

# Logiciel Libre

## Cours 6 — The Cathedral and the Bazaar

Stefano Zacchioli  
zack@pps.univ-paris-diderot.fr

Laboratoire IRIF, Université Paris Diderot

2016-2017

URL <http://upsilon.cc/zack/teaching/1617/1oglib/>  
Copyright © 2014-2016 Stefano Zacchioli  
© 2011-2013 Jose Gato Luis, Teófilo Romera  
License Creative Commons Attribution-ShareAlike 4.0 International License  
[http://creativecommons.org/licenses/by-sa/4.0/deed.en\\_US](http://creativecommons.org/licenses/by-sa/4.0/deed.en_US)



# Outline

1 Essay overview

2 The Lessons

3 Criticism

# Eric Steven Raymond (ESR)



[https://commons.wikimedia.org/wiki/File:](https://commons.wikimedia.org/wiki/File:Eric_S_Raymond_portrait.jpg)

Eric\_S\_Raymond\_portrait.jpg

- December 4, 1957
- Fetchmail, gpsd, emacs editing modes
- “The Cathedral and the Bazaar,” published in 1997 → Became a prominent voice in the open source movement
- Co-founded the Open Source Initiative in 1998

# Origins

- First version of the paper written in 1997.
- Several revisions published until 2000.
- The author unveils a “development model” through the history of the Linux kernel and one of his own tools.
- This model is presented as revolutionary, since it is useful to build large software systems with very light organization.

# The models

- **The Cathedral:** The “classic” model.
  - ▶ Closed environment.
  - ▶ Small group of leaders/developers.
  - ▶ Only “stable” releases or, in some cases, “betas”.
  - ▶ Used both in classic development models, such as waterfall, spiral, etc; and classic FLOSS projects at the time.
  - ▶ Examples: GCC, GNU Emacs.
- **The Bazaar:** The model introduced by Linus Torvalds.
  - ▶ Open environment, almost any person can participate.
  - ▶ There are no clear leaders, undefined number of developers.
  - ▶ However, there is a benevolent-dictator figure.
  - ▶ “Release early, Release often”.
  - ▶ Examples: Linux.

# The surprise

The Bazaar style of development:

- with a community resembling a large babbling bazaar of diverse agendas and approaches
- with archive repositories where anyone can propose a modification
- but emerging from this: a stable and coherent large software system

This was surprising:

- Why Linux did not fly apart in confusion...?
- ...and why Linux seemed to go from strength to strength at a speed barely imaginable to cathedral-builders?

# Outline

- 1 Essay overview
- 2 The Lessons**
- 3 Criticism

# Lesson 1

*Every good work of software starts by scratching a developer's personal itch.*

- Most successful free software projects have been started by developers with needs addressed by their “pet” project.
- In the world of proprietary software, programmers spend their time building programs that they neither need nor want.
- This motivation could explain the high quality of results given by Linux.

## Lesson 2

*Good programmers know what to write. Great ones know what to rewrite (and reuse).*

- Linus Torvalds did not try to write Linux from scratch. Instead, he started by reusing Minix code and ideas.
- Although today all reused Minix code has been removed or rewritten, while it was there, it provided scaffolding for the infant that would eventually become Linux.
- The source-sharing tradition of the Unix world has always been friendly to code reuse.

## Lesson 3

*Plan to throw one [version] away; you will, anyhow.*

*— Frederick Brooks, The Mythical Man-Month*

- We really do not understand the problem, until the first implementation is done.
- So if you want to get it right, be ready to start over at least once.

## Lesson 4

*If you have the right attitude, interesting problems will find you.*

- Eric's problem was that he needed a POP protocol client to work with.
- And he found an abandoned one.
- He proposed patches to fix shortcomings (= the right attitude).
- By interacting with the author, he found out he was no longer interested in maintaining it, and positive about passing it over to someone else (= interesting problem).
- The problem was the continuation of the abandoned client, and Eric took it over and started to coordinate it.

## Lesson 5

*When you lose interest in a program, your last duty to it is to hand it off to a competent successor.*

- Before abandoning the development of a free software, you should find another person to continue its development.
- Fortunately, in the bazaar world, some other hacker will find your abandoned work soon (assuming it's an interesting enough project, that is), and will start to develop it for his own needs.

## Lesson 6

*Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.*

- In Linux, *some* users are also hackers.
  - Thanks to source code availability, these users can be *effective* hackers.
  - This can be useful for shortening debugging time.
  - These users will diagnose problems, suggest fixes, and help in improvements.
- ⇒ The importance of having users.

## Lesson 7

*Release early. Release often. And listen to your customers.*

- This is another Linux characteristic: during very active development periods, lots of versions were released.
- Sometimes, more than one in a day.
- This maintains the hackers constantly stimulated and rewarded:
  - ▶ stimulated by the prospect of having an ego-satisfying piece of the action,
  - ▶ and rewarded by the sight of constant (even daily) improvement of their work.

## Lesson 8

*Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix becomes obvious to someone.*

### Definition (Linus's law, by ESR)

*Given enough eyeballs, all bugs are shallow.*

- Somebody finds the problem, and somebody else understands (and fixes) it.
- Cathedral vs bazaar intuition:
  - ▶ In the cathedral-builder view of programming, bugs and development problems are tricky, insidious, **deep** phenomena.
  - ▶ In the bazaar, bugs are generally shallow phenomena — they turn **shallow** when exposed to a thousand eager co-developers collaborating in next release.

*Smart data structures and dumb code works a lot better than the other way around.*

- It is difficult to understand the code written by others,
- but when we understand the data structures, understanding the code is easier.

## Lesson 10: Eric tries the model

Eric S. Raymond tries successfully the model, with the following principles:

- Releasing early and often.
- Adding everyone who contacted him about the implemented program to the beta list.
- Announcing new releases to the beta list, stimulating people to participate.
- Listening to beta-testers, polling them about design decisions and taking in consideration patches and other feedback from them.

The consequence:

*If you treat your beta-testers as your most valuable resource, they may become that, eventually.*

## Lessons 11 and 12

*The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.*

*Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.*

- Eric learned this when saw good ideas for his application suggested by users.
- The importance of giving credit where credit is due.
- These ideas helped him to understand that, initially, he was looking for the solution to the wrong problem.

## Lesson 13

*Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.*

— *Antoine de Saint-Exupéry.*

- When the code is getting both better and simpler, that is when we know it is right.
- Reminiscent of the UNIX philosophy, but on code internals rather than (geek) user interface.
- At this moment, the software maintained by Eric was, not only very different, but also simpler and better. It was time to change its name and give it its new identity: “fetchmail” instead of “popclient”.

## Lessons 14 and 15

*Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.*

*When writing gateway software of any kind, take pains to disturb the data stream as little as possible — and never throw away information unless the recipient forces you to!*

## Lessons 16 and 17

*When your language is nowhere near Turing-complete, syntactic sugar can be your friend.*

*A security system is only as secure as its secret. Beware of pseudo-secrets.*

- These are lessons learned directly from the concrete application: *fetchmail*. Not related with the management part of software engineering.

## Lessons 18 and 19

*To solve an interesting problem, start by finding a problem that is interesting to you.*

- partial re-statement of the “scratch own itch” lesson

*Provided the development coordinator has a communications medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.*

# Outline

1 Essay overview

2 The Lessons

3 Criticism

# Criticism

- No scientific rigor (it is an essay).
- Based on anecdotal evidences (not a systematic study).
- Raymond tries to generalize specific cases (Linux, fetchmail) to all free software projects.
- Some critics say that Linux is, in fact, an example of cathedral process: there is a leader, and a hierarchic structure of people with delegated tasks. Also, responsibilities are distributed in that structure although not explicitly.

## Criticism (cont.)

- Lessons to be put in perspective after 2 decades
- Free Software = Bazaar?
  - ▶ Is there *one* development model for free software?
  - ▶ Are all projects really built from many contributions of *many* developers?
- Bazaar definition may not be accurate.
- Free Software is not actually a bazaar.
  - ▶ It is not even a development model.
  - ▶ (It isn't a *business* model either.)
  - ▶ Free Software is software that respects the 4 freedoms of its users.

# A Second Look at the Cathedral and the Bazaar

(by Nikolai Bezroukov)

- *Brook Law*<sup>1</sup> still apply to Internet-based development
  - ▶ Internet only increases the quality of the pool of developers
- *Given enough eyeballs, all bugs are shallow*
  - ▶ Why waste the time of skilled developer with testing?
  - ▶ Not all bugs are created equals
  - ▶ How to “force” developers to fix a *specific* bug?
  - ▶ Do not fix ugly code, throw it away!

---

*In The Mythical Man-Month, Fred Brooks observed that programmer time is not fungible; adding developers to a late software project makes it later. He argued that the complexity and communication costs of a project rise with the square of the number of developers, while work done only rises linearly. This claim has since become known as "Brooks's Law" and is widely regarded as a truism. But if Brooks's Law were the whole picture, Linux would be impossible.*

1.

## A Second Look at the Cathedral and the Bazaar (cont.)

(by Nikolai Bezroukov)

- *Does Linux belongs to the Cathedral model or to the Bazaar model?*
  - ▶ A not so democratic Bazaar
  - ▶ Kernel core Cathedral?
- Does “FOSS development model” automatically yield the best results?

# Eric Raymond Influence

- Netscape Communications

- ▶ 22 January, 1998
- ▶ Netscape publicly released the source code of Netscape Communicator 4.0
- ▶ *On behalf of everyone at Netscape, I want to thank you for helping us get to this point in the first place. Your thinking and writings were fundamental inspirations to our decision.*

— Eric Hahn, executive vice president and chief technology officer at Netscape

- Wikipedia

- ▶ Another great example of large-scale collaboration
  - ▶ *[The Cathedral and the Bazaar] opened my eyes to the possibility of mass collaboration*
- Jimmy Wales, Wikipedia co-founder

# References

- Eric S. Raymond, *The Cathedral and the Bazaar* (1997-2000)  
<http://www.catb.org/~esr/writings/cathedral-bazaar/>
- Nikolai Bezroukov, *A Second Look to the Cathedral and the Bazaar* (1999)  
<http://firstmonday.org/article/view/708/618>
- Karl Fogel, *Producing Open Source Software: How to Run a Successful Free Software Project* (2005)  
<http://producingoss.com/>