4 Standing in Front of the Open Source Steamroller

Robert L. Glass

I am a contrarian by nature. I have a certificate pronouncing me the "premier curmudgeon of software practice." I am the author of a regular column in *IEEE Software* magazine called "The Loyal Opposition." I have been standing up in front of advancing software steamrollers throughout my career:

- Attempting to diminish the communication chasm between academe in industry, beginning as far back as the early 1960s (and, I would add, continuing to this day)
- Opposing the apparent inevitability of IBM hardware and software in the 1960s (I participated on a team that put a complete homegrown software system—including, for example, a homegrown operating system—on its IBM hardware in order to allow us to more easily transition to other vendors later on)
- Questioning the insistent academic push for formal methods in software, beginning in the 1970s (I questioned them then, and I fault the academic community for advocating without evaluating them now)
- Looking objectively at each new software fad and fancy, from the structured approaches to object orientation to the agile methods, to find out whether there is any research support for the hyped claims of conceptual zealots (there almost never is)

All of that marks me, of course, as someone you might want to listen to, but not necessarily believe in!

Here in this book on open source and free software, I want to take that same contrarian position. To date, much of the writing on open source software has been overwhelmingly supportive of the idea. That overwhelming support is the steamroller that I intend to stand up in front of here.

Before I get to the primary content of this chapter, disaggregating and questioning the primary tenets of open source, let me present to you here some of the qualifications that have resulted in my feeling confident enough to take such an outrageous position:

- I am a 45+-year veteran of the software profession.
- I have deliberately remained a software technologist throughout my career, carefully avoiding any management responsibilities.
- As a technologist, I have made the most of what I call the "Power of Peonage"—the notion that a skilled and successful technologist has political powers that no one further up the management hierarchy has, because they are not vulnerable to loss of position if someone chooses to punish them for their actions; one of my favorite sayings is "you can't fall off the floor" (I once wrote a book of anecdotes, now out of print, about that "Power of Peonage")
- Two of my proudest moments in the software field involved building software products for which I received no compensation by my employer:
  - A Fortran-to-Neliac programming language translator, back in those anti-IBM days I mentioned previously, which I built in my spare time at home just to see if I could do it (I called my translator "Jolly Roger," because its goal was to make it possible to transition IBM-dependent Fortran programs to the vendor-independent Neliac language my team had chosen to support).
  - A generalization of a chemical composition scientific program whose goal was to calculate/simulate the thrust that mixing/igniting certain chemicals in a rocket engine would produce. When I began my Thanksgiving Day work at home, the product would handle only combinations of five chemical elements. When I finished, the program could handle any additional sixth element for which the users could supply the needed data. (This addition allowed my users to determine that boron, being touted in the newsmagazines of that time as the chemical element of rocketry's future, was worthless as an additive.)

So, am I in favor of programmers doing their own programming thing on their own time? Of course. It is the claims and political activities that go along with all of that that form the steamroller I want to stand in front of. Let me tell you some of what I dislike about the open source movement.

Hype

Hype is not unknown in the software field in general. The advocates of every new software idea exaggerate the benefits of using that idea. Those

exaggerated claims generally have no basis in reality (see, for example, Glass 1999). Unfortunately, and perhaps surprisingly, the advocates of open source are no better in this regard than their callous proprietary colleagues. Claims are made for the use and future of open source software that simply boggle the rational mind, as described in the following sections.

Best People

The claim is frequently made that open source programmers are the best programmers around. One author, apparently acting on input from open source zealots, said things like "Linux is the darling of talented programmers," and opined that the open source movement was "a fast-forward environment in which programming's best and brightest . . . contribute the most innovative solutions" (Sanders 1998).

Is there any evidence to support these claims? My answer is "No," for several reasons:

- There is little data on who the best programmers are. Attempts to define Programmer Aptitude Tests, for example, which evaluate the capabilities of subjects to become good programmers, have historically been largely failures. In an early study, the correlation between computer science grades and practitioner achievement was found to be negative. Although some programmers are hugely better than others—factors up to 30:1 have been cited—nothing in the field's research suggests that we have found an objective way of determining who those best people are.
- Since we can't identify who the best people are, there is no way to study the likelihood of their being open source programmers. Thus those who claim that open source people are software's "best and brightest" cannot possibly back up those claims with any factual evidence.
- It is an interesting characteristic of programmers that most of them tend to believe that they are the best in the field. Certainly, I know that few programmers are better than I am! It used to be a standard joke in the software field that, if a roomful of programmers were asked to rate themselves, none of them would end up in the second tier. Therefore, I suspect that if you took any group of programmers—including open source programmers—and asked them if they were the field's best and brightest, they would answer in the affirmative. That, of course, does not make it so.

Most Reliable

The claim is also frequently made that open source software is the most reliable software available. In this case, there are some studies—and some interesting data—to shed light on this claim.

The first thing that should be said about open source reliability is that its advocates claim that a study identified as the "Fuzz Papers" (Miller, Fredriksen, and So 1990; Miller et al. 1995; Forrester and Miller 2000) produced results that showed that their software was, indeed, more reliable than its proprietary alternatives.

As a student of software claims, I have followed up on the Fuzz Papers. I obtained the papers, read and analyzed them, and contacted their primary author to investigate the matter even further. The bottom line of all that effort is that the Fuzz Papers have virtually nothing to say about open source software, one way or the other, and their author agrees with that assessment (he does say, though, that he personally believes that open source may well be more reliable). Thus it is truly bizarre that anyone would claim that these studies (they are peculiar studies of software reliability, and to understand why I say "peculiar," you should read them yourself!) support the notion that open source code is more reliable.

Since then at least one academic researcher has done further, real studies of open source code reliability. In that work, we find that open source programmers do not tend to use any special reliability techniques—for example, fully 80% of them do not produce any test plans, and only 40% use any test tools. The author surmises that "open source people tend to rely on other people to look for defects in their products" (Zhao and Elbaum 2000).

That latter point deserves more discussion. One popular saying in the open source world is "Given enough eyeballs, all bugs are shallow" (Raymond 2001, 41). What this means is that the open source culture, which involves people reading and critiquing the code of others, will tend to find and eventually eliminate all software bugs. But there is a problem with this belief. It is based on the assumption that all open source code will be thoroughly reviewed by its readers. However, the review of open source code is likely to be very spotty. We will see later in this chapter that open source people tend to review heavily code that particularly interests them, and spend little or no time on code that does not. Therefore, there is no guarantee that any piece of open source code will be thoroughly reviewed by members of the community. To make matters worse, there is no data regarding how much of any open source code is in fact reviewed. As a result, the belief that all open source code will be subjected to "many eyeballs" is naive and, in the end, unprovable.

Most Secure

Just as the claims are rampant that open source software is more reliable than its proprietary alternatives, there are analogous claims that it is more

secure. The more the drumbeat of concern for secure software accelerates, the louder those claims become.

Unlike the software reliability issue, there is very little evidence on either side of the ledger regarding software security. Certainly security holes have been found in proprietary software. Certainly also, holes have been found in open source code (see, for example, Glass 2002a). And both sides have made strong claims that their software is either secure, or that they are making it so.

Probably the most accurate thing anyone can say about software security is that (a) it is all too easy for programmers to leave holes, independent of how the code is being written (for a list of the top five security-related software defects, see Glass 2003a); (b) the perversity of "crackers" tends to mean that wherever they seek security holes, they are likely to find them, and they tend to seek wherever the loudest claims are that the software is secure! (For example, in the book *Know Your Enemy* (Honeypot Project 2002), a study of cracker techniques by using "honeypot" systems to trap them, one "black hat" was specifically going after Linux-based .edu systems because of their claims of invulnerability, a chilling thought for both open source advocates and academics who use their wares).

And with respect to the open source claims, there is plenty of anecdotal evidence (e.g., Glass 2003b) to back both the security claims of the open source advocates and their proprietary counterparts, but there is really no definitive evidence to cause either side to be seen as victorious.

Economic Model

Probably the most fascinating thing about the open source movement is its economic model. Here are people often willing to work for no recompense whatsoever, except the joy of a job well done and the accolades of their peers! It all seems terribly noble, and in fact that nobility is undoubtedly part of the appeal of the open source movement.

It also seems faintly Utopian. There have been lots of Utopian movements in the past, where workers banded together to work for that joy of a job well done and for the common good (and, once again, for the accolades of their peers). There are two interesting things about Utopian movements. They begin in enormous enthusiasm. And they end, usually a few decades later, in failure. What are the most common causes of Utopian failure?

▪ The impractical nature of the economic model (I will discuss that in following sections).
▪ Political splintering, as discussed in a following section (note that some Utopian religious societies, such as the Harmonists, also failed because they decided to practice celibacy, but that fact seems totally unrelated to the open source movement!).

So, regarding that practicality issue, just how impractical is the open source movement? To date, the answer would appear to be that there is no sign of the movement's collapse because it is impractical.

There is little evidence one way or the other as to the size of the movement, but the frequency of its mention in the computing popular press would tend to suggest that it is growing. Its advocates are also increasingly outspoken. And companies have sprung up that, while not making money on open source products, are making money on servicing those products (e.g., Red Hat).

Then there is the issue of Communism, an issue that is usually present in discussions about the problems of open source, although it has rarely surfaced. There is a faint whiff of Communism about the concept of working for no financial gain. Open source is certainly not about "from each according to his ability, to each according to his need," so that whiff is indeed faint. But the sense of nobility that open source proponents feel, in working for no financial gain, resonates with some of the other basic Communist philosophies. And the open source proponents themselves can sometimes sound just like Communists. One columnist (Taschek 2002) recently spoke of anti-Linux forces as "capitalist interests," and later on as "capitalist forces." He also claimed that some anti-Linux people are behaving as "Microsoft lackeys." While opposing capitalism and using the word "lackeys" is not proof that the author of that column is Communist, the rhetoric he chose to use certainly reminds one of Communist rhetoric. Whether Communism is a good thing or not is, of course, up to the reader. But in this discussion of the practicality of the open source economic model, it is worth noting that the Communist system is in considerable decline and disfavor in today's world.

It is particularly interesting that advocates of open source refer to "the cathedral and the bazaar" in their discussions of the movement and its alternatives (Raymond 2001). In that view, open source represents the bazaar, a place where people freely trade their wares and skills, and the proprietary movement is represented by the cathedral, a bricks-and-mortar institution with little flexibility for change. I find that particularly inter-

esting, because, when I first saw this particular analogy, I assumed that open source would be the cathedral, a pristine and worshipful place, and proprietary software would be the bazaar, where products and money change hands, and there is a strong tendency toward working for profit! I suppose that those who invent analogies are entitled to make them work in any way they wish. But my own thinking about this pair of analogies is that the open source way of viewing it is fairly bizarre!

Does the open source economic model show promise of working in the long term? There is no evidence at present that it will not, but on the other hand it is worth noting that the analogies we can draw between it and other relevant economic models is primarily about models that eventually failed.

Political/Sociological Model

It is common in open source circles to see the participants in the open source movement as willing, independent enthusiasts. But, on deeper analysis, it is evident that there is a strange kind of hierarchy in the movement.

First of all, there are the methodology gurus. A very few outspoken participants articulate the movement and its advantages, through their writings and speakings, spreading the gospel of open source.

More significantly, there are also product gurus. Because of the large number of people reading, critiquing, and offering changes to open source products, there is a need for someone to adjudicate among those proposed changes and configure the final agreed-upon version of the product. Linux, for example, has its Linus Torvalds, and it would be difficult to imagine the success of the Linux operating system without Linus. I will discuss a bit later why a product guru is needed.

Then there are the contributors of open source products. These are the programmers who develop products and release them into the open source product inventory. Some of these contributors, if their product is especially successful, may become product gurus.

And finally, there is that great mass of open source code readers. Readers analyze and critique code, find its faults, and propose changes and enhancements. As in many hierarchies, the success of the open source movement is really dependent on these readers at the bottom of the hierarchy. Its claims of reliability and security, for example, are in the end entirely dependent on the rigor and energy and skill which the readers place in their work.

To understand this hierarchy better, it is necessary to contemplate how it functions when a change or an error fix for a product is proposed. The change moves up the hierarchy to the product guru, who then makes a decision as to whether the change is worthy of inclusion in the product. If it is, the change is made and becomes a permanent part of the product.

It is when the change is rejected for inclusion that things can get interesting. Now the reader who identified the change has a dilemma to deal with. Either he/she must forget about the change, or make that change in a special and different version of the product. This latter is actually considered an option in the open source movement, and there is a verb—*forking*—that covers this possibility. The reader who wants his change made in a special version is said to have *forked* the product, and the further development of the product may take place on both of these forks.

But forking is an extremely uncommon thing to do in the open source movement, with good reason. First of all, there is the possibility of loss of commonality. The Unix operating system, for example, is roundly criticized because there are multiple versions of that product, each with its own advocate (often a vendor), and as a result there is really no such thing as *the* Unix system any more. That is a serious enough problem that it warrants strict attention to whether forking a product is really a good thing.

There is an even stronger reason, though, why forking is uncommon. It has been well known in the software field for more than 30 years that making modifications to a standard product is a bad idea. It is a bad idea because, as the product inevitably progresses through many versions—each of which usually includes desirable changes and modifications—the forked version(s) are left out of that progress. Or, worse yet, the new changes are constantly being added to the forked version (or the forking changes are being added to the new standard version) by the person who created the fork, both of which are terribly labor-intensive and error-prone activities.

Now, let's step back and look at the impact of this forking problem on the field of open source. The claim is frequently made by open source advocates that programmers who find fault with a product are free to make their own fixes to it, and are capable of doing so because they have access to the product's source code. That's all well and good if those changes are eventually accepted into the product, but if they are not, then the very serious problem of forking arises. Thus the notion of the average user feeling free to change the open source product is a highly mixed blessing, and one unlikely to be frequently exercised.

There is another interesting sociological problem with open source. Again, the claim is frequently made that users can read their code and find and fix its problems. This is all well and good if the users are programmers, but if they are not this is simply a technical impossibility. Code reading is a difficult exercise even for the trained programmer, and it is simply not possible, to any meaningful degree, for the non-programmer user. What this means is that only code where programmers are the users is likely to receive the benefits of open source code reading, such as the "many eyeballs" reliability advantages mentioned previously. And how much code is used by programmers? This is an important question, one for which fortunately there are answers. The largest category of code, according to numerous censuses of programs, is for business applications—payroll, general ledger, and so on. The next largest category is for scientific/engineering applications. In neither of these cases are the users, in general, programmers. It is only for the category "systems programs" (e.g., operating systems, compilers, programmer support tools) where, commonly, the users are in fact programmers. Thus the percentage of software that is likely to be subject to open source reading is at best pretty minuscule.

All of these open source hierarchic sociological conditions are a bit ironic, in the context of some of the claims made for open source. For example, one zealot, in the midst of reciting the benefits of open source and urging people to switch to its use, said "Abandon your corporate hierarchies," implying that hierarchies didn't exist in the open source movement. As we have seen, that claim is extremely specious.

Remember that we spoke earlier about Utopian societies eventually dying because of political splintering? So far, the open source movement has nicely avoided many of those kinds of hazards, especially since forking (which might be the source of such splintering) turns out to be such a bad idea (for nonpolitical reasons). There is one serious fracture in the open source movement, between those who believe in "free" software and those who believe in "open source" software (an explanation of the differences is vitally important to those on both sides of the fracture, but of little importance to anyone else studying the movement from a software engineering perspective), but so far, except for public spats in the literature, these differences seem to have had little effect on the field.

However, there are some interesting political considerations afoot here. Both the open source and proprietary supporters have begun trying to enlist political support to ensure and enhance the future of their approaches (Glass 2002b). For example, one South American country (Peru) is considering a law that would require "free software in public

administration," on the grounds that free software is the only way to guarantee local control of the software product (as we have seen earlier, that is a dangerously naive argument). And the U.S. Department of Defense is said to have been inundated by requests from proprietary companies like Microsoft to oppose the use of open source code in DoD systems. This kind of thing, ugly at best, may well get uglier as time goes on.

All of that brings us to another fascinating question. What is the future of open source . . . and how is it related to the past of the software field?

The Future . . . and the Past

First, let's look back at the past of open source software. Raymond (2001) dates open source back to "the beginnings of the Internet, 30 years ago."

That doesn't begin to cover open source's beginnings. It is important to realize that free and open software dates back to the origins of the computing field, as far back as the 1950s, fifty-odd years ago. Back then, all software was available for free, and most of it was open.

Software was available for free because it hadn't really occurred to anyone that it had value. The feeling back then was that computer hardware and software were inextricably intertwined, and so you bought the hardware and got the software thrown in for free. And software was open because there was little reason for closing it—since it had no value in the marketplace, it didn't occur to most people in the field that viewing source code should be restricted. There were, in fact, thriving software bazaars, where software was available for the taking from user organizations like SHARE, and the highest accolade any programmer could receive was to have his or her software accepted for distribution in the SHARE library, from which it was available to anyone in the field.

Freely available, open software remained the rule into the mid-1960s, when antitrust action against IBM by the U.S. Department of Justice first raised the issue of whether the so-called "bundling" of software with hardware constituted a restriction of trade. Eventually, IBM "unbundled" hardware and software, and—for the first time—it was possible to sell software in the open marketplace. However, IBM (it was widely believed at the time) deliberately underpriced its software to inhibit a marketplace for software, which might enable computer users to move beyond IBM products into those of other hardware vendors (who had historically not offered as much bundled software as IBM). Thus, even when software was no longer free and open, there was still not much of a marketplace for it.

It was a matter of another decade or so before the marketplace for software became significant. Until that happened, there was a plethora of small software companies not making much money, and nothing like today's Microsoft and Computer Associates.

Whether all of that was a good thing or a bad thing can, of course, be considered an open question. But for those of us who lived through the era of software that was free and open because there were no alternatives, a return to the notion of free and open software (and the loss of the capability to profit from products in which we feel pride) feels like a huge regressive step. However, there aren't many of us old-timers around anymore, so I suppose this argument is of little interest to the issue of the future of open source as seen from the twenty-first century.

Because of all of that, let's return to a discussion of the future of open source. Advocates tend to make it sound like open source is without question the future of the software field, saying things like "the open-source age is inevitable," while chiding Microsoft (the primary putative enemy of open source, as we have already seen above) for sticking to the "buggy whip" proprietary approach (Pavlicek 2002).

Raymond (2001) goes even further. He says things like "Windows 2000 will be either canceled or dead on arrival," "the proprietary Unix sector will almost completely collapse," "Linux will be in effective control of servers, data centers, ISPs, and the Internet," and ultimately, "I expect the open source movement to have essentially won its point about software within the next three to five years." And then he proposes a blueprint for fighting the war that he sees necessary to make it so, with things like "co-opting the prestige media that serves the Fortune 500" (he names, for example, the *New York Times* and the *Wall Street Journal*), "educating hackers in guerrilla marketing techniques," and "enforcing purity with an open source certification mark." It is important to realize, of course, that we are well into his predicted three-to-five-year future, and none of those things have happened. Clearly, open source zealots are going to have to readjust their timetable for the future, if not give up on it entirely.

There are other views of software's future, of course. Some of those views I have expressed in the preceding material, which suggest that open source, far from being software's future, may be a passing fad, a Utopian-like dream. Others simply go on about their business ignoring open source, for the most part, participating in the proprietary software market place with only an occasional uneasy glance over their shoulders. Still others, looking for a safety play, are betting on both open source and proprietary software, developing products consistent with both approaches (ironically, IBM is

one of those companies). But perhaps my most favorite view of the future of the software field comes from Derek Burney, CEO (at the time of this pronouncement) of Corel, a company which had elected to host its future tools development work on the open source Linux operating system. Responding to a question about developing open source versions of Corel's WordPerfect software suite, he said "We have no plans to do so." Then he added "In my opinion, open source makes sense for operating systems and nothing more."

The future of open source? It could range anywhere from "the inevitable future of the software field" to "it's only good for operating systems, nothing more" to "it's in all probability a passing fad."

No matter how you slice it—inevitable future or passing fad—the notion of open source software has certainly livened up the software scene, circa 2005!