention to a topic that most people outside the technology world hardly ever think about.

The campaign is coordinated by the Free Software Foundation Europe[4], but the passion and effort in cities around the world are local. In Romania's capital, Bucharest, a group of activists visited[5] a number of government buildings, each time telling the authorities that "I can't read your documents". In South Africa, the Department of Arts and Culture is holding a celebratory hour. In Buenos Aires, Argentina, eight organizations are organizing[6] an evening of information and discussion about Open Standards. In many countries, as in Vietnam, local groups are setting up information campaigns in universities and elsewhere.

Cakes appear to be a particular favourite. FSFE groups are awarding two of them to German and Austrian radio stations that offer their streams in Ogg Vorbis. A third one goes to the Slovenian Supreme Court, which has adopted ODF as its default document format.

Spreading, but not without a fight

Over the past years, numerous countries [pdf][7] have adopted policies on Open Standards. The Netherlands[8] lead the way, by mandating that public bodies use Free Software and Open Standards from May 2008. Many others have followed, such as South Africa, Japan, Brazil and a number of European countries.

Denmark[9] is the latest nation to join the group, requiring its public bodies to start using ODF for its documents from April 2011. There are differences between all these policies, and they are being implemented with varying degrees of success. But the direction is clear: The public sector is moving to Open Standards. Not without a fight, though.

Europe in the lead

It is striking that out of 11 out of 18 countries that have adopted ODF for their public sector (according to the ODF Alliance) are in Europe. While multiple factors are involved here, such as relatively high market shares for Free Software, one element is crucial.

In 2004, the European Commission issued a recommendation known as the European Interoperability Framework[10] (EIF). The document's stated goal is to promote interoperability between public bodies in Europe, with a view to delivering "pan-European eGovernment services".The EIF's means of choice are Open Standards and Free Software. Crucially, the text contained a relatively strong definition of what an Open Standard is.

The European Commission complemented the EIF recommendation with the OSOR[11] project. The not-so-snappily titled "Open Source Observatory and Repository" quickly became a central platform for public bodies across Europe to learn from each other about their experiences with Free Software and Open Standards. The portal also allows public bodies to upload and share Free Software which they have developed themselves.

[Disclosure: From late 2006 to mid-2009, I worked for a contractor of the OSOR project, UNU-MERIT[12]. One of my tasks with OSOR was to write case studies about the use of Free Software and Open Standards in the European public sector.]

Diverging views in the European Commission

Beyond the OSOR project, different parts of the European Commission have very different views on Open Standards. The Informatics department, responsible for the Commission's internal IT systems, has long relied on framework contracts with Microsoft. In an ironic twist, the unit running the OSOR project is currently a part of this department.

On the other hand, the (now former) competition Commissioner Neelie Kroes went out of her way to highlight the importance of Open Standards. During an event on June 10, 2008 she remarked[13]:

As purchasers, we need to be smart when we buy technology. We need to be aware of the long term costs of lock-in: you are often locked-in to subsequent generations of that technology. There can also be spill-over effects where you get locked in to other products and services provided by that vendor. That is just bad purchasing.

She added:

But there is more to this than ensuring our commercial decisions are taken in full knowledge of their long term effects. There is a democratic issue as well.

[...]

I know a smart business decision when I see one—choosing open standards is a very smart business decision indeed.

Right ahead of Document Freedom Day[14], those tensions are coming to a head[15]. The Commission is developing two very different draft documents that will have a profound effect on the use and spread of Open Standards and Free Software in Europe, and possibly elsewhere.

The European Interoperability Framework: Revised into oblivion

The European Interoperability Framework (EIF) is currently being revised. The process to update this key document started in 2006, with a public consultation held in the summer of 2008. The document which the EC presented for comments still contained a strong definition of Open Standards, and gave Free Software a crucial role in providing interoperability in the public sector.

After the public consultation, EIF version 2 disappeared into the dark interior of the EC. Since then, two drafts have leaked: One in November 2009, and one in mid-March 2010. Astonishingly, these drafts no longer contain a definition of Open Standards. Free Software as an enabler of interoperability has virtually disappeared. This reflects the comments made by the Business Software Alliance[16], a Microsoft-backed lobby group. FSFE maintains a comparison page[17], showing how key parts of the text have evolved since the consultation process.

FSFE and other groups have highlighted[18] both the substantial problems of the text, and the lack of transparency of the process in which it was created. Confronted with the latest draft, we are now asking[19] the Commission to go back to the drawing board, and start over based on the consultation draft from Summer 2008. In its present form, the text would only cement the status quo.

The Digital Agenda: Standardization power struggle

The second document at the center of current debates is the "Digital Agenda." This is a relatively short text, setting out the Commission's policy on all things digital for the coming five years. It is prepared by the Information Society department, which Neelie Kroes is now heading. Though the document hasn't been published officially, the parts concerning Open Standards are available here[20].

According to this Digital Agenda, the European Commission would "issue a recommendation to streamline the use of Open Standards in public services and public procurement". It would also "[u]pdate the European Interoperability Framework to promote an open approach to technology and interoperability".

Both things would be very good for European citizens and their public authorities, since they would increase the use of Open Standards and, as a consequence, Free Software. So it is no surprise that there is now a fierce lobbying battle raging around the text, since it would make life a bit more difficult for the companies that currently dominate the software market with their proprietary applications.

Yet the Digital Agenda is under attack from another angle as well. It calls for a reform of the European standardization system, so that standards coming out of ICT fora and consortia such as OASIS would be recognized. This sits very badly with those departments of the EC that are currently in charge of stan-

dardization: Enterprise and Internal Market. To them, this agenda threatens to take away part of their portfolio and power. Incidentally, those two departments also don't think that recognizing standards prepared in ICT fora and consortia is a good idea.

These issues are moving quickly, with new developments and rumors coming out of the European Commission almost every day. Together with other groups, FSFE is working hard to preserve the Digital Agenda's push for Open Standards.

DFD worldwide—you're not alone

All this shows that the gains that Open Standards have made can't be taken for granted. Lobbyists for proprietary software companies are chipping away at them every day, exploiting internal differences within the European Commission as they go.

Incidentally, an Italian court ruled yesterday that public authorities in Italy's Piedmont region can legally maintain a preference for Free Software in their purchasing decisions. The court considered that such a requirement refers to a characteristic of the software, rather than to a specific product or technology.

This should give a further boost to public bodies that want to use Free Software and Open Standards. It should also remove an obstacle for those that are interested, but haven't yet made the jump.

In this context, Document Freedom Day[21] is a day of hope. It shows that people around the world are passionate about Open Standards, Free Software, and the freedom to use technology as they wish. Governments in Europe and elsewhere should take note. ■

1. www.documentfreedom.org/
2. www.en.wikipedia.org/wiki/Odf
3. www.fsfe.org/projects/os/def.en.html
4. www.fsfe.org/
5. www.nicubunu.ro/documentfreedomday-2010/
6. www..vialibre.org.ar/2010/03/28/dia-mundial-de-los-documentos-libres/#more-5246
7. www.odfalliance.org/resources/Adoptions-ODF-2010-Feb.pdf
8 www.computerworlduk.com/toolbox/open-source/open-source-business/news/index.cfm?newsId=6677
9. www.h-online.com/open/news/item/Denmark-to-implement-ODF-document-standard-918962.html
10. www.ec.europa.eu/idabc/en/document/3473/5585#finalEIF
11. www.osor.eu/
12. www.merit.unu.edu/
13. www.europa.eu/rapid/pressReleasesAction.do?reference=SPEECH/08/317&format=HTML&aged=0&language=EN&guiLanguage=en
14. www.documentfreedom.org/
15. www.h-online.com/open/features/Watering-down-European-standards-966955.html
16 www.bsa.org/
17. www.fsfe.org/projects/os/eifv2.en.html
18. www.fsfe.org/news/2009/news-20091127-01.en.html
19. http://blogs.fsfe.org/gerloff/?p=324
20. www.davidhammerstein.over-blog.com/article-digital-commissioner-kroes-proposes-eu-policy-of-open-standard-46997444.html
21 www.documentfreedom.org/

IMAGE CREDITS

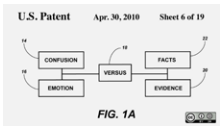
Cover image
www.flickr.com/photos/ashleighb77/3708369320/

The day TuxPaint became contagious
www.opensource.com

Introducing students to the world of open source: Day 1
Asheesh Laroia

The four capital mistakes of open source
www.flickr.com/photos/dahlstroms/3861945279
www.flickr.com/photos/pinksherbet/4812267249/

Rethinking office design
www. opensource.com



Total victory for open source software in a patent lawsuit
www.opensource.com

Interview: PJ on the beginning, ending, and future of Groklaw
www.cafepress.com/groklaw.154236618

Student participation in open source projects
(A professor's perspective)
www.flickr.com/photos/11755880@N00/3856681802/

Three unspoken blockers that prevent professors from teaching open source community participation
http://www.flickr.com/photos/montypython/4074525329/

Join the M revolution—Get your tools
www.opensource.com

Open source cancer research
www.flickr.com/photos/cmrf_crumlin/4838073754/
in/photostream/

History of open source in government
www.opensource.com

Document Freedom Day: Passion and politics
www.documentfreedom.org/Artwork2010

Download an electronic copy of this book at www.opensource.com/best