Open Source Philosophy

Timothy Daly

February 5, 2004

Abstract

These notes are from the first lecture from a course on Open Source Programming. The lecture is about the philosophy and ideas underlying Open Source programming. A broad range of non-programming topics are introduced and illustrated with "war stories".

Contents

What matters most is what you do for free – Gorka

1 Open Source Sea Change

1.1 Internet Effects

By design the Internet is a neutral, peer-to-peer worldwide network with information served both from end points as well as internal nodes.

The "First Industrial Era" was born out of World War I. Large numbers of men needed to cooperate both inside the army and in business in order to win the war. Thus business was structured very similar to the army and continues to be structured that way to this day. Other innovations included factories, limited liability, trade unions and newspapers. People were easily replaced until the trade unions forced changes in working conditions.
(WW1 story) [1]

The "Second Industrial Era" introduced the ideas of a publicly traded corporation, commercial banks, professional women, and business schools. People expected to be hired, trained, and employed for a lifetime and then retire.

The "Third Industrial Era" is today. You personally are your own corporation. You will end up switching jobs every 18 months. Your "resume" will be the sum total of your interaction on the web.

1.2 Closing the Gates

Back in the dawn of computer history all software was open source. When I was at school I had access to the source code for the operating system, the compilers, and just about anything else. Of course, they were on magnetic tape and I needed to find the IBM product code, the library tape number, request and schedule mounting the tapes, write a program to read the tape and print it, and schedule my "job" so it ran while the tape and the tape drive were available. CPU time was charged by the microsecond [5], memory was charged by the number of bytes used times the time of use, disk space was charged by the sector, tape mounts were charged by the minute, programs were charged by the punched card and the printouts were charged by the page. But the source code was free and open source.

Around the time the "Personal Computer" was being developed the idea of charging for software. It wasn't entirely novel (University of Waterloo charged for the Watcom compilers) but it was not pervasive until the PC arrived. Bill Gates didn't invent charging for software. (He did invent making his software required as far as I can tell). However, the PC really opened the market for proprietary software. There was a booming market in paid software. Which led to a booming market in "cracked" software. Which led to a booming market in anti-copy software. Which led to a booming market in "cracking software". It was a regular technology arms race. (sound familiar?)

Richard Stallman [2] tells the story of trying to fix a printer driver around this time. He couldn't get the software which annoyed him. Eventually he decided that software should be free. And the rest, as they say, is history.

So now we are in a situation where the world is beginning to see the value of free software. In some sense it is like the value of free science.

2 Opportunity

2.1 The World from your Couch

From your living room couch it is now possible to affect the world overnight. This is a new thing in the world for most people. The internet has made it possible to modify software used by virtually everyone.

2.2 Self-hiring

There are over 50 thousand open source projects on dozens of sites. Some of the sites are specific to a language (e.g. only Java applets) while others such as Sourceforge [3] and Savannah [4] list thousands of projects. Besides writing the software there are sites which only distribute packaged forms of software (such as rpms).

Almost every project needs volunteers. Many have died because the original developer gave up on the project. Others move very slowly because the original developer does not understand the details necessary to make the next step (e.g. how to package the software into an rpm). Many are active and all you have to do is start working on them. Fedora, RedHat's new open source project, has no official "developer" status and anyone can work on it even though Fedora is a major distribution from the largest open source software company.

So you could, without asking anyone's permission, become a developer on the world's largest software distribution before the day is finished. Within a few days you could perform a valuable service. For instance, Fedora needs quality assurance testers. That basically means that you have to download, install, and try to run various pieces of software. If you find a bug you report it. If you find enough bugs and learn to file sufficiently helpful bug reports, your name gets known as a useful tester.

Or perhaps you speak two languages. You could pick projects in the Fedora distribution, modify their error messages to be in your second language. Soon you will be known as "the internationalization guy on Fedora".

The key point is that you decide to participate. You hire yourself.

2.3 Reputation Building

Having been hired you live and die on your reputation. It is important to "find your level" without making a lot of noise about it. You need to make your contributions speak for themselves. No matter how good you are at your skill you won't be able to argue your way into a good reputation.

And a good reputation is everything. It is your resume. Be careful what you say and how you say it because "the net" never forgets. Reputations are built up over time based on perceived values just like resumes are built over time based on employment. People like Linus Torvolds, Richard Stallman, and Eric Raymond have sterling reputations and have earned them over many years of hard work.

In short, you won't be a star overnight.

3 Culture

3.1 Mythical Man-month

When IBM decided to build its next generation it estimated the size of the project in months, decided on a schedule, and figured out that they needed a thousand programmers. So they hired them. Does it surprise you that the project was late? The whole idea of the man-month arises from traditional construction projects and was transplanted "by analogy" to software construction. Needless to say, it was a horror show of a project. Unfortunately this is not the only disaster for software based on bad analogies. (The whole analogy to "property" continues to do damage today).

The Mythical Man-Month [5] is required reading. It deals with the issues in managing large projects. There is much wisdom to learn from this little book. For instance, when you graduate and get a corporate job your manager will ask you for a schedule for your project. The question seems innocent enough and you'll probably do your best to give an accurate answer.

You just made your first software mistake. After 34 years in this business I know of *no-one* who can create an accurate software schedule. You schedule will be wrong. Worse yet, it will be wrong on the *low side* by *way* too much. But now your manager has your promised delivery date and he will give it to the customer. You'll be late, the project will be late, the customer will be angry and you'll be fired. There is much to learn. Lesson number one is never, *never* give a schedule for software.
(comcast story)

3.2 Scratch an Itch

Programmers generally like to learn things. They like to fix things. This leads to one of the key reasons that programmers work on projects. They get a piece of software and it doesn't do what they want. So they either (a) start from scratch and rewrite it or (b) get the source and fix it.

3.3 Gift Culture

Open source programming is a gift culture (Eric Raymond [6]). The more you give away the richer you are. The richest people give away everything to everyone all the time. They debug other people's programs. They write

their own programs and give them away. They create new subsystems for other projects. They "straddle" two similar projects and bring them into cooperation.

They lead, they follow, they help, they communicate, participate, and cooperate. They are a godsend to every project they touch. They know people because people want to know them. So they can act as a point of connection, as a "friend of a friend". They can always be reached. They always answer their mail. They work hard for important causes and are known for their work.

Think of Richard Stallman. He wrote huge piles of code. In fact, he wrote or helped write a lot of the code you call "linux" as part of the "GNU" project. He's always being asked to give talks (he was speaking to the president of India this week). He is always on the road. Yet every email I ever sent him (and I've sent about a dozen) always, *always* gets an answer. And he programs while he's on the road. He's a very rich man in the gift culture even though he's not financially well off.

They say that in the land of the blind the one-eyed man is king. In the gift culture, Mother Teresa is queen.

3.4 Continuous Learning

So, you want to do open source programming? The open source "scene" (to use a term from the 1960s) is always in motion. You have to learn every day. You have to try to use new tools. You have to use a new language, learn a new skill. You have to dive into projects you don't understand and "come up to speed" very quickly. You have to be comfortable with being lost, with making public mistakes, with getting "flamed" for "newbie" errors. It's all part of the learning process.

In a corporate culture you can "sort-of" learn a skill, say, networking. You play at the skill for a few years, get some seniority (i.e. survives a few downsizings) and retire into management. You don't have to be good at your skill. You don't have to learn new things. You don't have to have any management training. You don't even have to learn how to manage. You just have to survive job cuts.

There are no "retirement" jobs in a gift culture. You have to learn and get rich or you get left behind and drop out.

3.5 Websites

As part of the continuous learning you need to read a lot every day. I read most of the articles on Slashdot [7], virtually every article on The Register [8], about half the articles on Newslinx [9], and every word published on Groklaw [10] every day. And once a week I read Linux Traffic [11]. Plus I subscribe to the Fedora-developers mailing list so I can "get up to speed" on this distribution.

3.6 Foo

Don't be a l4m3r. Know the words of the culture, what they mean and how to use them. Read the Jargon File [12]. Unlike the real world you don't ever get to verbally speak to other members of projects. So you don't know the jargon. And it shows up in your email. You don't "look for it on your desk", you "grep your external cache".

4 Join vs Make

4.1 The Simple Idea

Every program is built on a simple idea. You have to see thru all the junk and figure it out. The best way to do this is to write a little program that will do the same thing. An example is a mail client program. You can write the simple idea in a working program in just a few lines. The *Fetch* program given later will show you a working example.

The simple idea for most programs generally has dozens of features and optimizations added to it. But it also probably has a standard definition someplace. Mail clients follow an RFC standard [17]. You should get a copy of the standard and read it. In fact, while you're there you should read many of the other standards. Did you know there an RFC standard for sending mail by carrier pigeon (RFC 1149 [18])

4.2 The 3x Rule

Projects take much longer than you would expect. Very much longer. There is a famous "3x rule" which is very close to the actual time projects take for each phase. Assume that you start a project and get it to work. Call this effort 1 unit. Assume it took you a month. How long will other parts of the effort take?

1*3 = 3 units: to give the project to your co-worker
You have do clean your project up, put it somewhere online, and give access to it. You need a simple "cheat sheet" of what to type to start it, stop it, and do a simple example. You need to answer the endless little support questions.

1*3*3 = 9 units: to give the project to the department
You have to really clean up your project, package it up in a zipped form, put it on a department server, advertise it to the department, write up some documentation, get it to run on a strange machine, field duplicate questions from many people.

1*3*3*3 = 27 units: to publish it as "shareware" on the net.
You have to put up a website, package the file for download, write detailed html pages, handle server outages, handle backups, get it running on Macs, set up and manage source code control with bug tracking, and field duplicate questions on mailing lists

1*3*3*3*3 = 81 units: to make it into a commercial product
You need a trademark, you need a lawyer, you need a company, you need financing, you need advertising, you need a sales force, you need hosting services, you need a support staff, you need a shrink.
(Eclps story)

4.3 Logarithmic Developer Effort

If you decide to be a lead developer on your own or some inherited open source project you should be aware that 90be yours. Of the remaining 10Of the remaining 1workers get scarce after that. Nobody will tell you this. Of the 50,000 projects there are probably 50,000 active developers. Most of the projects are idle, some of them have a single developer. Several have two or three developers. Really big projects have 10 developers.

4.4 Enter Late and Dominate

Mezick's Theorem [13] states that you should "Enter late and dominate". As the saying goes "You can tell the pioneers. They are the ones with the arrows in their backs." Large companies know this. Watch what Microsoft does. It lets a new company break a new market. Microsoft partners with the new company, promising that the new company will be their "go to market" company, the "point of the spear." Microsoft promises cash (in small increments) so the small, successful company can get bank loans and grow quickly. If the product succeeds Microsoft withdraws the money, keeps the technology gained thru the partner contract, integrates the technology into Windows, and lets the small company fail. No risks, a prepared market, and a big upside gain.
(Doublespace story)
(Sendo story) [21]

5 Law

5.1 Property vs Speech

(candle analogy story)
(DeCSS story) [22]

5.2 Property vs Rights

Weber [1] points out that theories of property revolve around a collection of "rights" held by the "author" of a work including the right to access, the right to extract, the right to sell and the right to lease the work.

In recent years the theory of property has been applied, somewhat incorrectly in my view, to software. The lawyers are reasoning by analogy of software to books. The analogy is weak at best and laws are being written that ultimately cause problems for all. Software is not a book even though you can print it. In

fact there is nothing that is similar enough to software so that a law written for another area can be readily applied.

That has not stopped the congress from trying since there is a great deal of money at stake.

5.3 Trademarks

Trademarks are a legal monopoly on the use of a term or symbol in commerce. Axiom was a trademark for a computer algebra system sold in England. The Axiom trademark cost about $20k to search and file. Trademarks are intended to protect the public from "confusion". Thus, Coke is a trademark which Pepsi cannot use.
(fedora story)

5.4 Copyright

Copyright occurs "at birth" of a work and extends for 95 years. Copy rights include control over the right to access, to extract, to sell or lease other rights.

Lawrence Lessig [14] has a blog that covers this issue on a daily basis and is worth reading. (Mickey Mouse story)

5.5 Fair Use

Fair use applies to things like copies. It is fair use to copy an article for your own education. It is fair use to quote from a book for scholarly purposes. Fair use is a limited grey area of copyright that is being lost as you read this. Digital Rights Management technology, such as in the Apple iTunes music, is preventing fair use.

5.6 Patent

Patents apply even in ignorance. So if you write a program, say a blackjack card game program, you will find out that you just violated a patent on computer card game programs. You will be sued even though you didn't know you violated a patent and defending the suit will cost you millions.
(IBM vs SUN story)

Patents have the potential to cause huge damage to the free software industry. You need to become aware of the patent threat. You need to become involved in fighting against software patents. You need to do this now. Europe is about to vote on software patent laws.

5.7 Trade Secret

Trade secrets apply to software only if you tell no-one how you do something. Coke is a trade secret. Trade secrets are lost as soon as someone tells someone

else. In order to protect trade secrets you require protection and paperwork a legal trail with non-disclosure agreements.

Trade secrets are lost if they are discovered independently. You have to show somebody told somebody. They are allowed to guess.

5.8 License

The GPL [15], Modified BSD [16], and similar licenses are based on using the copyright rights but exploiting the copyright holder's right to distribute rather than the right to exclude.

Open source has whole religious sects based around various licenses. Endless discussions and flame wars occur between non-lawyers that last for days at a time. The famous joke is:

```
I was walking across a bridge one day, and I saw a man standing on the
edge, about to jump off.  I immediately ran over and said "Stop!  Don't
do it!"

"Why shouldn't I?"  he said.

I said, "Well, there's so much to live for!"

"Like what?"

"Well ...  are you religious or atheist?"

"Religious."

"Me too!  Are you Christian or Jewish?"

"Christian."

"Me too!  Are you Catholic or Protestant?"

"Protestant."

"Me too!  Are you Episcopalian or Baptist?"

"Baptist."

"Wow!  Me too!  Are you Baptist Church of God or Baptist Church of the

Lord?"

"Baptist Church of God."
```

```
"Me too!  Are you Original Baptist Church of God, or are you Reformed
Baptist Church of God?"

"Reformed Baptist Church of God."

"Me too!  Are you Reformed Baptist Church of God, reformation of 1879,
or Reformed Baptist Church of God, reformation of 1915?"

"Reformed Baptist Church of God, reformation of 1915!"

To which I said, "Die, heretic scum!"  and pushed him off.
```

5.9 Code is Law

Lawrence Lessig [14] makes the point that "code is law". You can see this effect when you call the bank and ask them to do something that their software won't do. Suddenly you get the excuse "but the computer won't let me". Code is law.

5.10 Intellectual Property

Intellectual Property is a term that is widely used to describe the collection of things covered by trademarks, patents, copyrights, trade secrets, and licenses. In fact there is, as far as I can tell, no legal definition of Intellectual Property. It is a marketing term. If you hear someone talking about their Intellectual Property you can rest assured that they are not lawyers and should be ignored.

Not withstanding that fact you should learn as much as you can about the various legal issues. They *are* important and they will cause you much grief.
(IBM story)
(Centreport story)
(SCO Story)

6 Ethics

Given that you are working for yourself you have only your own moral and ethical standards to guide you. That would seem to lead you to believe that you can do anything and no-one can stop you. Well, just to put a check on your imagination try to remember that everything you do on the net is being recorded for all time. It's your reputation to make or break.

6.1 Libre vs Gratis

The current term in vogue is "free and open source software" (FOSS). The "free" portion refers to the idea that is embodied in the GNU Public License (GPL). The "open source" portion refers to the idea embodied in the Open Source license. They are not the same.

Richard Stallman wrote the GPL. He tried to make sure that

> everybody could modify, study, use, and change software
>
> if you change it you pass on your changes.

To do this he wrote a license that states that

> if you modify GPL software **and**
>
> you distribute the modified software
>
> then you have to make your changes available

This essentially guarantees that the user of your software has the same rights you have.

When you hear the term free think "free as in speech" (libre, frei) rather than "free as in beer" (gratis, kostenlos).

Open Source software is an attempt to make the GPL more "business friendly". It basically requires you to make the source code available. It is a weaker (in a requirements sense) license than the GPL.

6.2 Credit

Give it. Never take it. And never take it off someone else. Credit has certain properties:

> credit trivial to steal
>
> credit is "coin of the realm" (IBM robot story)
>
> credit is cheap to share
>
> credit is basis of community

7 Standards

The open source community has certain standards. It is not a lawless land even though it appears to be to those who don't "get it".

7.1 Technical: What Can I Do

Open source software follows certain community expectations. For instance, you are expected to use certain kinds of tools such as *automake* and build packages using certain tools such as *RPMS*. This course will teach you these expections and these tools.

7.2 Ethical: What Should I Do

Along with the technical expectations there are moral and ethical expectations. These are not usually stated but you need to develop a strong sense of "right vs wrong".

8 Development Model

8.1 Communication

8.1.1 Etiquette

RFC 1855 (Netiquette Guidelines) [19] is a standard for etiquette on the internet. You should read this guideline. When email was first used it was standard practice to send someone a copy of this. Now with email so widely used it is assumed that everyone has read it.
(salt shaker story)

Mailing lists have their own form of etiquette. You have to be careful to follow the purpose of the list. Posting off-topic questions will quickly earn you a reputation, one you will not soon live down.

Newsgroups have thousands of readers worldwide. Generally there are a few "gods" on each newsgroup. For instance, in sci.math.symbolic, you shouldn't argue with Richard Fateman unless you are very, very sure of your point. Richard has been around forever and has seen it all. Every newsgroup has at least one "god". For example, Dan Barlow on comp.lang.lisp. Listen; read a lot of entries. It quickly becomes obvious who knows and who doesn't.

8.1.2 Task Volunteering

Advocacy is volunteering. If you are making an impassioned plea for a feature, a fix, or your latest idea you'll probably convince everyone on the mailing list. Well reasoned, passionate arguments tend to win. But wait! Who is going to *implement* your change? Surely you don't think that anyone else cares enough about it. That leaves only you. So next time you get to dancing and singing on a mailing list be aware that you are volunteering to do *your* task.

You don't program therefore you can't contribute? Surely you jest. In any effort of any size there are dozens of tasks that do not involve programming. A couple instances suffice to make the point:

> documentation. There isn't a project on the planet that is fully and carefully documented. If you can write you can document. Even if you write badly at least there will be a starting point for documentation.
>
> translating. You speak two languages or maybe you can read english but are a native speaker of another language. Pick any project. They need you to work on "internationalization", that is, translating all of the menus, messages, and documents to your native language.
>
> porting. Many porting tasks simply require access to a machine. For instance, you just bought a new 64-bit computer. Pick a bunch of projects, compile them, and try to run them. File bug reports with detailed console logs.

packaging. Linux distributions like RPMs, Windows like InstallShield. If you have access to InstallShield and there are projects that can run on Windows you can build installation packages for them.

Projects everywhere need quality assurance testing. All you have to do is download the project, build it, play with it, and file bug reports. Speaking of bug reports, many projects needs a "bug administrator". They need someone to nudge people about the high priority bugs. They need someone to listen to the mailing list and file bugs that users complain about but fail to insert into the bug database.

Projects need to be pried out of the developer's hands. A programmer never believes his project is ready and certainly it is never finished. But it pays, especially with a new project, to follow the motto: "Release early, release often". That way you get feedback about what is right and wrong about the project. You can lead the way by setting up an early release plan and focusing the efforts to get a working
(comcast story)

9 Money

How do you make money in open source?

I claim that you have to earn "creative dollars" first. Many years ago I saw a survey where they asked management and workers what was most important to workers. The results said:

	MANAGEMENT	WORKER
1	Money	Creative Work
	...	...

Creative dollars are what you earn by building up your reputation on the net. You have to give away what you know and contribute what you can. In short, you have to build up a track record and prove you are worth hiring. Real money-making businesses will pay attention. They will hire you based on your reputation and pay you what you are worth. You must prove your worth first.

And then there is "knowledge production" (e.g. science) where you earn money for contributing what you know to the common good. This is the academic path. You can begin to see this showing up on the MIT website [23]. MIT is giving away it's teaching efforts in order to become rich in the gift economy.

10 Time Independence

You can reach out and touch the world from your couch. And The world can reach in and touch you. If you work in open source you work in a world where people are awake while you sleep. Thus you get email 24 hours a day, 7 days a week. There are no set working hours. It can be hard to adapt your lifestyle to

this. Programmers have addictive personalities and will tend to work 18 hours at a single stretch and then do nothing for two days.

11 Place Independence

And the world is everywhere so you're unlikely to ever meet the people you "work with". You have to get used to the fact that people have a different native language (so their emails are hard to read) and have a different agenda than you have (so you can't seem to get them to "fix" things your way). Since you interact with them in email you have to be very careful. You have to always be civil, you have to always answer their emails, you have to try to work as hard as you can to give a quality answer. It can be very slow and time consuming to interact by email but it is in the nature of open source to be worldwide.

References

[1] Weber, Steven, *The Success of Open Source* Cambridge University Press (2003) ISBN-0-674-01292-5

[2] Stallman, Richard, **"http://www.fsf.org/philosophy/philosophy.html"**

[3] Sourceforge, **"http://sourcforge.net"**

[4] Savannah, **"http://savannah.gnu.org"**

[5] Brooks, Frederick, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, Reading MA (1975)

[6] Raymond, Eric *The Cathedral and the Bazaar*, **"http://www.firstmonday.dk/issues/issue3_3/raymond"**

[7] Slashdot, **"http://slashdot.org"**

[8] The Register, **"http://www.theregister.co.uk"**

[9] Newslinx, **"http://www.newslinx.com"**

[10] Groklaw, **"http://www.groklaw.com"**

[11] Kernel Traffic, **"http://kt.zork.net/kernel-traffic/latest.html"**

[12] Jargon File, **"http://catb.org/ esr/jargon/html"**

[13] Mezick, Dan, **"http://www.newtechusa.com/ViewPoints/DominateLate.asp"**

[14] Lessig, Lawrence, **"http://www.lessig.org/blog"**

[15] GPL License, **"http://www.fsf.org/licenses/licenses.html"**

[16] Modified BSD, **"http://www.fsf.org/licenses/licenses.html"**

[17] RFC 1939, *"Post Office Protocol - Version 3"*, **"http://www.faqs.org/rfcs/rfc1939.htmlp"**

[18] RFC 1149, *"Standard for the Transmission of IP Datagrams on Avian Carriers"*, **"http://www.faqs.org/rfcs/rfc1149.htmlp"**

[19] RFC 1855, *"Netiquette Guidelines"*, **"http://www.faqs.org/rfcs/rfc1855.html"**

[20] Bastiat, Frederick, *"That Which is Seen, and that Which is Not Seen"* **"http://www.jim.com/seen.htm"**

[21] Phone Maker Sendo to sue Microsoft **"http://archive.infoworld.com/articles/hn/xml/02/12/23/021223hnsendo.xml"**

[22] Electronic Frontier Foundation **"http://www.eff.org"**

[23] MIT Open Courseware site **"http://ocw.mit.edu"**