

What's bothering me the most is some of the people who have gotten enthusiastic about [The Cathedral and the Bazaar], but, in their enthusiasm, are arguing something like a bad parody of it.
-- Eric S. Raymond, in comment "Fame? Ego? Oversimplification!"¹

There are writings so well-known and entrenched in their particular niches that familiarity with their core message is a prerequisite for inclusion. There are also writings so often discussed that many do not even bother reading them – after all, they feel like they’ve “read it already”, and can pick up the major points in conversation. Eric S. Raymond’s *The Cathedral and the Bazaar (CatB)*, a collection of essays (including the essay that gives the book its name) published in 1999 by O’Reilly and also available online, falls into both of these categories. This should immediately seem a bit problematic – taken to its extreme, inclusion in both would imply a group of people simultaneously assuming knowledge and deriving their own understanding of a work from this assumed knowledge. Thus, while *CatB* stands as the “voice of the hacker movement” (Linus Torvalds went as far as to claim, “This is how we did it”), its core message is often assumed and, I would argue, not understood completely. Though Raymond’s core essay, “The Cathedral and the Bazaar”, does paint open source in a quasi-infallible light, or “magic pixie-dust”, he takes care in the other essays to add nuance to his argument, and decide exactly when open source is “the right solution”. In this paper, I will first present Raymond’s overall picture of open source development, and then investigate and analyze his recipe for which types of project benefit from an open source model. Through this, I hope to arrive at a refined version of Raymond’s recipe. As the focus of my study, I will delve into the fourth essay in *CatB*, entitled “The Magic Cauldron”.

The Magic Cauldron in the Context of *CatB*

While each of Raymond’s essays is self-contained in that it does not assume knowledge from the other essays in the collection, it is still useful to understand where “The Magic Cauldron” sits in relation to the remainder of the papers. At the center of *CatB* is the self-titled essay, in which Raymond relates his experience with leading an open source project (fetchmail) and observing the development process of Linux, one of the largest and most visible open-source projects. Raymond emerges from this essay a strong believer in open source, claiming that “no closed-source developer can match the pool of talent the Linux community

¹ <http://www.catb.org/~esr/writings/cathedral-bazaar/fame.html>

can bring to bear on a problem.”² The problems and limitations of open source are explored later; I will return to these in my critical analysis of “The Magic Cauldron”.

As a collection of essays, *CatB* is tied together by the notion that the traditional conception of the software industry— sale-value software in a box developed by co-located developers — is, firstly, not as common as popularly thought, and second, is detrimental to good work and innovation. I will briefly explore both of these claims, as given by Raymond.

A stroll through Best Buy or Fry’s will reveal shelves upon shelves of boxed software, all by a few major names — Adobe, Microsoft, Intuit, etc. Specialist magazines also contain advertisements, by many of the same names. It is easy (and common) to thus assume that the majority of the software industry consists of developing and shipping boxed software. We here fall prey to one of Tversky and Kahneman’s biases of human judgment, availability: since these boxed-software companies are the most visible and available, we assume they truly represent the industry as a whole.

Not so, claims Raymond. Though he does not delve too far into statistics beyond his own informal polls, he maintains that 95% of software is developed in-house and never sold. Initially, this claim seems a bit flimsy without any further backing statistics. However, we will assume Raymond is correct here, given that empirical statistics for such a claim would actually be almost impossible to obtain — few companies are willing to reveal such details about their in-house process.

The second of Raymond’s claims relates to the actual value of the traditional model in creating good software. His argument for why a distributed open source model is a better idea can be broken down into three stages — historical, investigative, and academic.

The first of these is explored in “A Brief History of Hackerdom”, where Raymond delves into the early days of software development. A disconnect is shown between the heady early days of ARPANet and the state of the software industry. When ARPANet emerged,

² “The Cathedral and the Bazaar” — The Social Context of Open Source Software

electronic highways brought together hackers all over the U.S. in a critical mass; instead of remaining in isolated small groups each developing their own ephemeral local cultures, they discovered (or re-invented) themselves as a networked tribe.³

As Unix grew in popularity and in-house software development flourished (particularly in the finance industry), the early hacker ethos of collaboration was buried underneath management and bureaucracy. Thus, Raymond characterizes the closed state of the software industry as antithetical to the historical hacker ethos.

Raymond's second stage of exploration is experimental; in "The Cathedral and the Bazaar", he plunges head-first into open-source development, by assuming control of a full-featured email client, fetchmail. His experience is overall extremely positive, and leads to one of the most often quoted phrases in *CatB*:

8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone. Or, less formally, "Given enough eyeballs, all bugs are shallow." I dub this: "Linus's Law".

Raymond experienced a wide community of developers and user/bug-hunters ready to assist him. One element he does not address in his experimental phase is the nature of the software; would his project have been as successful if, instead of an email client, he was developing a sequential database? This is another question we must address when deciding when open source is "right".

Raymond's final stage is theoretical/academic – he brings in an economics perspective to the nature of open source development. In short, he views the hacker culture, and open source development as a whole, as a gift-exchange culture with an ego-boost component. Thus, the status of developers is determined by the contributions (gifts) they have provided their tribe – and their reward is a boost in ego. Raymond misses an opportunity here by not speaking of crowding-out effects and their impact on motivation; while he claims 'pay-for-performance' is a poor model for developers, his arguments lack the foundation that the psychology research could have added. Further, he does not distinguish properly between 'fame' and 'ego-boost', a lack of distinction that led many to criticize his paper. His later

³ "A Brief History of Hackerdom" – The Early Hackers

writings do a better job of defining the motivating element behind open-source hackers as “peer repute.”

Moving beyond explanation, into application

So far, Raymond has provided a mostly compelling argument for why the open source, distributed model is natural for software development. This, it seems, is as far as most people go in understanding Raymond’s argument. I would argue, however, that the remainder of his work – where he attempts to characterize the right situations for open source – is central to *CatB*, and the reason why Netscape and other software companies looked to him as the spokesperson for the open source movement.

Raymond, in the section “When to be Open, When to be Closed” in “The Magic Cauldron”, presents the following list:

[Open Source is right when...]

- (a) Reliability/stability/scalability are critical.
- (b) Correctness of design and implementation cannot readily be verified by means other than independent peer review.
- (c) The software is critical to the user's control of his/her business.
- (d) The software establishes or enables a common computing and communications infrastructure.
- (e) Key methods (or functional equivalents of them) are part of common engineering knowledge.

I will visit the first four of these rules in turn. The fifth rule is interesting, but focusing on the first four will allow for more tightly focused discussion.

(A) Reliability, stability, scalability are critical

The first of Raymond’s claim harkens back to his “Linus’ Law” – if open-source is indeed the most efficient method of bug-hunting and fixing, it seems natural that mission-critical software will benefit from open source. This seems to be backed by the success of several large-scale projects – particularly Linux and Apache. I would argue, however, that there remains a stumbling block for open source to establish itself as the true dominant force in the domain of rock-solid software: the service industry around it needs strengthening. It is still the case that system administrators will deploy closed-source solutions because they are backed by companies known for their support. When a system goes down, and the manager is yelling for an explanation, a simple “I’ll just call IBM and they’ll help up solve

this” sounds far more reassuring than, “I’ll just check the Gentoo message-boards.” Red Hat has come a long way in establishing a solid service reputation, but their domain remains mostly the Linux OS. If a few more service companies established themselves as reputable sources of 24/7, on-demand support, Raymond’s claim would be reinforced: “Reliability, stability, scalability are critical, and open-source support service companies are in place to reassure deployers.”

(B) Correctness of design and implementation cannot readily be verified by means other than independent peer review.

In “The Cathedral and the Bazaar”, Raymond had previously explored the notion of peer review, showing that independent peer review is a powerful source for checking correctness, and the open source development provides a wealth of such reviewers. I believe Raymond overstates the importance of peer review, however – it seems more a complement to traditional internal software testing, not a replacement for it or even necessarily a more thorough solution. A large-scale software company could feasibly employ a division devoted to unit testing, regression checking, and frequent evaluation of software iterations, with much the same effect.

Of more interest are situations in which independent peer review is currently impossible, and open-ness is not only desirable but the responsible choice. Voting machines, for example, currently have no mechanism for peer review, and the internal software testing solution presented above is unacceptable. Infrastructure components (which we’ll return to in claim (d)) also currently have no peer review mechanism, but one would be socially desirable, to ensure fairness throughout the system and prevent censorship. Raymond was on the right track on thinking about peer review, but did not see the situations in which peer review can add true value. Instead, claim (b) might state, “Correctness of design and implementation must, for reasons of public concern or other societal interests, be verifiable by independent peer review.”

(C) The software is critical to the user’s control of his/her business.

This claim seems mostly a re-statement of (A), with the important addition of the notion of “control”. In a later chapter Raymond claims, “when your key business processes are executed by opaque blocks of bits that you can’t even see inside (let alone modify) you have

lost control of your business."⁴ This is one of the most impassionate claims that Raymond makes, and it is an effective one – business owners (particularly in small- to mid-size companies) should be concerned as to the electronic foundations of their business. But merely having “open source software” underneath does not seem to help much – how many of these business owners can trod through thousands of lines of C++ in order to fix a patch? These owners might be able to hire a consultant to code a patch or an extension to provide the necessary functionality, but even that consultant is likely to be overwhelmed by unfamiliar code. Instead, I would argue that open-ness is not enough, and *documentation* is the key here. By ‘documentation’ I refer to both the developer documentation that allows future developers to understand the current code, and also end-user documentation. Amended, rule C would state “open source is right when the software is critical to a user’s control of his/her business, and documentation is available so that user or a future developer can easily extend the software.”

Unfortunately, documentation has been a weak point of open source software – I believe the reasons for this are threefold. Firstly, most coders’ specialty is not writing, and most writers’ specialty is not coding. Finding documentation writers that neatly bridge that gap is difficult. Secondly, interfaces and functionality are often in flux, and new point releases may change details that render current documentation obsolete – the quick release cycle of open source turns out to be a disadvantage here. Finally, the “ego-boost” described by Raymond does not quite apply to documentation writing; it is often a thankless job.

How can the open source community change the current landscape of documentation writing? The first step is to provide a set of guidelines for what constitutes good documentation, on both the developer/code side and the user-interface side. Secondly, more projects should follow the Mozilla Foundation’s lead in establishing common APIs that are frozen for the entirety of a version cycle (for example, all *1.x* releases). Finally, documentation writers should be recognized for the crucial role they play in development.

⁴ “The Magic Cauldron” – Open Source and Strategic Business Risk

(D) - The software establishes or enables a common computing and communications infrastructure

Since Raymond's paper was written in 1999, he did not have the benefit of Lawrence Lessig's recent writings to draw on; we have this privilege, and can thus bridge both writers. Raymond's "Magic Cauldron" is relatively apolitical, but rule (D) has become increasingly relevant, as Yochai Benkler and Lessig have written at length⁵ about the importance of maintaining freedom and open-ness through all layers of communication. In other words, it is not enough that we have free speech at the topmost, 'content' layer – this freedom will be threatened if the 'physical' and 'logical' layers are not free as well.

Software plays a serious role in the 'logical' layer of the communications and computing infrastructure, defining protocols and mediating messages between users of this infrastructure. Open source software and open protocols become vital as components of this logical layer, ensuring that this layer remains free. The Internet has, mostly, been built on open architecture: in its domain name resolution (DNS), its principle protocols (TCP/IP), its most popular web-server (Apache), etc are all either open protocols or open source software.

Raymond's claim was spot-on, even before the current push for greater regulation and control of the Internet. Thus, I would not amend it, but only add that two things must be accomplished by the open source community to ensure rule (D) gets applied: they must ensure that their solutions remain the most secure, efficient, easy to deploy, even in the face of proprietary solutions. Secondly, they must actively advocate, in parallel with developing their superior product, the fact that maintaining freedom in communications infrastructure is in fact an issue of democracy.

An Amended Version of Raymond's Recipe; Concluding Remarks

In sum, we have arrived at the following list:

[Open Source is right when...]

- (a) Reliability, stability, scalability are critical, and open-source support service companies are in place to reassure deployers
- (b) Correctness of design and implementation must, for reasons of public concern or other societal interests, be verifiable by independent peer review.

⁵ See Lessig, L. (2002). The architecture of innovation. Duke Law Journal. 51(6):1783-1801

- (c) The software is critical to a user's control of his/her business, and documentation is available so that user or a future developer can easily extend the software
- (d) The software establishes or enables a common computing and communications infrastructure.
- (e) Key methods (or functional equivalents of them) are part of common engineering knowledge.

This task of expanding and updating Raymond's 'recipe' is particularly relevant given open source's current transition from an unknown, relatively minor player in the software ecosphere to a major force, with the ability to enact social and economic change. My amendments to Raymond's original rules sometimes add a political charge absent from the original – particularly rules (b) and (d) – but I feel this is an almost inevitable trajectory. While his arguments focus only on when open source is right for a business or product, the last five years have shown that software is speech, and software is increasingly political. With these amendments, the recipe gains the added element of when open source is right for *society*. Though Raymond is probably reluctant to add this element to his rhetoric – and risk alienating those who see software as a mere tool – it is likely to be the next main role to be played by open source.