The Rise and Fall of Open Source

Or: forkbombing an OSS community project

Tonnerre Lombard

Nov 5th, 2006

1 Product overview

In the beginning, there was the source. There were no regulations whatsoever on it, so it was de facto free. The source code of almost all software projects was freely available on the newly created internet, and was shared on tapes between companies and universities. All gained knowledge was contributed back into science and research.

It was only in the late 70s that a different paradigm started to come up. Commercial software was being developed in large amounts by companies who refused to contribute back to the community they benefited from. However, the open source movement prevailed and grew larger over time. Documents were written and books were published in order to describe the paradigm which came as a big surprise to most of the upcoming information technology corporations.

However, over time, Open Source became involved a lot in the competition that was caused primarily by the burst of the dotcom bubble. Suddenly, a lot of IT companies were fighting for their lifes, and Open Source lost a lot of corporate support. Suddenly, a lot of projects had to come up with their own resources and advertisement.

Another thing to observe during the time was that a lot of projects started forks for dubious reasons. This lead to a major vacuum of resources. Also, some projects forked off a variety of child projects, consequently undermining all efforts to create an useful product. In a vast amount of areas, the closed source products overtook the open source counterparts in terms of functionality because innovation was basically stalled.

This had severe implications. While the «unanimous» Open Source browser FireFox is still leading the browser market (because Microsoft has failed to catch up with the features so far - this might change with the arrival of Internet Explorer 7), thus marking itself as a flag ship of Open Source technology along with OpenOffice.org, Open Source systems have lost a big market share in different areas, where the competitors simply brought up better products. These areas are usually covered with a vast amount of Open Source projects, none of which provides the required features. This is also one of the major reasons why a big share of the embedded market was lost to VxWorks.

In 2006, we were facing probably the most massive thinning of Open Source projects so far. But maybe for the first time, even a significant number of major projects ran into a similar crisis. Now that the problem has already reached the backwaters of the seas of Open Source, it is time to look deeper into the reasons.

2 Strengths

The biggest and most advertised advantage of Open Source is, of course, that it gives every single person in the Internet (and elsewhere!) the possibility to contribute to the best of their ability. If anyone finds that a line of code in the entire source tree ought to be changed, he can check out the source, make his change, test it and then decide whether to publish a patch or whether to keep the change to himself. Keeping it to himself can also be an intelligent choice in cases where the change is mainly an adaption to the local system of the user.

However, there is already a resource drain happening in this place. Most of the Open Source developers will know these people who find a bug, fix it and forget to send in a patch, because they're busy fixing applications left and right while the one which broke was actually in the middle. So it would be wrong to assume that even a majority of patches ever see the light of the day.

Another thing that happens from time to time is that the developer or group of developers around a project decides not to accept the patch. In this case, the patch will be changed, abandoned or converted into a new concurrent source project. In the last case, the submitter of the patch or sometimes even a heavy user of it takes the original source code, applies the patch and publishes his source tree along with the changed documentation on a different place. This process is called forking.

Another advantage is, of course, that Open Source software is usually not tied to any marketing strategies (however, in some cases, it is). This means that there's usually no pressure on the maintainers to get the product out on a certain date that marketing has announced, or with specific features. In the closed source world, products are usually fixed for a certain date and have to come out that same day in the early morning, no matter whether it has passed thorough tests before that time or whether there are even still known bugs in it.

This, however, doesn't mean that a roadmap for Open Source projects would have to be considered impossible. In fact, it is very well possible to promise a certain functionality for a certain date. This promise can only be made on the basis of what the core developers of the project are capable of creating until that date. There is however the possibility of a «positive surprise» if more developers join in, because suddenly you deliver more functionality than you originally promised.

Another advantage of this dynamic development model is, of course, that you are free to release patches or bugfix releases at any point in time. Whenever a bug occurs, it is very likely that the person discovering it has already come up with a patch for it, and if not, it's usually not hard for a dedicated group of developers to find it. Once the bug is fixed, the patch/bugfix release goes out and gets implemented quickly by the users and distributors.

The last advantage that is going to be mentioned here is motivation. Open Source is normally volunteer driven and sometimes supported by companies. This means that all contributors are usually highly motivated to produce their software, which makes them more focused on the issues. A wise man said that a man defending his house and family outmans 10 paid soldiers. This is also the case in the software industry: Open Source producers usually work a lot faster than paid developers.

However, like always in life, some of these advantages have their downsides.

3 Exploiting the paradigm

Forking off software involves a serious decision. In fact, forking software can almost be compared to a divorce: all the goods get divided and distributed in some way over the two development groups. The household or maintenance cost however remains the same for both parties. Also, a fork means that both groups have to readjust their equipment, because usually they lost some of it to the «other group».

Probably the biggest reason people see to fork a project is the fact that developers tend to disagree on a lot of things. Usually, it all starts with a clash of interests. Some developers decide that they don't want to continue the development of their product under the current circumstances. The reasons therefor are various and thus covered in the next section.

Open Source is in itself designed to make concurrent development from independent parties as simple as possible. Thus, most of the tools of today are designed to allow a code base to be cloned easily. The most modern source control systems go even further and omit the implementation of a central server.

This is good if some developer goes on vacation and takes his laptop with him - (s)he can make incremental changes and develop concurrent features without needing an internet connection. After a while, the changes can be merged back into the head revision individually. The developer gets the comfort of a source control system without the need for a permanent connection.

This does also mean, however, that the cost involved in forking off a new project is significantly diminished. In fact, it is already built into the modern source control systems. A working revision of a repository is in fact a fork of it, and each source mirror could be used for concurrent development. Thus, a fork mostly happens by someone mirroring the tree and putting it online in some other place.

But at this time, the doubling of maintenance cost which was mentioned above kicks in. This means that less resources are available to do actual innovation and more resources are required for fixing security holes, janitorial tasks and normal bug hunting. Depending on the number of people working at the project, this can mean tat innovation is slowed down significantly, halted or even negative – if insufficient resources are available for basic maintenance, the project becomes gradually unuseable.

This is of course quite useful to the closed source concurrence. If innovation of an Open Source project is effectively stalled, it is easy to reproduce all features provided by the software and add just a few new ones or clean up the interface, so people will go for the commercial product because it is, from any point of view, better. It is indeed very hard for the closed source software selling companies to compete with a vivid Open Source project, because the stream of innovative ideas in the Open Source community is indeed much stronger than it is in the world of closed source development.

In such cases, it might prove very effective for a closed source software producer to send someone out to contribute to an Open Source project. At some point in time, they might decide that it has now reached a business compatible state. At this point, they might provoke a fork of the project by exploiting the psychological vulnerabilities outlined in the following chapter. Once the community is split and everyone is forced to decide which part of it he belongs to, it is very hard to undo the split because most of the time, the fork also involves a lot of hostilities.

At this point, the closed source vendor only has to reproduce the current functionality of the product and give it a new design – yes, a lot of Open Source user interfaces suck. The vendor ends up with a best seller, and the Open Source community earns, well, nothing.

4 Vulnerabilities

There is a very simple set of common disagreements that seem to be considered severe enough to start a fork.

In some cases, developers simply disagree over the adoption of a new technology into the project. In these cases, some developers decide that the new technology is useful to the aims of the project, and want to embrace it immediately in order to make the project as a whole more fancy. However, the other fraction of developers usually doesn't like the idea of adopting the new technology, and doesn't see any point in doing so. This group then decides against the use of the new technology.

Most of the time, the majority group decides the way the project will go, and the minor group either accepts the decision, or forks off a child project which embraces the opposite of the decision. However, in some cases the minority tends to win because they're in control over the servers or the release engineering process. In these cases, a fork is much more likely, of course.

Another possible case is when developers disagree over the use of a source control system. Some projects have been developed historically on the grounds of CVS, but suddenly a group thinks that, for example, Subversion is a much better acronym for their source control. At some point, the Subversion group might decide to fork off a child project which does the same development, but on a Subversion server.

Yet another technical reason is for example what happened in the case of XFree86 and X.Org. In this case, XFree86 was lead by a small core group which decided which patches are allowed to go into the main distributions and which don't. There was even a dictator of the whole tree: David Dawes.

In 2003, a group around Keith Packard continued to restructure XFree86 in order to reflect the state of the art of current graphics cards (especially the acceleration architecture). The idea was also to split XFree86 into small, individual projects which can be upgraded independently, rather than having one monolithic tree. However, the rest of the core team didn't like the idea, and after some issues, Keith was ejected from the team for «conspiracy». At this point he decided to create a real conspiracy, and forked off the X.Org project. X.Org received vital updates that the XFree86 project refused and was soon accepted by all distributions as the new X server for UNIX like systems[1].

This type of forks, which are done because innovation was stalled by a not-so-benevolent dictator, are probably the only positive reasons to do a fork.

However, the probably worst reason to do a fork is personal dissent. There is a number of «alpha geeks» out there, and some of them don't like each other because they feel that instead of cooperating, they ought to be competing. Sometimes this leads to rapid evolution, but sometimes this leads to forks.

It seems that there are plenty of reasons for developers to start disliking each other. Sometimes there was a personal tragedy, sometimes it's just that the community shows more understanding for the one than for the other. Usually, this leads to envy, envy leads to a technically minor clash and this clash leads to a fork. This fork usually has the consequence that two groups try to fight each other, which decreases innovation dramatically.

Even though these disagreements are so simple, though, it seems that the majority of people aren't paying enough attention to them and trying hard enough to get out of their way.

[1]Currently, only NetBSD and MirOS still use XFree86. NetBSD uses it optionally due to its size and MirOS due to dissent of the maintainer.

5 Similar vulnerabilities

5.1 Rewrite competitors

Sometimes, the newly raised competition to Open Source project isn't raised from the project itself, but from people who disagree with the original project and start up a rewrite. In these cases, the impact is even higher. Not only are resources drained from the first project because people tend to run over to the other one, but the entire development effort is duplicated.

Probably the worst example of this is OpenSSL with its rewrite competitor GnuTLS. OpenSSL is a full cryptographics implementation which includes implementations for the standards SSLv2, SSLv3 and TLS. It is BSD licensed and due to its ease of use, it has been the default SSL implementation for Open Source projects for years.

However, there was a group of people who got annoyed with the fact that due to regular security audits, OpenSSL had to be upgraded every now and then because security holes were found. Also, they didn't like the fact that OpenSSL was under a BSD license rather than the GNU GPL. Thus, they created GnuTLS and the gcrypt API. However, most of their implementations aren't properly audited yet, because the GNU philosophy doesn't require extensive audits of the GNU software. The GNU people believe in the thousand eyeball principle, and thus believe that merely publishing their API will give it the required audit.

Howeer, the knowledge about proper SSL implementations which has been gathered over the years by the OpenSSL team isn't given in the GnuTLS team. GnuTLS is currently showing just about the same vulnerabilities that OpenSSL did during its very first years. Cryptography is a very difficult matter, and some vulnerabilities simply consist in a difference between the time of connection setup and the time that an error message is returned.

Even worse: while OpenSSL tries to create a transparent read/write API for the programmer so he doesn't need to be aware of SSL and its implementation details, GnuTLS attempts to be more low-level, so a lot of developers are required to gain a certain knowledge of cryptography as well, which they can't possibly be trusted to get right. This is probably not a good idea, and it is mainly caused by the fact that the profound knowledge that went into OpenSSL was simply neglected, rather than extended.

6 Threat mitigation

There are a lot of things that can be done to mitigate the threat of a project fork. For the developers, for example, it is very important to differenciate between personal dissent and technical problems. A lot of forks are done based on personal dissent, while technically it would have been better for the project (and the community), if no fork had been done.

Another thing to do on the developer side is basically diplomacy. There's no reason to insult people on a personal level if they just made a technical mistake, and there's no reason to insult maintainers on a personal level merely because they weren't content with your changes. Even though this type of insults may appear to be «cool» to some people, they actively serve to drive a developer community apart.

But there are also things maintainers can do specifically. It is, for example, a great relief on the pressure to fork, if different experimental branches exist, where different «new things» can be tried, or different optimizations can be made. In most cases, some of these branches will prove worthwile, and can simply be merged into the stable branch after thorough testing. There's just no «one and only way» of doing things. People will do things their way, and if you try to stop them from doing that, you might lose them entirely. Remember, a different tree from one project isn't as much damage as a fork.

In the real world, there is already an established mitigation model in a set of projects: the BSD community. There are several BSD projects which seem to be competing to an unknowing audience, but in fact there is a lot of cross development. The BSD community does something which could be called *managed diversity.* There are several projects which have specialized for different operating areas: OpenBSD, which is a BSD operating system for routers, NetBSD, which is for desktops and embedded platforms as well as for platform independent applications, and FreeBSD, which works as an entry point for people migrating from Linux environments, and has specialized on x86 based workstations.

Besides the three mainstream projects, there are a number of parallel projects which are mainly customizations of one of the main projects (DesktopBSD and PC-BSD, the FreeBSD clones with an Ubuntu like flavor, pfSense, the FreeBSD/OpenBSD embedded router for home and small office use, DragonFlyBSD, the micro kernel FreeBSD clone, etc.)

The point of these BSDs is that there is a certain type of optimization, but there is also a lot of cross development taking place. For example, device drivers are normally reverse engineered by the OpenBSD people and ported to FreeBSD and NetBSD, or ported from Linux by FreeBSD and from there to OpenBSD and NetBSD. However, FreeBSD does, for example, lack a strict server infrastructure, which would be pointless to have on a desktop. OpenBSD, on the other hand, isn't that heavily equipped with loadable kernel module support (which is much more advanced under FreeBSD), because on a server, LKMs are a bad idea in the first place.

This way, the specialization and customization of the projects is still given, while the overhead of development induced by the separation of the projects is fairly low, leading to a vast amount of resources available for innovation.

This method of mitigation has already been stress tested several times. The last stress test was performed by Charles Hannum (mycroft) on the NetBSD project[2], and it went perfectly well. In fact, it created a counter movement to the original message (*NetBSD is dead*), by calling out for two bug fixing and a current release rebranching session, all of which seem to have gone well, making the NetBSD project even more healty and vivid.

7 Discovered by

Tonnerre Lombard <*tonnerre@bsdprojects.net*>

[2] *http://mail-index.netbsd.org/netbsd-users/2006/08/30/0016.html*