The Rise and Fall of Open Source
Or: forkbombing an OSS community project

Tonnerre Lombard

Nov 5th, 2006

1 Product overview

Initially, all software was de facto free. Companies and universities shared the source code of their products on the newly created Internet and on tapes. The closed source community was created only 10 years later, but even though these corporations didn't contribute back to their community, Open Source grew larger and finally became a movement, with all the books and documentations involved.

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. A lot of projects had to come up with their own resources and advertisement by then.

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. 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 the possibility to contribute. 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 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 problem that may occurr is that the maintainer decides not to accept the patch. In this case, the patch will be changed, abandoned or converted into a new competing source project, thus creating a fork.

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 is 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 can create 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 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 so, 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. 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

A fork of a project always comes with a drain of resources. Just like with a divorce, all goods get divided and distributed over the new communities. The maintenance cost however is doubled, and some equipment must be replaced because it got lost 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, thus allowing for non-central branches and offline development. This means that a source checkout is in fact an operation equal to a fork, unless a merge happens later.

But in case of a fork, 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 that 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, a closed source software producer could 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, and tell their mole to provoke a fork of the project by exploiting the 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 is outplayed.

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, a number of 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 doesn't like the idea of adopting the new technology. This group then decides against the use of the new technology. The majority group decides the way the project will go, and the minor group either accepts the decision, or forks off a child project. (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.

Yet another technical reason is when a stall of innovation occurrs due to an unneededly restrictive maintainer (usually a dictator of the project tree). In this case, the lack of innovation gets increasingly significant, and a fork is very likely to happen with the patches that were refused. This is what happened with XFree86 and X.Org, and is probably one of the only beneficial types of 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 enemies (due to envy, with a technically minor clash as a given reason). Sometimes this leads to rapid evolution, but sometimes this leads to forks.

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.

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.

In some cases, this even has severe negative consequences, if, for example, the technical knowledge and experience is gathered in the original project, while the other one has better marketing. This may lead to bad products being used all over while the better concurrence doesn't attract much attention.

6 Threat mitigation

There are a lot of things that can be done to mitigate the threat of a project fork. For the developers, 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.

Also, 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.

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. There's just no «one and only way» of doing things. Always remember that 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.

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. 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.

7 Discovered by

Tonnerre Lombard <*tonnerre@bsdprojects.net*>