f i ® s t m ¤ ñ d @ ¥
PEER-REVIEWED JOURNAL ON THE INTERNET

Fundamental issues with open source software development

by Michelle Levesque

Abstract

Fundamental issues with open source software development by Michelle Levesque

Despite the growing success of the Open Source movement, most of the general public continues to feel that Open Source software is inaccessible to them. This paper discusses five fundamental problems with the current Open Source software development trend, explores why these issues are holding the movement back, and offers solutions that might help overcome these problems. The lack of focus on user interface design causes users to prefer proprietary software's more intuitive interface. Open Source software tends to lack the complete and accessible documentation that retains users. Developers focus on features in their software, rather than ensuring that they have a solid core. Open Source programmers also tend to program with themselves as an intended audience, rather than the general public. Lastly, there is a widely known stubbornness by Open Source programmers in refusing to learn from what lessons proprietary software has to offer. If Open Source software wishes to become widely used and embraced by the general public, all five of these issues will have to be overcome.

Contents

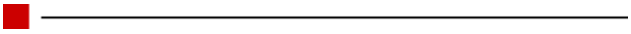
Introduction

It's my Open Source project and I'll code what I want to.

Over the past few months, I've found myself charged with the task of taking an existing Open Source project (to avoid pointing fingers, let's just call it Project X) and customizing it for academic use. Though I won't claim to be an expert in the realm of all Open Source software programming trends, I have a lot of exposure to it: I consistently try to use Open Source technology whenever possible (I fully support the sociology behind the movement), I've been a major player in a few small Open Source projects still in development, and I now have the experience of a few months of working on Project X. So despite not being an expert, I believe my opinion can stand as a relatively well–informed one.

I have five major complaints about Open Source [1] software development, but in advance I would like to clarify two things. First of all, there will always be exceptions to every rule. For example, I believe that relatively few complaints listed here apply to the Open Source browser Firefox [2] which continues to surpass my expectations. I'm discussing general trends that I've noticed, not specific cases. Secondly, I don't think that these are unresolvable problems. The purpose of this document is to raise awareness — not to mindlessly complain — in hopes that the Open Source community may begin to change their mind–set about some of these issues and work towards improving them.

That being said, I've found the five most important flaws with Open Source software development to be as follows:



User interface design

Project X comes with a neat interactive calendar. Just as you'd expect, you can schedule events, share events with others, and resolve conflicts. However no one will ever know about it, because in order to see the calendar module, you have to know the URL of the module in advance: there are no links to it, aside from one that's buried several pages deep. Project X's user interface is a nightmare. There are lots of little colourful tabs and the layout is certainly aesthetically pleasing, but it's extraordinarily difficult to figure out how to do even simple tasks. Very little of the interface is intuitive, and some tasks you can't even use the GUI to accomplish: you have to manually edit the database.

For some reason, Open Source projects seem to have a lot of trouble with user interface design. A good example of this is the Mac OS X situation. I've seen people with relatively little computer experience navigate around the OS X desktop for a few minutes,

and then turn around and tell me that it "flows very nicely" and "just feels nicer" than what they're used to. If I'd put the same person on KDE or Gnome, they probably would have spent half of their time fighting their own intuition, and the other half wondering why they were being forced to sit in front of such a clunky desktop when their Windows XP computer worked so much better. Regardless of whether or not *you* think that KDE and Gnome are poor front ends (see Religious blindness), Mac OS X has proven to be a much more usable, friendly, intuitive interface to a Unix back–end than anything that the Open Source community offers.

I suspect that there isn't one single reason for the poor quality of user interfaces, but here are some explanations I've heard roaming the Open Source circles: geeks value integrity over beauty; the gender gap in Open Source communities; it's intuitive to the programmers so why would they fix it? (see Programming for the Self), the belief that a pretty user interface can always be designed later once they're done the *real* work, the belief that user interface design isn't real work, and several others.

What the user will see - and what they' ll judge the project based on - is the user interface. If it's inadequate, no one outside of other geeks will touch the program.

Many of these sound like very plausible causes, and I suspect that the true reason for user interface neglect is a combination of many of them. However, if the Open Source community wishes to truly prosper and have their tools used by the general public, it is fundamentally necessary for them to recognize that the majority of the users will never know that they happened to invent a particularly clever algorithm for synchronizing the multi–threaded editing of their complex data structure. What the user *will* see — and what they'll judge the project based on — is the user interface. If it's inadequate, no one outside of other geeks will touch the program.

Documentation

Project X has fairly lengthy documentation, which isn't always the case in Open Source projects. Unfortunately, the documentation is for a deprecated version of Project X and they just slapped the newest revision number on the title. Also, it assumes that your system is configured exactly the same way that the documentation writer's system was configured. You also need to be very experienced in system administration, and know how to manipulate the programming language it's written in, because errors are going to occur that aren't mentioned in the documentation, and you're going to have to know how to debug them.

Open Source projects tend to have a major problem with providing decent documentation — if they provide any documentation at all. Because they don't have a contractual responsibility to provide this documentation, it's usually intended to be a general guide rather than a complete manual that you could hand to a novice. Imagine what the following sentence looks like to someone who knows little about Unix and is installing it for the first time: "You will need a list of MD5 checksums for the binary files. If you have the `md5sum` program, you can ensure that your files are not corrupt by running `md5sum -v -c md5sum.txt`." [3] The most common response to this complaint is "if they can't understand it, they're not ready to install it," but then how are they expected to learn? Documentation should always cater to the lowest common denominator.

Without adequate documentation, Open Source projects are inherently at a disadvantage.

Also, anyone who has ever had to debug a problem in Open Source software knows that the answers don't lie in Open Source software documentation: they're found in Usenet articles, bulletin boards and chat logs. Users who can't figure out how to do something runs to `alt.projectx.devel` and asks the same question as hundreds of users before them. Some expert takes pity on their plight and responds to their question, but never documents this answer. So when the next user hits the same problem, the process has to be repeated. Additionally, this is making the fundamentally flawed assumption that users are capable of finding these alternative streams of communication [4], or that they're patient enough and care about the product enough, to bother. Without adequate documentation, Open Source projects are inherently at a disadvantage.

Feature–centric development

Upon installing Project X, you have a powerful set of tools at your fingertips for accomplishing a wide variety of tasks, with dozens of impressive features associated with each of these tools. Unfortunately, upon installing Project X, your Unix groups break. During my numerous correspondences with the developers of the main fork of Project X, I was shown where to download patches for many of the problems in the Project X release, and reminded that although it still had some problems, Project X certainly had lots of cool features, didn't it?

At the very first lecture of the Software Tools and Systems Programming class that I took, we were carefully instructed that the best software tools are small programs that do one thing well and interface cleanly with the other tools. This sounds like a philosophy which is perfectly suited for the Open Source movement: if you have many contributers and they all create one (or several) small programs that do one thing well and interface cleanly with the other programs, a very clean and powerful system can come out of it.

And I believe that this has been proven by the durability and longevity of the Unix operating system.

But somewhere along the line, I think that Open Source programmers forgot about these fundamental concepts. Feature creep began to set in (and for once it wasn't because there were clients that kept demanding more). It's the individual programmer who wants to add the feature to a project that enables the current 2D graph to be displayed in 3D (accessible by cellphone, pager, and instant messenger), and then make it work with any of keyboard strokes, button clicking, mouse gestures, voice command, and laser pointer awareness. There's nothing exciting about being the programmer who cleans up some of the libraries by converting common code blocks into singular functions.

With so much emphasis on features and geek cred, fundamental aspects of a programming project go missing. This isn't always core functionality like it is in Project X. Sometimes it's a lack of focus on the user interface (see User interface design), documentation, coding standards, security, project direction, specified target audience (see programming for the self), etc. It's highly unlikely that *all* of these are wrong within a single project, but I suspect that at least *some* of them are missing in most Open Source projects. And I strongly believe that this is because of the personal advantages of ignoring the mundane work and spending more time building the "fun stuff" [5].

Programming for the self

When I was first installing Project X, I hit a roadblock that I couldn't get around, so I took a leisurely stroll to the Project X IRC channel. After explaining my problem to the Project X developers, they helpfully provided me with several commands and file changes that immediately solved my problem. However, I would have never been able to discover these fixes on my own, until weeks later when I became *much* more familiar with the Project X code base. When I asked them how they expected new users to figure this stuff out, they shrugged and replied that it had always seemed fairly intuitive to them.

A very common problem among software developers (and not just ones working on Open Source projects) is the fallacy that intuitiveness is problem–specific rather than audience–specific: what is easy to them will naturally be easy to everyone else [6]. Therefore, they don't bother simplying anything that they regard as intuitive. This problem is particularly potent in Open Source projects due to members of the community investing time only in building what features they themselves would want to use (see feature–centric development), and the fact that audiences are often not as well defined as they are in commercial software (because you don't have to literally sell to a specific group). Also, Open Source projects don't have the usability experts that commercial software projects may employ.

The result is that Open Source projects are made by programmers for programmers, who then can’t understand why the general public would bother with proprietary software when this Open Source tool is working so well for *them*. Meanwhile, the rest of the world begins to associate "Open Source" with software that’s only accessible to the technocratic elite.

Before the start of every Open Source programming project, a conscious decision should be reached about whether this project’s target audience is other programmers or the general public. If it’s the latter, there should be a regular effort to ensure that all elements of the project are accessible to this target audience. Programming for the self is an easy trap to fall into, but one that needs to be avoided at all costs when it’s not applicable.

Religious blindness

"All too often, people wear their technical affiliations on their sleeve (or perhaps on their T–shirts)" — Tim O’Reilly [7].

Since the beginning of the hacker age, programmers of all types have been "unfortunately intolerant and bigoted on technical issues" [8]. So when it comes to devout Open Source programmers, there's a strong tendency to immediately reject all proprietary software and anything to do with non–Open Source programs. Because of the philosophical and sociological issues behind the Open Source movement, this resistance is particularly stubborn.

While this has the advantage of increasing Open Source software usage amongst programmers themselves, unfortunately it has the side effect of preventing the Open Source community from learning what proprietary software has to teach. Concepts invented in the world of proprietary software are automatically rejected on the assumption that there’s nothing that could possibly be learned from those who are competing with their movement.

Concepts invented in the world of proprietary software are automatically rejected on the assumption that there’s nothing that could possibly be learned from those who are competing with their movement.

There are still many things which software available for the Windows operating system does better than any present Unix–based system. Rather than admit that Windows is ahead in some areas, the tendency is to just ignore those particular areas. Likewise, since Apple has had so much success with Mac OS X, the Open Source community should be

investing significant amounts of time and effort into cloning Apple's good work, rather than insisting that Gnome and KDE are just as useful.

Fighting for one's political stand is an honourable action, but refusing to acknowledge that there might be weaknesses in one's position — in order to identify them so that they can be remedied — is a large enough problem with the Open Source movement that it deserves to be on this list of the top five problems.

Concluding remarks

I believe that the Open Source movement has produced vast quantities of valuable software, as well as raised public awareness as to issues like open access and open content. The idea of freedom of information has been tied to the hacker culture since their beginning at the MIT AI labs [9], and remains an issue of much discussion today.

Open Source software is being increasingly used in developing nations, where the cost of proprietary software is too high, and by governments around the globe, who are resisting reliance on a proprietary company.

However, Open Source technology continues to remain foreign to the large majority of computer users. In order to accommodate these users, I believe that the five issues I mentioned must be seriously addressed and actively resolved. Due to the increases in Open Source usage, these changes should take place as soon as possible. They should be resolved before those sampling what Open Source has to offer decide to switch back.

Some of the issues' resolvement would simply require a change of mindset, and some of them entail more work, but in the end what they would produce is what should always be the primary goal anyway: high quality software. FM

About the Author

Michelle Levesque is a researcher for the Citizen Lab at the Munk Centre for International Studies and a student at the University of Toronto's Computer Science department. Her job includes designing and implementing programs to enumerate and circumvent state–imposed Internet content filtering. This past summer she and two colleagues traveled to Guatemala and Chiapas, Mexico to study how computer technology — especially Open Source technology — can be used to benefit civil society in developing nations. The trip has been made into a documentary entitled *Hacktivista.* Please direct comments to: ml@cs.toronto.edu

Notes

1. In this paper, I use the general term *Open Source*, though often I'm exclusively discussing Free Software. As well, when I use the term *Open Source projects*, I'm usually referring to projects that have a contribution base wider than one or two individuals. I'm also aware that some companies release Open Source versions of their software, and though I certainly appreciate their donation, I'm excluding these Open Source projects in this particular paper's definition of *Open Source*, as some of my statements do not apply to them. I made these generalizations for the point of simplification, and not for any political motivations.

2. Mozilla FireFox at http://www.mozilla.org/products/firefox/, accessed 20 February 2004.

3. *Debian Installation Manual*, Chapter 3.

4. "Reports from lots of users is unusual too; my usual rule of thumb is that only 10% of users have any idea what newsgroups are (and most of them lurk >90% of the time), and that much less than 1% of even Mozilla users ever file a bug." Nakakoji *et al.*, 2002.

5. "They just don't like to do the boring stuff for the stupid people!" — Bruce Sterling, 2002.

6. "While hackers can be very good at designing interfaces for other hackers, they tend to be poor at modeling the thought processes of the other 95% of the population well enough to write interfaces that J. Random End-User and his Aunt Tillie will pay to buy." — Eric S. Raymond, 1999.

7. O'Reilly, p. xii.

8. *A Portrait of J. Random Hacker.*

9. Levy, p. 7.

References

"Debian Installation Manual," at http://www.debian.org/releases/stable/i386/ch-preparing.en.html, accessed 20 February 2004.

Steven Levy, 1984. *Hackers.* New York: Penguin.

K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida and Y. Ye, 2002. "Evolution Patterns of Open–Source Software Systems and Communities," *Proceedings of International Workshop on Principles of Software Evolution*, at http://www.kid.rcast.u-tokyo.ac.jp/~kumiyo/mypapers/IWPSE2002.pdf, accessed 22 March 2004.

Tim O'Reilly, 1999. "Forward," In: Jon Udell. *Practical Internet Groupware.* Sebastopol, Calif.: O'Reilly Associates.

"Portrait of J. Random Hacker," at http://info.astrian.net/jargon/A_Portrait_of_J._Random_Hacker/Weaknesses_of_the_Hacker_Personality.html, accessed 20 February 2004.

Eric S. Raymond, 1999. "The Revenge of the Hackers," at http://www.oreilly.com/catalog/opensources/book/raymond2.html, accessed 20 February 2004.

Bruce Sterling, 2002. "A Contrarian Position on Open Source," at http://www.oreillynet.com/pub/a/network/2002/08/05/sterling.html, accessed 20 February 2004.

Editorial history

Paper received 1 March 2004; accepted 15 March 2004.

Contents Index

Fundamental issues with open source software development by Michelle Levesque
First Monday, volume 9, number 4 (April 2004),
URL: http://firstmonday.org/issues/issue9_4/levesque/index.html