CASE STUDIES II: Communities in Open Source Projects

Sergio Arroutbi Braojos

May 4, 2014

Contents

1 Introduction

This report is an intention to show which are the main concerns for different important people of the Open Source environment from the Community perspective. Through a set of different Technical Talks, the main aspects regarding Community Management in Open Source, creating smart Communities, avoiding disturbance elements, performing correct Open Source Project management and, in the end, create a nice Community environment in order to have more possibilities for that particular project to succeed.

Very important people, such as Greg Kroah-Hartman from the Linux Kernel Project, or Stefano Zacchiroli, former leader of Debian GNU/Linux project, share their views regarding Community of their particular projects, as well as some other interesting aspects around them. Apart from well-known names in Open Source universe, as previous ones, other not so well-known names will share interesting information around other not so well-known Open Source projects, such as EdgeBSD, Mercurial or SilverCMS. Special interest in order to study Community focus and ways of avoiding "dangerous / poisonous" people is another Tech Talk, and will also be summarized in next sections of the document.

Last, but not least, this document will try to reflect the main aspects brought to foreground by each of the lecturers, as well as the main references, behaviors and ways of working followed by each of them.

2 FreeBSD

2.1 Introduction

On this Tech Talk [1], Robert N.M. Watson, from Cambridge University, talks about FreeBSD project, and how the community behind this project works. Questions as what is FreeBSD and FreeBSD project, what do you get when you get a FreeBSD distribution, or how does the FreeBSD project work in terms of contributors or licensing are responded on this talk, which is not focused in any particular technical feature of the project, but rather on its general way of work.

2.2 FreeBSD Foundation

Behind FreeBSD project, the FreeBSD Foundation exist, in order to support the project on different ways, but always without being implied on the technical decissions happening on the project, but rather with:

- Sponsor development
- Hardware purchase
- Collaborative R&D agreements
- Developer travel grants and sponsorship
- **Sponsor the project economically, by obtaining donations**

2.3 Project Necessities

On the talk, apart from exposing all the stuff that the project "produces", such as the FreeBSD kernel, the releases, the ports collection (packages), the documentation or support. However, on this talk, it is alsoexposed what the project "consumes", as a way of identifying the project necessities, to be pointed out the following:

- **Hardware**. Donated and sponsored, specially racked already prepaired hardware.
- **Bandwidth**. Needed in vast quantities, as in any other project, but even more for Operating System Open Source projects, where, every six months in this case, lots of people from around the world download tons of terabytes to test the new releases.
- **Financial Resources**. Travel grants, salaries, contracts and other type of grants.

- **Gratitude**. Thanks, good press, user testimonials or success stories are welcomed as well.

2.4 Community on FreeBSD Project

2.4.1 Committers

People with commit rights. They come from different 34 countries, from 6 distinct continents. The main development groups are in Nort America, Europe and Japan. There is also an important group of contributors from India and Australia.

Regarding ages, 30 to 40 years old group is the most large, with a mean age of 32.5 and a median age of 31.

In terms of occupation, there is a group of heterogeneous jobs such as professional programmers, hobbyists, consultants, university professors, researches or students.

Committers must have certain key characteristics:

- Technical expertise.
- History of contribution.
- Ability to work well in the project community.

2.4.2 Ports Committers and Maintainers

They are responsible of maintaining the binaries of the different software pieces available in the project. In 2006, 158 port committers existed, and over 1400 port maintainers for 16600 ports were active in the project. As average number, 85 ports/committer, 9 ports/maintainers and 8 maintainers/committer existed.

2.4.3 Mentors

As in other open source projects, mentors support people starting in the community. They propose new people to core group, or other group. A key factor is that people, apart from demonstrating the technical abilities, demonstrate how commited are with the community.

2.5 Events

There are two type of events regarding FreeBSD Project:

1. **Conferences**. BSDCon, MeetBSD, EuroBSDCon, AsiaBSDCon, BS-DCanada, etc.
2. **Developer Summits**. Two Day Events, normally three a year, one in North America, one in Europe, one in Asia. e.g.: in 2007, conferences in Ottawa, Tokyo and Copenhagen occurred. Focused in development and developers to know each other.

2.6 Conflict Resolution

Developers are normally common sense, easy to cooperate, independent and neutral. The conflict resolution is normally avoided by intentionally skipping overlap between members of the community.

However, strong technical disagreements will occur, involving technical and personal conflicts. If conflicts get out of hand, normally a member from the core group mediates to avoid flames.

There is a special kind of conflict, the "Bike Shed" [2], that appeared for the first time on this community, and that is a good example on a type of conflict that must be avoided, as this kind of conflict consist normally around unimportant stuff that is handled with many strong opinions from many different people in the community, meaning lots of discussion as well.

2.7 Conclusion

Free BSD is one of the largest, oldest, and most successful Open Source projects, built on millions of lines of source code from hundreds of committers and thousands of contributors. There are tens of millions of deployed systems hosting FreeBSD Operating System. The highly successful community model makes this community a very desirable project for any Open Source advocate to participate. Join the community!!!

3 How Open Source Survive Poisonous People

3.1 Introduction

On this Technical Talk [3], dated on June 2007, Ben Collin-Sussman and Brian W Fitzpatrick, who are Senior Software engineers in Chicago, and work on different Open Source projects, analyze how communities can cope with those poisonous people who some times appear on the Open Source communities.

Speakers consider protection of the comunity to be vital, in order to let the comunity focusing on the objectives on the project and avoid disruptions. In order to do so, speakers propose a set of stages, four in particular, that the project managers need to consider in order to face these disrupting elements. The four stages proposed are:

- Comprehension
- Fortification
- Identification
- Disinfection (considering this how to name the fact of getting them out the comunity)

Poisonous people appear sporadically on the Open Source project development. These kind of people can be identified, as they normally appear in order to disrupt the normal evolution of the project, by carrying out actions such as:

- Trying to distract the comunity with unimportant issues
- Performing emotional drain on the comunity
- Attempting needless infighting

An important aspect highlighted by the lecturers, is the clarification on how sometimes poisonous people act in that way without willing it. However, it is important to identify the disrupting elements and try to fix the issue as soon as possible.

Lecturers start by recommending Karl's Fogel "Producing Open Source Software" book, which goes into technical and no technical details about how Open Source Software must be produced, by showing practical examples on two very important Open Source projets, such as Subversion and Apache.

During the presentation, lecturers face different steps , shown below, that must be considered to try to protect against poisonous people, and if the “infection” is produced, how to get them out.

3.2 Fortifying the comunity

Lecturers talk about the importance of fortifying the comunity. Get the comunity focused, keep a good direction, mantain a pleasant environment, avoid “Bikesheds”. Basically, four elements are considered the most important to keep the comunity fortified, as much fortified, the better. Aspects such as:

- Politeness
- Respect
- Trust
- Humility
- Focus

Apart from previous aspects, it is important, for the comunity, to have a mission. Project managers must limit the scope and set a clear direction. e.g: ”Have a compelling replacement to CVS”.

The other important thing is to focus on the mailing lists netiquette rules. Aspects such as:

- Don’t rehash old discussion
- Don’t reply to all the mails
- Avoid writing on capital letters
- Avoid personal referrals
- Use of correct language

and in general all the rules described on the RFC 1855, are important as well to keep a polite environment in mail lists and other project channels.

3.3 Keep Project History

It is important to maintain the focus of the comunity. This aspect is highlighted by the lecturers. However, lecturers also insist on the necessity of "Not losing the Project's history", as a way of transparency and decission making help. The project management must keep documentation on different areas such as:

- Design decissions
- Bug fix history
- Mistakes made
- Lessons learnt
- Code changes

The main idea is to enforce the transparency and way of working of the comunity, and protect it from territoriality of project areas, considering that Open Source projects comunity members change as time goes by, and the work tasks and other aspects must be documented.

3.4 Healthy code collaboration

In lecturers opinion, code collaboration, revision and validation is considered to be one of the most important aspects for acomplishment of the comunity project objectives. Some considerations must be taken into to achieve them:

- Commit emails are the best way for people to know what is doing each-other
- Code reviews are there for people to know what you are working on. If they have interest or knowledge on the stuff, they will review them, but if not, at least they know the area you are now working on
- Big changes: on new branches for easier review
- Increase the "bus factor". Try to spread the core knowledge as much as possible
- Dont allow names on files (AUTHORS.txt, whatever, but no on the file). This avoids territoriality on the files

3.5 Well-defined processes

Processes are important, more important as the project grows up. Important aspects that must be faced for a community to protect against disruptures and external confussion, are, for instance:

- Bug fixing process
- Test of release images
- Patch Proposal / Reviewing / Acceptance
- New committers advertising
- Define committers permission politics

3.6 Voting

Contrary to what should be thinking regarding voting decissions on the Open Source Projects, voting should be considered as a last resort. Each comunity is a little bit particular regarding voting and internal democracy. There are very different approaches regarding this topic. However, lecturers consider that healthy comunities need rarely voting to make a decission, as the decission is normally easy to make considering comunity rules.

3.7 Poisonous people identification

Identifying poisonous people sometimes is obvious, but, other times, it is not so. However, lecturer provide some clues to identify quickly these kind of poisonous elements in the comunities, e.g.:

- Write on capital letters
- Have non-sense login names / mail addresses
- Use excessive punctuation
- They express themselves on a non-readable sense

However, sometimes poisonous people do not fit into previous characteristics. A more accurate approach related to poisonous people characteristics is proposed by the lecturers, as they normally are:

1. Hostile people
    - Insult the status quo
    - Ask for help on an angry way

    - Try to blackmail
    - Attempts to deliberately rile people
    - Conspiracy accusations

2. Lack of cooperation

    - People complain, dont help fixing anything
    - Unwilling to discuss design
    - They do not accept criticism

3.8 Disinfecting the community

There has to be a plan to assess the damage. Some questions must be performed, to be sure of the people disrupting on the project:

- Is this person draining attention and focus?
- Is this person paralyzing the project?

At the same time, some things need to be avoided:

- Do not feed the energy creature. They are looking for a fight. First thing to do, ignore them, for them to ssee this kind of behavior is not taking into account.
- Do not give jerks a purchase.
- Do not engage them.
- Do not take things from the emotional. Stay to the facts.

Meanwhile, these aspects are considered as good practices:

- Do pay attention to newcomers, even if they are initially annoying.
- Do Look for the fact under the emotion.
- Do Extract a real bug report, if possible.
- Do know when to give up and ignore them.
- Do know when to forcibly boot from the community, although some times is not an easy decission (e.g.: people writing constantly to a dev-list because have a lot of interest, but that is in the end draining the focus of the community).

3.9 Conclusion

As a summary, the lecturers group the procedure to clean the disrupting poisonous people in four steps:

- Comprehend: Preserve attention and focus of the comunity.
- Fortify: Build a health community.
- Identify: Look for tell-tale signs of possible trolls.
- Disinfect: Mantaining calm and standing on your ground, without involving emotionally.

4 SilverStripe CMS

4.1 Introduction

This speech is provided by Zigurd Magnusson [4], mainly, and Sam Minne, secondarily, both belonging to SilverStripe company, on 1 August 2013, for Google Tech Talks. The talk is centered, on the one hand, in software production on New Zealand, where is placed SilverStripe company. From the community perspective, knowing different zones of the world is important, to characterize the social and economical environment where FLOSS is produced.

On the other hand, the talk is centered on SilverCMS, the reasons why SilverStripe decided to launch this CMS as an Open Source project, and how the mission of managing people becomes a particular aspect on this kind of project.

4.2 New Zealand

Zigurd Magnusson starts by introducing the country where both lecturers come from, New Zealand. Regarding the country, the speaker highlights some characteristics from the country, for instance:

- Population: 4 million people (much less than the important states of USA). Similar to Oregon population.
- History:Lecturer describes New Zealand as an "Old country". Settled 800AD with pacific islanders. European discovery from 1634, received massive immigration from 1840.
- Considered to be a liberal country with liberal politics. The Prime minister (August 2007) was a woman.
- It has a small and remote economy: The economy is, in fact, very dependant on exports. For that reason, it is a country which must innovate in fields like: - Wine, Tourism or Movies... - ... and Of course, Software.

Regarding the main New Zealand Software production, speaker talks about next contributions:

- Jade: A programming language that provides a user interface to create classes and carry out other programming stuff.
- Aftermail: A company that provided email management platform.

- Xero: A company that develops SaaS (Software as a Service).
- Peter Guttman: Who has worked with PGP.
- PlanHQ: A web based Business Plan Software.
- Catalyst IT: A company that provides services around Open Source Software, such as Moodle, Drupal, Mahara or Kora.
- The most important use of FLOSS in New Zealand is the voting system: This system is an application written in Perl/Mason on top of a PostgreSQL database.

4.3 Silver Stripe

Silver Stripe is the company that the speakers belong to. The company was, in 2007, composed of 18 people, and is, basically, dedicated to get money by providing web building services. As a rule of thumb, the company staff had to dedicate 20% of their time to the Open Source products of the company. The main product of the company is SilverCMS, which is a Content Management System Open Source Project that provides a product that:

- Is a more complete tool than a blogging system.
- Focused on usability for the site manager.
- Out of the box Framework, which provides a complete solution but with an easy installation process.

In 2011, the system was the 15th most downloaded Open Source CMS, as shown by the Open Source CMS Market Share report [5] Besides this, the speaker describes why SilverStripe took the decission of developing their main project as an Open Source Project. The main reasons to do so where:

1. Transparency: advertising the product by showing it (promote downloads, demos, etc.) rather than advertising it.
2. Way to market: Cheaper, more effective, due to previous reasons.
3. Restitution: Similarly, there was an important reason for creating Open Source Software. As Zigurd Magnusson states: "We were in debt ..." : So, the company stuff considered that, as a way of restitution, they had also to provide it. This aspect is very related to community, as commnunities in Open Source enforce to promote Open Source Software creation and restitution to the FLOSS society by companies.

The negative thing of chosing this kind of software develpment strategy, was to know how to make money. Related to this issue, the lecture asserts: "People love our product, but dont want to pay for it".

4.4 Managing people

Zigurd Magnusson faces an important matter related to the community building, as it is the issue of task force management. To start with, Magnusson makes a strong recommendation of Karl Fogel's "Producing Open Source Software" [6]. It is not the first time than an Open Source advocate recommends this book [3], when talking about staff or community management. From Magnusson's perspective, the book was amazing. He read it in 24 hours, discovering that he agreed 100% on its content.

Indeed, Fogel's book is considered to be a Bible, and should be mandatory read by people managing software producers, even more if related FLOSS software producers [3]. In the end, producing open source software is not only centered on Open Source communities, but rather on a more wide concept around Open Source Software production, from different perspectives and on a more general view rather than what is exactly Open Source project and communities management.

In order to manage correctly people around Open Source software, some important aspects are considered by the speaker, for instance:

4.4.1 Publishing guide principles

In order to articulate all that common facts that were true on both company staff and contributors, but did not appear in any place. This lack of procecure could drive to:

1. Confuse and not differentiate roles inside the product.
2. Pain, due to people working on unneeded tasks.

4.4.2 Give recognition

From lecturer's perspective, recognition from management of the project is a very good practice, given that:

1. It is important to let people know when a good job has been carried out.

2. It is also important to reward the best workers, but must be precise on why or why not a prize has been given to a people or group of people.

4.4.3 Work delegation

As the time goes by, Magnusson has found out that the responsibilities are important to be determined, but without an excessive procedure. For this reason, he promotes the use of:

1. One page overview, then pass the responsibility: As a middle point between no management and very specific management. For this reason, they create a one page overview on how the project should be like, and then pass the responsibility to what the page asserts.
2. For previous reason, there is a community responsible for day to day questions and particular activies around certain tasks.
3. The guiding principles should also fill the edges and corner cases, where the people stands.

4.4.4 Mentorship

1. Mentors are very important for the organization, for starting people to gain knowled as fast as possible.
2. Mentors, however, are technical experts on one or some areas, and normally face high workloads, so it is important than they are considered to reduce their work load to help on mentorship.
3. It is also important to make mentors realize of their mentorship job, recognize it and help to promote it by enforcing their satisfaction due to this job. Fun, results and praise are ways of achieving so.

4.4.5 Design Process

1. Design process must be flexible, and focused on fixing development issues that exist.
2. Designers must not be afraid of making mistakes, but must rather be prepaired in order to fix them quickly.
3. Clever architecting does not replace making prototypes to throw away, rather it consider this as a normal practice within development process. Software moves quickly, new ways of implementing things appear, and architects must be prepaired for it.

4.5 Google Summer of Code

The speaker from SilverStripe focused later on contributions from very different students provided in the Google Summer of Code. Due to this, some important contributions to the project have been provided, in different aspects such as:

- Usability
- Internationalization
- Mashups
- Database Management
- Safari browser supportp
- Image manipulation
- Reporting
- etc.

The important issue here are the lessons lernt by the speaker. Within Google Summer of Code attending, it has been checked that there are some steps to follow when hiring, as they are applied for selecting students. Among these common steps, it can be found:

1. **Selection must be wise**. It is important to provide a coding exercise to inspect how a possible next developer of the project codes, and if she/he is capable of sorting out certain issues.
2. **Project selection**. More focused on GSOC, with this statement Magnusson wants to highlight that projects must be selected considering how smart and simple they are taking into account the skills needed to be matched.
3. **Start coding early**. The programmers must have the oportunity to start coding as eary as possible, for them to demonstrate they are valid to do so.

4.6 Q&A

Last, but not least, the speak ends with a Question and Answer round. In this round, some interesting questions are raised by attendants to the speak. Among them, there is a question having to do with a new movement in New Zealand around Web standards openness.

Sam Mine, the other speaker, clarifies how the interesting thing here is related to the fact that, being New Zealand a short, but tight, FLOSS population, when new proposals, such as previous one, appear, it is a normal thing to get people involved and interested on that particular task. So, what initially can be considered troublesome, sometimes mean benefits on the other hand.

As final question, the speakers explain about the future of the project, the different strategies that will be followed and how the product is thought to evolve in next years.

5 Camino

5.1 Introduction

On this Google Tech Talk [7], dated on January 2007, Mike Pinkerton, software engineer at Google, talks about Camino, an Open Source web browser for Macintosh Operating System. Nowadays, 2014, Camino Web browser has stopped to be developed, but this talk helps on understanding the battle that has been carried out up to date where only some Web browsers have been mantained on the market, and where others, e.g: Camino Web Browser, have resulted on been not developed any more due to market demands and continued Web evolution.

Mike Pinkerton starts a presentation of him, highlighting the different roles he has played as Web browser developers. He has been involved in MacDev group at Google, as software engineer, but he has also worked at Netscape (1997-2002). On the other hand, he has also performed super-reviewer and module owner roles at Mozilla Foundation. He has a strong experience and knowledge in Web Browsers due to previous reasons. Apart from that, he was, by 2007, **Project Leader of Camino Web Browser**, at Mozilla Project, the main reason for him to give this talk.

5.2 Agenda

Pinkerton gives a short presentation on the different aspects that will be exposed on this speech. Basically, the most important things discussed will be:

- Camino Milestones
- Lessons learnt from Mozilla and Camino
- Means of understanding Camino strategy

5.3 Mozilla History

It is important to know the history, in order to explain why there is Camino, apart from Firefox, together with Mozilla. The lecturer goes back to 1998, where only two Web browser existed. On the one hand, Internet Explorer, a free Web browser from Microsoft. On the other hand, Netscape Communicator. Netscape Communicator was not free. And Internet Explorer was winning the game, as it was delivered for free with Microsoft's Operating System. Microsoft wanted to monopolize the market, and was willing to spend any amount of money in order to make Netscape's browser dissappear.

Netscape could not fight against Microsoft, as was a much smaller company compared to Redmond's. So they needed to change the rules of the market, and turn around the strategy where Microsoft to a development model that Microsoft was not going to follow:

```
"One place we knew where Microsoft would not go was Open Source.
And we knew that they won't follow us to Open Source, because
their browser was so tight to the Operating System, that
releasing Internet Explorer as Open Source would involve
releasing part of the Operating System as well."
```

Apart from that, Pinkerton explains that Netscape knew that, being the only Open Source web browser, there would be a part of Open Source advocates, both hackers and developers, that would help defeating Microsoft, by helping developing new features, fixing bugs, and so and so forth. So, from Netscape perspective, **Comunity would help on defeating the competence**, as they were the first to be and Open Source Web browser, and contributions would come. It is interesting how a company can trust on switching to an Open Source based strategy, on top of a strong comunity that from Netscape's perspective was willing to help, in order to reach the company objectives in Web Browser markets, which was, indeed, not being defeated by Microsoft.

By March 1998, the closed source Web Browser code was released as Open Source. That moment was considered an important moment in software history, and even a documentary, called "Project Code Rush", was recorded, and it is a good documentary to follow the history. From lecturer's perspective, "it was fun". It was a very interesting moment, as a strong group was created (it was the preliminar Mozilla).

But, later, when they get back to work, they realize that the project was working. Netscape 5 release was accelerated due to contributions and fix from the comunity, that was contributing, besides this, on a smart way. Engineering group started dedicating more to review code contributions rather than contributing, and also to start thinking of architectural changes that could be performed in order to improve the product. But that moment, from management perspective, Gromit, which was the name for Netscape 5, was decided to be not released, in favor to develop a new from scratch program.

Apart from the impact on engineers, **there was an issue of not communicating to the Open Source comunity**. The only comunication was a not on the bug fixing tool. Obviously, the Open Source comunity was angry due to the decission taken by Netscape management, what caused a loose of credit from the company to the comunity.

Management also decided, for new developments, to keep just one team, as they could not afford multiple development teams for multiple Operating Systems. Development teams realized that there was two kind of roles inside software development. On the one hand, designers specialized on user experience, design, standards, and so on, and so forth. On the other hands, programmers that can code what designers say into software by coding through programming languages. The idea that appeared was to provide designers with tools that could help them to develop their ideas with their knowledge (CSS, DOM, HTML, XML, Javascript, etc.) to build the UI on top of them, and reduce the gap between design and programming. Real UIs could be built from designers, but still tools had to be developed.

A lot of work in front of the development project. They made what they could, the shipped together what they had by 2000, and shipped as Netscape 6. The speaker asserts:

```
"It was crap. Total, absolute, unadulterated crap.
I feel terrible that my name is connected to that
software project."
```

Back to the work, engineers had time to improve performance, reduce memory consumption, polish what was needed to polish, and Netscape 6.1 and Netscape 6.2 were released. This two versions (specially 6.2 version) was much more usable and robust. Netscape 7 was later released and was pretty better too.

5.4 Camino History

However, Gecko, which provided the infrastructure layout engine, was very based on Windows Operating System. It was difficult to make the browser a native browser for the other Operating Systems (Mac specially). There was a lot of work in order to port Gecko to Cocoa, which was Apple's native object-oriented application programming interface (API) for the OS X operating system. Porting Gecko to Cocoa application could help on releasing the browser on Macintosh computers, and look real native.

Mac users are really advocates of the OS X operating system, and want everything on Macintosh to continue working when browsing as well, as fast as in Mac OS, with same menus and key bindings, etc. For that reason, Camino was born, although initially was named ProjectX. This web browser, built on Cocoa and around Gecko, provided that native aspects Macintosh users were willing to use. People from comunity kept an eye on it, as it was promising, although still far from his big brother, Netscape.

When Camino 0.7 (Camino was how ProjectX was named later for legal issues) was released by 2003, people loved it. By that time, lots of users shared it and helped on spreading the software, as it worked really good and provided a very nice user experience on Mac, as it looked real native. It barely crashed, and was really well accepted.

As Netscape wanted to focus on its browser, Camino 0.8 was released to Open Source comunity, as a part of the Mozilla Foundation, by 2004. People, apart from using it, wanted to participate on developing it. Among other stuff, comunity thinks it is a good product and is willing to participate on continuing its development and fixing the issues that are to appear.

Camino 1.0 was later released, in 2006, in order to let the comunity know that the application was a stable application, and "was not going to damage the hard-drive". Engineers wanted to release early, so they decided on an amount of fixes that needed to be fixed, and after being fixed, they would release the 1.0 version. And so they did. The was also a user friendly web providing a lot of documentation on the bwrowser.

5.5 Lessons learnt

After previous events, the lecturer emphasizes on those aspects that must be handled in order to Open Source projects succeed along the time, based on the experience acquired on Mozilla:

5.5.1 Openness

As Mozilla/Netscape had lost a lot of street credit due to previously described decissions, such as discontinuating projects where comunity was involved. For that reason, from that moment, Mozilla needed to recover the street credit by being opener than ever. They wanted to be "open to the point of promiscuity", as Eric S. Raymond stated in the "Catheedral and the Bazaar". And to do so, **recovering comunity confidence**, certain objectives must be accomplished:

- They had to be open in the decissions that were going to be made. People feel incredibly more connected to those projects where they can understand what is going on, and will be more excited about participating on it. People must see the promise of the project, for they to participate but also for they to share with other potential members of the comunity.
- People from comunity must be involved. And they must be involved in decissions, as well as in other aspects of the project such as ways of working, for them to be loyal to the project, they must have voice inside the project.
- It is important not to loose the comunity confidence. If so, it is almost impossible to recover it and regain their trust.

5.5.2 User Centric

Initially, browsers by Netscape were developer centric. It was really hurt to configure the project without strong science computer knowledge, and was much oriented to geeks. Camino was indeed somehow of the same nature, not beeing user centric, and more focused to developers.

The problem of having that strategy and being like that, is that it is very difficult to create a strong comunity. In order to have a product usable from all kind of users, is to have a strong product, focused on the ease of use, and with a very easy and comprehensible documentation focused on non experienced users perspective. If previous aspect is achieved, a much bigger bunch of users can be acquired to use the product.

5.5.3 Weak Ownership

Another lesson learnt on Mozilla is related to ownership. In particular, module ownership on Mozilla project. Module owners are responsible of fixing hot issues of that module, taking decissions on architectural changes, etc. However, initially, module owners were Netscape employees just for the reason that they knew the code.Pinkerton asserts:

```
"If a module owner is not responsive, the whole process breaks down."
```

In general, comunities are very respectful with module ownership. **It is better that a module has no ownership rather than if a module as a weak one**. Without ownership, the comunity will make decissions on the steps to follow when an issue appears on that particular module, but if a module owner exist, no further steps will be performed by the comunity. It is a step back to have a weak ownership for an Oper Source project.

5.5.4 Testers are the most valuable resource

Testers are, from lecturer's perspective, the most valuable resource on the Open Source projects, due to the following reasons:

- Help find issues immediately. They run regressions and find the issues on a quick way.
- Provide reduced testcases. Focused on new developments and features.
- Test in environments and situations nobody would imagine. Different operating systems, with/without proxies, browse for weird URLs, etc.

5.5.5 Open Bugs

Related to previous statement, and the importance of testing, it is essential to maintain an Open Bug database. This will help on:

- Notifying about regressions results. And help on avoiding duplicate work on same issues.
- Provide the comunity about the project stability and quality. This helps incredibly on making the decission of when to release a version.
- Process can be reflected in the bug system. Bug systems are more and more complicated, allowing also to include new feature developments, etc.

5.5.6 Can't please everybody

Open Source projects can not always satisfy all of its members. It is even inappropriate to try to do so. And this is due to several factors such as:

- Not all ideas are good ones. Maybe are good ideas, but does not fit into the project.
- Every opinion must not be taken into consideration just for the fear of losing a contributor. Good contributors are those ones who understand the objectives of the project, and assume the constructive criticism.

- Good managers of Open Source projects are the ones who say “NO”. During an Open Source project life, saying “No” will be more frequent than saying “Yes”, and focusing on the important features is the most important aspect for comunity strengthen.

5.6 Camino present

Last, but not least, after presentation of the lessons learnt, which is the most important aspect from comunity perspective, lecturer talks about the present of the Camino project, which can be summarized in next items:

- Releasing new functionalities, which has been asked by the comunity for a long time.
- Start a more frequent release cycle. As looking for perfection is making the project to start being slow.
- Center on Leopard Operating System. To take full advantage and focus on one particular market.
- Migration to WebKit?. By 2007, Camino was considering of migrating to WebKit, as WebKit was progressing really fast, but Camino users did not agree on how Safari works, and the user experience that it provided in top of it.
- Continue the association with Mozilla Foundation. By 2007, Camino was considering also if it needed to continue being associated with Mozilla Foundation, or separate from them, as Mozilla seems to be centered on Firefox, and other projects seem to have no importance.

5.7 Q&A

On the Questions and Answer round, some interesting questions are brought forward, such as:

1. The number of users that Camino owns: 250000 by 2007
2. The benefits that WebKit provides compared to Gecko: which is the tremendous ease of use of the first compared to the second. Gecko is a very complicated unapproachable system, that needs a high level of rendering knowledge to work with.
3. The number of Gecko bugs standing for a long time: It is a known issue, and need some focus from company and comunity to fix them. However, moving to new technologies can help on some of them to disappear as well.

4. The number of Gecko bugs standing for a long time: It is a known issue, and need some focus from company and comunity to fix them. However, moving to new technologies can help on some of them to disappear as well.

5.8 Conclusion

From comunity perspective, the most important part of this tech talk has to do with two aspects:

1. **The strategy of going to an Open Source project**: As an option to survive against big proprietary competitors.
2. **The Mozilla Foundation lessons learnt**: Some aspects to keep on mind when developing Open Source within a comunity are:
   - Openness
   - User Centric
   - Weak Ownership
   - Testers are the most valuable resource
   - Open Bugs
   - Can't please everybody

6 Greg Kroah Hartman on the Linux Kernel

6.1 Introduction

On this Tech Talk on Google [8], happening on June 2008, Greg Kroah Hartman, who was an important role already by that date, and, who, indeed, has been increasing on importance in the project up to date, gives a talk on the Linux Kernel, emphasizing those aspects that must be considered by both Kernel users, lecture attendants, but also for Google as well. Who makes the Kernel, which is the state of the project in that date, the features of upcoming releases, the stability and quality of the project and other issues are brought forward in this talk.
The talk is around two main aspects from the comunity perspective, such as:

1. **Kernel Statistics**. Information about the number of commits, number of contributors, main companies involved, etc.

2. **Organization and Release Plan**. How is kernel organized? Must I send a patch directly to Linus Torvalds?. Which is the release plan on the Linux Kernel?. This kind of questions are answered during this part of the talk.

6.2 Kernel Statistics

To start with, Kroah-Hartman emphasizes on the “numbers” that Linux Kernel project was handling by that date. In particular, these are numbers **per day** on changes happening on the kernel development, as a mean number, for years 2007-2008 :

- 4300 lines added
- 1800 lines removed
- 1500 lines modified

These numbers are supposed to be of a stable project, which moves faster than any other number. Linux Kernel project supports more CPUs than any other Operating System, supported in more devices than any other Operating System, and is supposed to be continuing on increasing the number of features and devices that will support. The numbers give a **change rate of 3.69 changes per hour**, 24x7.

Apart from that, a graph is shown in order to demonstrate the amount of lines added, removed and modified.From the comunity perspective, these numbers are meaningful of the state of the project, how important it is in terms of comunity and how flexible, quick and fluid software development can be on an Open Source project.

Other important statistics about the kernel, have to do with the number of **LOC (lines of code), which was, by 2008, 9,2 million**, the number of contributors developers, **2399**. Another important factor is how important are contributions, in terms of work distribution. The top of the curve is getting flatter, meaning that work is being more distributed, as in 2005/2006, there was a distribution of 20 people, . By 2008, 30 people was contributing 30% of the work. This is a high increase on the distribution of the work, very welcome by the project management. Apart from that, number of top contributors in number of patches and the number of top reviewers is shown as well.

Other important data has to do with contributions in % by companies. By that date, Suse, Novell, IBM, Intel or Oracle were the top contributors. However, there was an important contribution number by "Amateurs" (18.5 %) and "Unknown Individuals" (5.5 %). Google, by that time, was in position 13, but without Andrew Morton, who was paid off by Google at that time, they would decrease to 40th position. By 2007/2008 up to 27 people had contributed to Kernel.

6.3 Kernel Organization and Release Plan

Having not changed too much up to date, the lecturer showed up the organization of the Kernel, as handling this huge amount of changes is a really very challenging mission taking into account previous numbers. Taking into account the huge number of contributors and developers on Linux Kernel comunity, it is expected that non all of them send their patches to Linus Torvalds in order to be merged into the repository. It is rather, as expected, and as Torvalds has declared more than once, a top-down trustness relationship.

In particular, developers contribute code by sending patches to driver/file maintainers. These maintainers, once filtered the patches provided, send them to subsystem maintainers, who send them mainly to Linus Torvalds. Sometimes, other main roles in the project, as was in 2008 Andrew Morton (or could be, today, Kroah Hartmans himself), receive those contributions directly.

By that date, there were up to 600 driver/file maintainer, who review the code provided by developers. Subsystem maintainers are responsible of “big areas”, such as USB, PCI, VFS, core, etc. The code is then sent to “Kernel Next Tree”, who merges everything and check if the build has broken.

The main idea here is how organized Open Source Project Comunities can be, and shows how being a fluent and quick development software comunity does not mean having no control on what is going inside the kernel.
In the same manner, an explanation of the release cycle is explained. Mainly, once an rc (Release Candidate) version is proposed, there is a pair of weeks to late commits. After that, bug fixing, regression running and tests start. After several release candidates, an stable version is “frozen”, normally in a period between 2 and 3 months, on what means obviously a “release often, release early” strategy. It is also a policy based on a “divide and rule” mechanism, where there is a lot of maintainers specialized in small pieces of this huge project. This means that, although stable version can be updated (Security updates, i.e.).

On 2008, Linux Kernel Project was already using Linux. The lecturer recognizes how this release strategy benefits from this control version, apart from the other benefits such as distributed development.

Another important aspect on Linux Kernel Project is about testing. The speaker clarifies that testing is not easy for an Operating System. Unit Test concepts do not apply. Indeed, testing activities are delegated into developers mainly. What Kernel project performs are regressions, meaning installation of a complete kernel version and testing. An important way of helping the project is just taking an RC version and test it on your laptop, in order to discover possible bugs.

Apart from previous aspects, and after showing a more complete bunch of statistics around developers, contributions and so on and so forth, the speaker also shows the upcoming new features of kernel 2.6.26 version, which is a complete whole of new features (more than 50?), giving a measure on how active and dynamic kernel development was by that date.

Last, but not least, regarding the future of the Kernel Project, the lecture, already by 2008, clarified how important and “cool” was KVM for the project. Nowadays, in 2014, it seems that this importance has been achieved, taking into account how technologies such as Cloud Computing and Virtualization has turned.

7 Mercurial

7.1 Introduction

This Google Tech Talk [9], dated on June 2006, Brian O'Sullivan, who had been contributor to Mercurial for a while, explains the appearing of this VCS (Version Control System), who appeared in parallel to Git, in order to provide a solution to help on the fact of distributed software development. The talk is basically around two aspects:

1. **Mercurial Technical Details**. Information about Mercurial revision control system itself, and other stuff which are important from the comunity perspective.

2. **Distributed Version Control**. The lecturer also talks about particularities of distributed software development, and how important distributed repository control management is.

7.2 Mercurial

From lecturer's perspective, the most important thing regarding Mercurial and any other VCS is the ease of use. He asserts:

```
"I want to focus on the issues of programming,
and not the issues of the version control system"
```

In lecturer's opinion, there are different reasons to use Mercurial. These factors are important to analyze, as can be used as a rule of thumb of those aspects which are important for distributed development comunities, and in particular for Open Source comunities:

- **Straight forward to understand**. Mercurial has a conceptual model very easy to understand. Basically, handles three concepts:

  1. Repository. It is a weight-light repository, based only on a set of files. This help on the Speed when working with mercurial.
  2. Changelog. To register changes and group information around each change.
  3. Manifest. To register changes on each particular file for each change.
  4. File Metadata. Which controls additional information on each file.

The basic idea from the comunity perspective is how easy is to understand how the VCS handles changes, files and history on time, and the ease of use when working with it on a daily basis.

- **Easy to maintain**. Written basically in python (95%). There is a couple of modules also written in C. This means ease of maintaining, as Python is an easier computer programming language compared to others, and is increasing on popularity. It is, besides this, a complete VCS, "working reasonably well", in a code base of 12000 lines (by that date, 2006).
- **Fast**. Mercurial is a very fast VCS. On a pair of very simple performance test, it seems not very much quicker compared to other systems, on not very large repositories. However, from lecturer perspective, due to its design around abstraction layers, and its strategy around optimization, Mercurial boost its competitors in terms of speed for bigger repositories. Lecturer asserts:

  ```
  "We focus on not writing or reading more than strictly needed"
  ```

- **Ease of use**. Mercurial provide some functionalities which are straightforward compared to other VCS. Lecturer gives an example on the ease of use of applying patches, for example. Other operation which is extremely easy is the revision navigation, which helps on aspects such as determining revisions where bugs were introduced.
- **Distributed**. A very important aspect. Indeed, O'Sullivan considers that distributed VCS goes further than strictly Mercurial, and considers the concept as an important thing to handle separately. Lecturer's consideration about this topic are considered on the next chapter.

7.3 Distributed VCS

The second part of this Tech Talk is focused on the "Distributed" aspect of the VCS. Very related to the comunity aspect, from lecturer's perspective, **choosing the VCS has direct impact on the way that the project is going to evolve**.

On a large company, all the people normally have access to all the parts of the project. On an Open Source project, task force is much more splitted and repository access is much more distributed. **Normally, people work**

with other people which talks the same language, and focus work with the same tools. This aspect is much straight-through with a distributed VCS.

Apart from previous aspect, distributed VCS allow developers to be off-line, meaning having no connection to central repositories. This is not possible with VCS such as Subversion or VCS. Distributed VCS, meanwhile, allow to work on a local manner. As long as the history is kept locally, and you can access your hard disk, is enough. This **distributed way of working allows more flexibility on development process**.

Another important aspect in distributed VCS is the **ease of branching and merging**. Forking is the normal thing on distributed development, what helps on cooperation in terms of difference reconciling on the different branches which are created by the different developers. Apart from that, O'Sullivan recoginizes as well how Mercurial comunity is trying to emulate Subversion comunity example.

Apart from that, distributed nature of Mercurial also helps on fixing the main issues that centralized version means, i.e.:

- Having multiple repositories.x
- Avoiding bottlenecks.
- Flexibility to handle load management.
- Use without network connection.

7.4 Conclusion

Last, but not least, the lecturer's recognizes how important Subversion has been for Open Source. Indeed, O'Sullivan recoginizes the great contributions of this project and great comunity that has been built around this Open Source VCS, focusing on two aspects which he considers key in Open Source environment:

1. Technical Merit.
2. Credit.

O'Sullivan highlights how both aspects are accomplished by main Subversion contributors, and how important would be for Mercurial Open Source project to start receiving contributions from them taking this considerations into account. As conclusion, encourages the audience to use Mercurial due to the different reasons explained:

- Ease of use. Short learning curve, and similarities in certain aspects with previous VCS.
- Fast and efficient. Lecturer asserts:

  ```
  "Mercurial, in the end, is easy to use, easy to install, not too
  different from CVS or Subversion, very well received from people
  who has worked with it in terms of speed and eficiency
  and versy easy to scale."
  ```

- Ease of software maintainability. The amount of lines of code is not much (12000 by that date). The fact of the software to be written in Python is also remarkable in terms of code maintainability.
- Multiplatform. Being written in Python, allows usage of the code in different kind of Operating Systems (those where Python is available).

8 Stefano Zacchiroli on Debian: 20 Years and counting

8.1 Introduction

On Tech Talk [10], dated on March 2013 as part of the NYLUG (New York Linux User Group), Stefano Zacheroli, Debian Project Leader of Debian, talks about this exciting project, how the community of Debian has evolved over its 20 years of existance, the main aspects of the project and the future expectatives.

8.2 Free Software

To start with, Zacheroli raises a question to the Audience: "Why Free Software?". And his response to the question is simple, as he considers it is a matter of Freedom from the perspective of having the tools being used under control:

```
"It is a matter to have the control. The most control
you have over the software running on your electronic
devices, the more confortable you feel"
```

8.3 Debian

Debian is, by 2013, up to 17000 package sources with releases of 3000 new versions of software per month. This means a challenge in terms of distribution. Debian tries to be the glue between people who make the software, and people who use the software. Obviously, what Debian proposes, is to **enhance the ease of use of Free Software in order to grow the community of users of this kind of software**. That is the main mission of Debian, and it is the main reason of Debian to be considered the most important distribution from the community building around Open Source Software perspective.

By 1993, Ian Murdock announced the release of Debian as a new GNU/Linux distribution. The main characteristics of the distribution are:

- Open Source OS (Operating System) competitive with comemercial OS
- Easy to Install. No doubt, the key aspect for Debian to succeed. This is a key factor for community to start,

- Buit collaboratively. No doubt, the key aspect for Debian to succeed. This is a key factor for community to start,
- FSF supported distro for a while.

8.4 Debian stable - Squeeze

Main product of Debian, it is the stable version of its operating system. From the community perspective, the latest stable version, which was Squeeze by that time, had certain particularities:

- Binary Distribution. The distribution is provided as a set of binaries, in order to make quicker the installation process.
- Release every 24 months. Which is a particularity, as it escapes from the "release often, release early" aspect of Open Source. The main aspect for doing this is that Debian is "Quality Focused", and has been considered the most stable GNU/Linux distribution along the time.
- More than a dozen architectures. The most architecture it is supported means also the most target users as well.
- Pure Blends. Pure Blends are a subset of Debian that is configured to support a particular target group out-of-the-box. Medicine, Chemistry, GIS or Education are examples of Pure Blends.
- Popularity Statistics. It is also remarkable how spread is the use of Debian. 1 out of 10 Web Servers use it. Apart from that, it is considered the most popular GNU/Linux distribution.

8.5 Next Debian stable - Debian Wheezy

Zacheroli gives an explanation of what was, by March 2013, the next Debian stable to be released, called "Wheezy". The most important factors, in terms of community building, were next ones:

- MultiArch Focus. Focus on 3rd party easy cross-compilation.
- New Archs. E.g: armhf, s390x.
- Desktop. GNOME, KDE, XFCE recent versions support. Apart from that, what is more remarkable is that Debian has started to include built-in Cloud Computing support, now that this incipient technology is so popular:
- Private Cloud Support. Openstack and Xen/XCP packages.

- Public Cloud Support. EC2 and Azure packages.

8.6 Debian Project

The most important aspects covered by the lecturer on this Tech Talk regarding the community lays around the objectives of the project, as well as the guideline documents to follow from the community:

- **Mission**. The mission of the project is quite clear: **Create the Best Free Operating System**.
- **Debian Social Contract**. Basically consists of the main guideline to be followed by all the members of the projects and the Open Source Ecosystems as a whole. It consist basically on:
  1. 100 % Free Software. As the rule of thumb.
  2. Don't hide problems. As a way of Commitment.
  3. Give Back. As a way of Restitution.
  4. Priorities: Users and Free Software. As a way of living.
- **Debian Constitution**. It was written on the project constitution. It gives the project a "Nation Nature", with its own constitution, which defines the Structures and Rules of a Free-Software compatible democracy.
- **Community Members**. Lecturer clarifies how strong Debian Project community is. By 2013, more than 1000 project members world-wide are part of this exciting project. Developers are based basically in Europe and North America (as 99% of Open Source Projects), with increasing contributor number from South America.

8.7 Debian Community

Following aspect faced by lecturer in the talk is around Community of Debian. From his perspective, Debian Community is characterized by three main aspects:

- **Open Development**. Basing development and packaging on Open Source means easier way to discover and fix the problems. "Do not hide the problems", as a rule of thumb of the daily work, and "Show me the code" as the best way of showing transparency and look for solutions to problems arising.

- **Communication**. Mailing lists, IRC, social networks (Debian on identi.ca), Web Services and other tools are the channels for community members to cooperate with each other.
- **Large number of Tech-savvy users**.

8.8 Debian Distribution Particularities

Later, Zacheroli talks about a very important aspect, which is how Debian, nowadays, survives the so competitive market that GNU/Linux distributions mean. In 1993, no more options were available, but in 2013, twenty years later, more than 300 Distros, according to [11].

In this aspect, why is Debian important for the comunity and what is the reason for its popularity? The lecturer emphasizes on those aspects that make Debian to be unique compared to other distributions:

- **Package Quality**. Debian is based on the "Culture of Technical Excellence". Package maintainers are software experts, in the end. The project leader asserts:

  ```
  "We release when it is ready"
  ```

- **Attachment to Freedom**. Debian is committed to Software Freedom, not only about the software being distributed, but also in other aspects such as:
  - Firmware. Software released is free in 100%, and Firmware is too.
  - Infrastructure. Debian uses no non-free web services, as well as no non-free tools or infrastructure.

  Previous aspects drive on a **Community Awareness**, where users are conscious of Debian not to betray Free Software principles, and meaning a **high bar for Free Software advocates**.

- **Indepence**. Debian is proud to be a company non-dependant project. This is remaarkable nowadays, where companies are so involved in Open Source Project environment. Debian survives on donations and gift-economy. This helps on Debian credibility around decission making, as Debian decissions rely on a "non-profit" strategy.

- **Decission Making**. Decission making on the project is based on two main aspects:

  1. **Do-ocracy**
  2. **Democracy**

  Both aspects mean that the comunity reputation is based on the work performed, there are no benevolent dictators, and above all, **Decissions are not imposed**.

It is important to consider this set of aspects in order to understand how this project has acquired a so well considered reputation, and how Open Source Project should chose a sameless politics if they want to reach to have a commited and happy community.

8.9 Debian Derivatives

Zacheroli faces the issue of other Distribution which are Debian based. The most popular one is Ubuntu, which in some aspects is much more popular than Debian. However, it is remarkable how the lecturer considers derivative works a normal thing on Open Source project, as it is considered in the fourth freedon around Free Software, related to derivative works.

Debian Project leader emphasizes how rather than being a danger for the project, Ubuntu has supposed sinergies for the Debian Project as well, as many patches from this distribution have been contributed back as well.

8.10 Contribution to Debian

Finally, Zacheroly takes an opportunity to aim the audience to contribute to the project. The most remarkable aspect here is how Project Leaders must also perform the role of Comunity Managers, helping on non technical aspects such as donation asking can be. Contributions to Debian can be done in different ways:

- Hardware Donations.
- Hardware-Related Services. Hosting, etc.
- Money. Above all, for sponsor development meetings.

Apart from that, there are other ways to contribute, having not to do with the monetary thing, but rather with aspect such as:

- Downloading, Using, Testing Debian, as well as reporting Debian bugs.
- Adopt Debian orphaned packages.
- Join packages team. There are different teams, organized around programming languages, fields of use, etc.
- Hack on Debian infrastructure.
- Work on non-development stuff. Works as translations, design (websites, themes), communication, marketing, documentation, accounting or events organising are key aspects for project to succeed.

As the last remarkable aspect about joining Debian community, Zacheroli encourages the audience to reflect on the kind of commitment and knowledge that they can provide, and acquire the role appropriated according to this.

9 EdgeBSD

9.1 Introduction

Dated on February 2014 as part of FOSDEM 2014 [12], Pierre Pronchery, a developer on EdgeBSD, shares his perspective about this particular project. On the Tech Talk [13], the lecturer clarifies what is the project, how it is organized. In the end, EdgeBSD is a BSD distribution, based on NetBSD, where some alternatives development branches have been started.

From the comunity perspective, this lecture faces very interesting issues, e.g.:

- NetBSD development model
- Git and Distributed development
- EdgeBSD development model
- Release system and improvements

9.2 NetBSD

To start with, the lecturer talks about NetBSD, a project started in 1993, which is considered one of the oldest "modern" open source project. It gave birth to OpenBSD, has about 256 developers today, and has a cathedral-like type of development. It is remarkable how this developer refers to the Cathedral model, referring to the Cathedral and Bazaar [14].

From lecturer's perspective, this OOSS (Operating System) is "great and beautiful" due to a set of characteristics it accomplishes:

1. Has a strong focus on quality.
2. It was very well designed.
3. It is cross-compiled for your architecture.
4. The project attracted lots of cool researches.
5. Contains much of the new features, such as Xen, ASLR, File Cryptography, ZFS support, etc.

9.3 NetBSD development model

From the comunity perspective, one of the most important topics has to do with the development model in NetBSD project, which helps on clarifying why the project has above characteristics. The lecturer gives some keys in order to let the audience discover why this comunity can be considered a "Cathedral like" community:

- Only official developers can commit changes. And being an official developer is not easy. You must belong to the NetBSD Foundation, apply for the role, and then it takes a while until you are granted with the permissions.
- Commits need usually to be reviewed.
- As it is based on CVS, branches remain forever.
- Existing features are not allowed to be broken.

In lecturer's opinion, **this development model can result harmful**, due to several reasons:

- Occasional contributions can not commit their patches. They can not easily become software developers inside the project.
- Someone has to be available to commit.
- If contributions are major contributions the issue is even worst. The contributions are more difficult to be reviewed.
- Reviewers, meanwhile, can not easily test and patch others contribution. This fact is even worst due to the fact of NetBSD project being using CVS.
- Even being an official committer, job is not easy. Using CVS means that you can't perform tasks such as reverting changes, break features, delete branches, etc. If you do so, people will receive mail (considered basically spam) and would go against your developer reputation.

Apart from previous issues, NetBSD project have other ones having not to do with the community, but rather with technicals considerations, such as NetBSD to be using its own versions of very important applications such as GCC, Xorg, bind, postfix, etc. This kind of aspects is making NetBSD not to be executable in the latest times, due to, for example, NetBSD Xorg official packages being broken.

9.4 CVS: The issue

From lecturer's perspective, CVS is not so bad. He prefers it rather than Git, as considers CVS much easier to fix, while Git is considered inconsistent, opaque and difficult to use. However, lectuerer considers that this VCS (Version Control System) has some advantages that makes it very suitable to ease distributed development. Features such as:

- Branch creation. Branch can be created basically for free.
- Offline work. Git local repository can be used off-line, and branches and commits can be performed without needing to push to official repositories.
- Stash and Stage.

9.5 EdgeBSD

EdgeBSD is, as the lecturer's clarfies, an staging area for NetBSD. Apart from that, EdgeBSD is using decentralized VCS (Git for its popularity).

Without considering the technical considerations, which are not relevant for the lecturer, EdgeBSD project results being more open, more affordable and of course more fun. Apart from that, it allows more people to contribute, by working in different brances, and provides an ecosystem in order to ease the contribution of polish patches to BSD.

All the previous aspects allow also to attract more research and contributions than NetBSD.

9.6 EdgeBSD development model

EdgeBSD contains a Git mirror (CVS to Git) in order to track NetBSD's original code. Apart from that, EdgeBSD contains some Git repositories in order to handle track master as well as select branches. Contributors can push to that branches.

EdgeBSD master branch and NetBSD branch are considered as the tentative branches.

9.7 Push Open Source Development

Another different aspect of NetBSD project compared to NetBSD project has to do with the intention of the project to facilitate Open Source development.

For this reason, the project approaches to factilitate Open Source Software (OSS) development due several aspects such as:

- Provide a good development infrastructure to developers. For this reason, the project provides e-mail, calendar, secure Instant Messaging (IM), VoIP and conferencing facilities, among other stuff.
- Automated check and procedures. "Build-o-matic" features, such as "compile this patch for this architectures", or "push this to master patch if it compiles ok".

9.8 Edge BSD release

EdgeBSD provides a release system, that consists of the main NetBSD branch with some extra functionalities enabled, continuous security updates and bug fixes and stable and testing new packages. From lecturer's perspective, releases have to provide some key features in order to the distribution of the OS to be successful. Among the desired features for EdgeBSD releases, next ones should be there:

- Graphical Text Based installers.
- Default environment, with known hardware and software support.
- Ready to flash images for a range of devices.
- Milestones, continues updates.

This is a remarkable aspect from Community building perspective, in order to make the product to reach to a wider target user community.

9.9 Conclussion

Last, but not least, the lecturer provided some "hands-on" demonstration in order to explain how Git repositories are organized and how Gitolite helps on managing users. Apart from that, some URLs are provided in order to promote the project [15] This lecture is very interesting in order to demonstrate how a Fork can appear due to several reasons, and one of them is, as in EdgeBSD project, that **some people who can not contribute to a project due to its development process strategy**. This issue can result on Fork appearing in order to accommodate all those developers who are ocassional contributor but still want to include their contributions to the community.

10 Debian Secrets what I wish I knew before joining Debian

10.1 Introduction

On Tech Talk, dated on February 2013 as part of FOSDEM 2013 [12], Lucas Nussbaum, who is the current Debian Project Leader (DPL), gives a presentation on those aspects that we would like to have discovered before joining Debian project.

From the comunity perspective, this lecture faces very interesting issues, e.g.:

- Decission Making in Debian Project.
- **Debian Patterns**.

Debian determines that decision making is based in two fundamental aspects:

1. Do-ocracy. In this project, as in 99% of the Open Source Projects driven by comunities, **reputation follows work**, and never the other way round.
2. Democracy. There is no benevolent dictator, as in other Open Source Projects. Decission making is not imposed.

But how does this work in practice? What makes Debian a so unique, "but sometimes frustrating", project? Debian follows some patterns in order to accomplish the project decission making strategy. In this talk, Debian DPL provides some key aspects in order to clarify why, despite being a project considered to be one of the most democratic ones, "Debian is not a rant", and not everybody is continuosly flooding with their opinions on how to drive the project, make the decissions or follow a particular strategy.

10.2 Debian is magic

DPL remarks that the **main value** existing on Debian has to do with its **Task Force**. Debian Project is marked because of its great community, considering that:

- Great experts on computing science, working on a volunteer basis.
- Project is under continuous improvement, and making these improvements free for everybody.

- Culture of technical excellence. Debian is driven by a “Quality First” strategy, where lots of thing can be learnt just by following mail lists.
- No boss. A figure of a DPL exists, but just to give a vision, meaning no obligation to follow it. There is also a Technical Committee, but it is only a last resort option.
- Special Organization. Debian is based on a majority of people who are not paid by any company. This gives a strong power in terms of independency and community driven decission making.
- Lots of subgroups and subprojects. “This involves an interesting mix of problems and challenges” from lecturer’s perspectives.

The technical talk is mostly focused in last two points, in order to clarify how Debian Project is organized, and the different issues that the community faces along the time.

10.3 Do-ocracy

What is do-ocracy. It is a very simple concept for the lecturer: **“Those who do, decide”**. However, those who do, decide on both their work, but also about the other’s work. This aspect can drive to several issues:

- Usual problem: A wants B to do something. B disagrees on doing so.
- Solution: **Technical Committee** (TC). This committee is the only one who can overrule this kind of situations. The decission takes long, usually months, rather than weeks. Although, normally, TC does not overrule the maintainer (B in this case). TC is compounded by eight people, normally proposed by the DPL. They are not normally needed. 2010 and 2011 needed no TC action. On 2012 just a vote about multiarch packaging tool, dpkg, was needed.

10.4 Patterns

10.4.1 Benevolent Dictators, but still dictators

There is a typical scenario where conflict appears. It has to do with maintainers who mostly work correctly. He usually works alone, without belonging to a team. Normally, he/she insists on ack’ing all decissions. There is normally a source of disagreement with other Debian Developers (DDs). They are normally related to:

1. Design Decissions.

2. Things not done on time, quickly enough.

However, this kind of situation has to do with **people complaining, but normally not offering help**. Normally, their proposes is even worst.

In this kind of situation, the maintainer must drive the situation by making everything public, using "BTS" or "civil mail list", and being patient. Apart from that, the conflict must be handled with **focus on technical issues, offering solutions such as patches or any other type of help, and avoiding personal offense**.

There are also some procedures to follow in other situations. For example, when opening a Bug related to packaging, and starting a conversation around the bug, it is important to CC about that particular bug for people to understand, quickly, what the mail is referred to.

Conflicts are not that common. Normally, package maintainers make good decissions, and none of previous situations must be faced.

10.4.2 Going Further: Core teams

Debian project has a set of "core teams". This kind of teams take care of project key areas, such as archive (ftpmasters), testing (release teams), machines (DSA), security, and so on and so forth. These teams are characterized by some aspects:

- They make very good decissions most of the times.
- However, when conflicts appear, it results on **big flamewars**. The recent example is squeeze release. Release date was announced months after announcement, what resulted on hurting the project.
- Core teams are powerful. This result on "partitions of powers" inside the project. They are not easily convincible when critical changes are decided.

10.4.3 Please, send a patch

As a do-ocracy driven community, there is a typical situation appearing sometimes:

- A wants B to do something.
- B responds: "Please, send me a patch".

This demand is normally very reasonable, but other times it is not so. A very basic situation, for example, is when B made a typo in a man. There is no point for B not to fix that. In the end, **optimizing the taskforce must prevail**. If B has already performed the clone, just needs to fix and submit. Maybe, A, has to clone, be providen with access, etc.

10.4.4 Nobody feels empowered to do the work

Normally, a common issue is that people working on a certain team do not know their role inside the team, as most teams don't have a strict hierarchy.

```
"Normally, people are Team Leaders, but do not know it.
This means normally to be blocking decissions, by just not
responding to an e-mail."
```

There are two solutions in order to avoid this kind of situation:

1. Give always feedback to those who ask for it.. The sooner, the better.
2. Just do it. Normally, it can be reversed anyway.

10.4.5 Nobody wants to do the grunt work

There is some kind of work that, despite of being grunt, must also be done. However, Debian community member is normally a computer geek, that does not want to do this other kind of projects. Debian is willing to receive other kind of geeks, such as publication communication geeks, accounting geeks, documentation geeks, etc. However, Debian is making progress in this area.

10.4.6 Over-optimized process

DD's are normally selfish. They normally design the most appropriate procedures for themselves, which are not necessarily the best for the users or the rest of the community.

Bug Tracking System (BTS) is dedicated to track the process and decide on actions such as orphaned packages, packages that must be removed, sponshorsip requests, and so on, and so forth. However, patches have being stopped to be tracked with BTS.

10.4.7 Button-up design and adoption

Debian Project follows the characteristic of "Bottom-up" design, and in the end, decission making. This is normally common to all the Open Source projects, and in Debian, due to the independency that rules the community, even more. In the end, the project follows some common characteristics:

- It is decentralized.
- Most innovations and contributions start on a "scratch-an-itch mode".
- Process is long. Sometimes, innovations last months, even years, to expand to other areas and the rest of the project as well. The typical example is Debian services,

From lecturer's perspective, this bottom-up way of working, focused on decentralized groups with no hierarchy and strong autonomy, favours the innovation.

10.4.8 Packaging practices inside teams

There are a lot of teams who are involved in packaging, but in the end they have not many different needs. Their needs are normally related to optimizing packaging, migrating from SVN to Git to favour decentralized development, manage patches, etc.

However, it seems that, due to the fact that the teams are decentralized and autonomous, there are as many workflows as teams. Apart from that, workflows are not normally documented. This does not ease the fact of people moving between teams, as working in different teams mean learning different workflows.

There is a need inside the community to fix this. **Development workflows must be designed in order to decrease entry barriers and favour documentation**.

10.5 Conclussions

Last, but not least, conclussions of the talk are provided by the lecturer:

- **Several patterns.** Already been described in previous section, those patterns must be taken into consideration to understand Debian way of working.

- **Debian particularities.** Being Debian so decentralized and independent makes the project to be interesting and have patterns not possible on other more vertical organizations.
- **Cultural excellence, technology geeks, restitution.** Why not joining?

11 Community, the most important asset

After previous Tech Talks, some conclussions can be extracted in order to summarize the main aspects around Open Source project communities. The most important one, as a personal opinion, is the **Community as the most important asset of Open Source**. Open Source projects are built on top of their Communities, independently on the nature of it. There are some projects with companies involved, where most of the contributions are from software developers by hired companies, other projects are only built on volunteer contributors, e.g.: Debian, although the normal situation is to have a mixture of both of them.

Despite of the nature of the Community, the important issue for an Open Source project community is to create a nice environment around the project, as well as a smart ecosystem that allows the project to succeed, both from the User Community side, what involves the number of people using the product, but also from the User Contribution side, involving the number of people contributing to the product, be it through occasional contribution, be it through dedicated collaboration over time.

What are the main aspects to take into account for having a healthy community environment to helps an Open Source Project succeed?. Previous describe talks have proposed some of them, that are summarized below:

- **Community Building, Promotion and Advertisement**. One of the most important thing regarding Open Source projects has to do with how Community is built, promoted, managed, and, in the end, how the Open Source Project is advertised to the audience.

  In the end, each of the talks above, apart from the aspects shared to the audience, include references to join that particular Community, or at least, help on building the particular project either with donations (monetary, hardware, services, etc.) or with task force involvement, not just with software contributions, but also with translations, help on documentation, contributions to legal issues, market share and advertising involvment, and so on and so forth.

  Other very useful help for a particular Community is to use their product, be it an Operating System, be it a CMS, be it a Web browser, etc. This is, in the end, the first step in order to be involved in a

Community.

- **Quality**. Once a user gets to know a product, be it an OS, be it a Web Browser, be it a CMS, etc., the important thing is to **have product with good quality** in order to retain the user and make him or her to use it from that particular time. Regarding this, some projects, such as Debian, as described in section 10, are focused on **Technical Excellence**. In order to reach Quality, Community Managers, Technical Leaders of the project and, as far as possible, main Community contributors, have to be aligned in order to get attractive and robust software.

- **Community Management**. Community management is, in the end, the other important aspect regarding the Open Source Projects and their Communities. Which are the main aspects to be followed by community managers and project leaders in order to retain both users and collaborators in the project? There are some rules provided in the talks that can be followed as a "rule of thumb". Once a person arrives to a community, there has to be a nice environment created in order to retain that person on that particular community. The main aspects regarding healthy communities have to do with aspects already described. Among them, the most remarkable are next ones:

  - **Decission Making: Do-ocracy**. For an Open Source project to be successful, it is essential to accomplish a requirement: **Do-ocracy**. In this way, the community perceive that those who contribute in more often are also those who have the word in order to make decissions. Debian project is a reference in this aspect, as described in both section 8 and 10.
  - **Avoiding poisonous people**. Specially interesting from community management perspective is the talk described in section 3, and the different procedures to follow in order to protect the communities against disruptive people and, in the end, **get the community to be focused on the important issues of the project**.
  - **People Management**. Behind an Open Source Project grouped around a Community there is **People in the end**. Related to this, Community Management team has to be careful in order to take measures that allow fortifying the community, such as described in section 3. Apart from that, it is also recommended

to accomplish some requirements in order to favour people managment, with actions such as those ones described in section 4, related to publishing guide principles, give recognition, favour work delegation, include mentorship programs or preparing an smart design/implementation process.

- **Development and release process**. It is important, from the community perspective, to have an agile method to develop and release software. Specially relevant according to this aspect is the information provided by Greg Kroah-Hartman regarding Linus Kernel development and release process, as described in 6. Among the main aspects to be remarked on that particular talk, there are two. On the one hand, "The trusted relationship network" that applies to a big community such as Kernel Linux one. On the other hand, how can a model based on a "release early, release often" strategy favour the success of a project and its community. It is also important how having a closed structure in development process, such as happens on NetBSD, can lead to forks of the project, e.g: EdgeBSD. Section 9 contains a description on how a fork tries to solve the issue of people trying to occasionally willing to contribute a project and not being able to do so, as happened on NetBSD.
- **Openness**. Making decissions without considering Community concerns can be very dangerous. An example is Netscape/Mozilla, who had lost a lot of street credit due to some decissions, such as discontinuating projects where comunity was involved. Open Source Projects must be "Open to the point of promiscuity", as Eric S. Raymond stated in the "Catheedral and the Bazaar". This aspect was covered in the talk of Mozilla and Camino, described in section 5.
- **Distributed Community**. Normally, Open Source Projects have a "Distributed Nature". It is important to take into consideration this aspect, and shape development tools and process according to this distributed nature, as described in section 7.
- **Conflict resolution**. This aspect is another issue to keep an eye on. Although conflicts are not that common in healthy Communities, as described in section 2, sometimes they appear, and then there have to be mechanisms to face them. For most of the projects, in particular those where Benevolent Dictator role exist, this role will have to take the decission. Meanwhile, other

projects, such as described for Debian in section 10, a Technical Committee exist in order to resolve this kind of situation.

  – **Patterns and anti-patterns**. Last, but not least, it is also important to **consider the most important patterns and anti-patterns in Open Source**. In section section 10, some of them were highlighted. Additional anti-patterns can be checked on [16].

As a general conclussion, it must be remarked how previous aspects have to be taken into account from Community Management perspective in order to achieve the success of an Open Source Project. The main idea to keep in mind: **It is not only software, above all, there are people behind Communities**.

References

[1] Rober Watson. How the FreeBSD Project Works, 2007.

[2] Poul-Henning Kamp. A bike shed (any colour will do) on greener grass.., 1999.

[3] Ben Collin-Sussman and Brian W Fitzpatrick. How Open Source Survive Poisonous People, 2007.

[4] Zigurd Magnusson. SilverStripe CMS, 2007.

[5] Waterstone. OpenSource CMS Market Share Report, 2011.

[6] Karl Fogel. Producing Open Source Software, 2013.

[7] Mike Pinkerton. Camino Web BrowerSilverStripe CMS, 2007.

[8] Linux Kernel Maintainer Greg Kroah Hartman. Technical Talk on Linux Kernel Project, 2008.

[9] Brian O'Sullivan. Mercurial Project, 2006.

[10] Stefano Zacchiroli. Debian: 20 Years and counting, 2013.

[11] DistroWatch. Distro Watch Search Distribution, 2013.

[12] FOSSDEM. FOSSDEM 2014, 2014.

[13] Pierre Pronchery. [FOSDEM 2014] The EdgeBSD Project, 2014.

[14] Eric S. Raymond. The Cathedral and the Bazaar, 1997.

[15] EdgeBSD. EdgeBSD, 2014.

[16] Community Management Wiki. Community Management Anti-patterns, 2014.