

Project Management

Final assignment
Release of a FLOSS product by a SME

Ricardo García Fernández

March 4, 2013

©2013 Ricardo García Fernández - ricardogarfe [at] gmail [dot] com.

This work is licensed under a Creative Commons 3.0 Unported License. To view a copy
of this license visit:

<http://creativecommons.org/licenses/by/3.0/legalcode>.



Contents

1	SME Introduction	3
2	Publish the project	3
2.1	Dual License	3
3	Market niche - Competitors analysis	5
3.1	Code climate	5
3.2	Sonar	6
4	Management general path	7
5	Management Policies	8
5.1	Where will be published the code ?	8
5.2	Communication strategy and channels	9
5.2.1	Mailing Lists	10
5.2.2	Forums	11
5.3	Managing volunteers and attracting new users	12
6	Technology	12
6.1	Technical Infrastructure needed	12
7	Business scalability	12
7.1	Evolution	12
7.2	Emphasis	12

1 SME Introduction

Code Garden is our SME. We develop software products using quality patterns. Quality patterns are our main goal, applying Quality patterns to software development as is. plants and the code can not be kept alone, they can grow with life, diverted, or wither in a forgotten place. So plants like the code, need extra care, some gardeners, so they can grow and flourish.

Technical Debt is not a monster chasing us in every development sprint is another *ROL* that we accept and we have interaction with it. We need him and he needs us.

Thus, we created a software product that helps us to deal with Technical Debt, **Greenhouse**.

Greenhouse is our tool to track the progress of the evolution of code quality within a controlled environment. Using quality metrics for each programming language helps us reduce technical debt faster

"we are the code we write"

2 Publish the project

We want to publish *Greenhouse* as a FLOSS¹ project with two Software Licenses.

One a FLOSS License and the other a Private License. We chose a *Dual-License* product.

Thus we can serve every type of costumers as MySQL business model explained by Elena Blanco[6].

For anyone who wants to develop and distribute but does not want to release the source code for their application, MySQL is able to provide a commercial licence. Because MySQL has full ownership of the MySQL code, it is able to tailor its commercial licensing terms to meet the unique requirements of users interested in embedding or bundling MySQL.

2.1 Dual License

Free Software License and Private, Brief discussion about licenses (your company has heard about some BSD or GPL, but they are not sure!).

FLOSS License selected to publish *Greenhouse* is *GPLv3*². This FLOSS License provides all freedoms to the project:

¹Free Libre Open Source Software

²GNU Public License Version 3 - <http://www.gnu.org/licenses/gpl-3.0.html>

- the freedom to use the software for any purpose,
- the freedom to change the software to suit your needs,
- the freedom to share the software with your friends and neighbors, and
- the freedom to share the changes you make.

The other License is a private License. A private License gives us more flexibility through market niche.

Good thing to take in advance using a Private License are:

- Avoid possible or unexpected License Violations.
- Build a project in your company and be 100% sure that you are not violation any License under your product.
- This License envelops the whole product into a private version.

This private License gives you the opportunity to deal with *Greenhouse* libraries and include into other projects using full capabilities. This way you can avoid License problems with derivated works and be compatible with other FLOSS Licenses.

Brian Behlendorf[7] explains this model with a success and its weaknesses with the community development:

You have to be very careful, though, to make sure that any code volunteered to you by third parties is explicitly available for this non-free branch; you ensure this by either declaring that only you (or people employed by you) will write code for this project, or that (in addition) you'll get explicit clearance from each contributor to take whatever they contribute into a non-free version.

We are very concerned about how to interact with the community. Our goal is provide to community the opportunity to develop solutions to our clients for *Greenhouse*.

I would claim that if you treat your contributors right, perhaps even offer them money or other compensation for their contributions (it is, after all, helping your commercial bottom line), this model could work.

Last quotation is how *Transvirtual*³ in Berkeley applied this model to a commercial lightweight Java. Encourage the developers to work and earn money directly with the software they make. We believe in this model to create a community involved around the product:

- The usability of the product.
- Tangible incentives as our case the money.

³<http://www.transvirtualsystems.com/>

In this way we will try to solve the dilemma of FLOSS developments that can be converted into proprietary solutions.

Therefore, an exchange of knowledge by salaries to the community through an assignment of copyright to Code Garden by using the *GPLv3 License*, they maintain the authoring and earn tangible rewards too.

We are the copyright holders of *Greenhouse* and thus , we want to share the maintainability with community knowledge to provide solutions for companies with private Licenses of the product.

We want to evolve with the community and spread our developments and remain FLOSS.

3 Market niche - Competitors analysis

Code analysis is increasing everyday in software development. SME & Big Enterprises focus its products near quality. Why ? Because Software Development is measurable, high measurable I could say. Every software is developed guided by patterns through developers and the final product (talking about clients) is released to the client showing its functionality but what happens with all the code developed inside ?

The code evolves as the development grows. In a development team is difficult to measure the quality of the product grained.

Other sample of the use of measure the quality is when you have to choose between two libraries to implement another service that use the functionality implemented by them. One metric to take care is the code quality that you can apply for them. Using some objective metrics you can retrieve a numeric result that gives you a general idea. Or if you want to add an existing module/library you can choose the one which its result is near to your software, not lesser not higher. It is a way of seeking for a balance in development and knowledge about complexity of a module to import to your product.

There are some samples of Analyzing tools but we are to analyze *Code Climate* and *Sonar*.

3.1 Code climate

Code climate - <https://codeclimate.com/>. Code quality analysis in Ruby Language. But we will not focus on language but with what gives us the tool1.

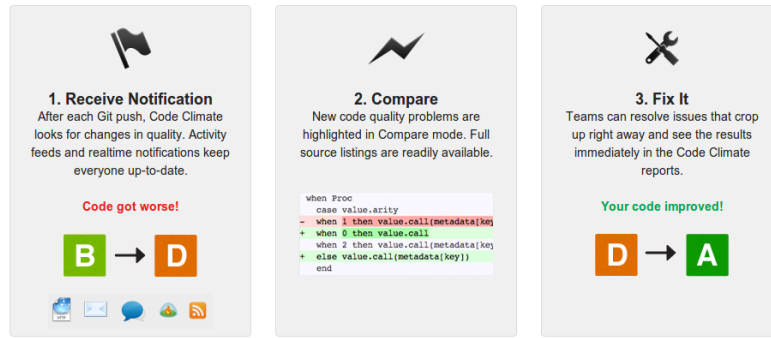


Figure 1: Code Climate Notifications

This tool gives us interoperability with Software Forges like Github integration. Source Code Management with Git repositories, Code Quality evolution, Notifications, Comparison tools. This tool set is simple but efficient, this is very important. Play with the ease of integration into a forge development offering hosting server, pay and enjoy the product (one click purchase).

Includes the main features of services; *PaaS*⁴, *IaaS*⁵ and *SaaS*⁶. Using *bluebox services* - <https://bluebox.net/>: Virtualized Environments on Actual Hardware.

We want those services available for every developer and easily result visualization.

It's free (gratis) to use for FLOSS projects.

3.2 Sonar

Sonar - <http://www.sonarsource.org/>. Sonar slogan is very clear:

Put your technical debt under control

Despite Code Climate, Sonar covers lots of languages and is a FLOSS product. A very good point to consider for the use of this software. Controls and test every corner of your project².

⁴Platform as a Service

⁵Infrastructure as a Service

⁶Software as a Service

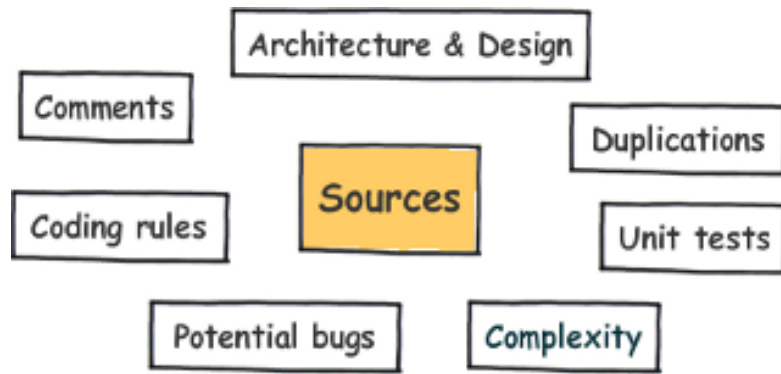


Figure 2: Sonar covers 7 axes

This Software gives you the opportunity to integrate with your Source code management and a result metrics result visualization panel with Nemo3 - <http://nemo.sonarsource.org/> using SaaS solution through *Cloud Bees* - <http://www.cloudbees.com/platform-service-sonarsource.cb>.

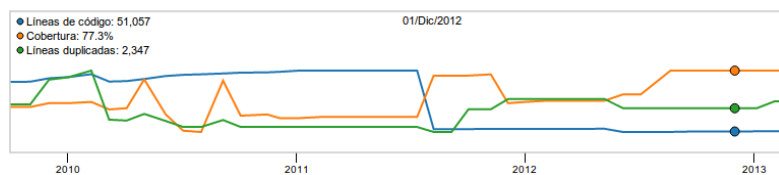


Figure 3: Nemo evolution graph

Sonar is focused in code Analysis its market niche as Code Climate but offering easy configurable solutions for integrate your projects with giving you the possibility to analyze our software and visualize the results among time.

This is our niche, Software Analysis, visualization and interoperability between Source Code Management.

4 Management general path

services and personal. Social and political organization of the project/community. Communities, Enterprise ROLE, Single Vendor or Apache Software Foundation.

5 Management Policies

5.1 Where will be published the code ?

First is not where, first is, which SVC - *System Version Control* - will be used ?

We choose a DVC - *Distributed Version Control* - System instead of an CVC *Centralized Version Control* System. A *DVC* increases the capabilities to developers (including community contributors) to develop solutions and test their development in every environment. The main reason is that a *DVC* gives you the opportunity to work without being connected to internet and save all your historical revisions. The other reason is that applying *DVC* Systems (mainstream) workflow Branch per Feature from Martin Fowler analysis[8] and how to fit with Continuous Integration Development, increases the productivity and facilitates the integration of new developments minimizing the problems with merging in *CVCs*. Thinking about publishing a project as FLOSS with the idea of creating a solid and integrate community we must use a *DVC*.

There are different *DVCs* to choose:

- Monotone - <http://www.monotone.ca/>. monotone is a free distributed version control system. It provides a simple, single-file transactional version store, with fully disconnected operation and an efficient peer-to-peer synchronization protocol. *Introduced hash commits.*
- GNU Arch - <http://www.gnu.org/software/gnu-arch/>. It is used to keep track of the changes made to a source tree and to help programmers combine and otherwise manipulate changes made by multiple people or at different times.
- Bazaar - <http://bazaar.canonical.com/en/>. Bazaar is a version control system that helps you track project history over time and to collaborate easily with others. *GNU Arch fork.*
- Darcs - <http://darcs.net/>. Darcs is a free and open source, cross-platform version control system, like git, mercurial or subversion but with a very different approach. Thanks to its focus on changes rather than snapshots, Darcs can offer a freer way of working, and a simpler user interface. It's written in *haskell*.
- Git - <http://git-scm.com/>. Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
- Mercurial - <http://mercurial.selenic.com/>. Mercurial is a free, distributed source control management tool. It efficiently handles projects of any size and offers an easy and intuitive.

After a deep analysis for choose one *DVCs*. We decided to use Git. Our decision is based in the facility to propagate through internet, most development forges allow and promote the use of Git instead of others, this could be a hype but a hype that works

because if this system has not a good *interoperability* APIs and easy hooks configurations will be replaced easily. And was born from Linus Torvalds after BitKeeper affair with taxes and its 'special' license clause⁷ to manage Linux kernel source.

After choosing Git we must decide where. We decided to host our Git Repository in Github - <https://github.com/> - Projects and community in Github have a lot of visibility and uses a clean user-friendly interface furthermore is a Git powered forge that allows an easy implementation of hooks to our repositories. Why is this important to us ? Because our project is a code analyzer, where could be better hosted to be tested than in an easily interoperability configurable forge that spread hook integration? I think this forge increases the visualization to our work, organization capabilities and community communication, the most important issue.

GitHub has an easy patch/merging/pull-request tool to merge work from contributors, using code visualization, comments in commits and patch visualization. It's very easy to apply a contribution. Furthermore gives to the project set of quick analysis graphs from contributors commit activities⁴ and branch history, these analysis are a key feature to gain visibility for the community.

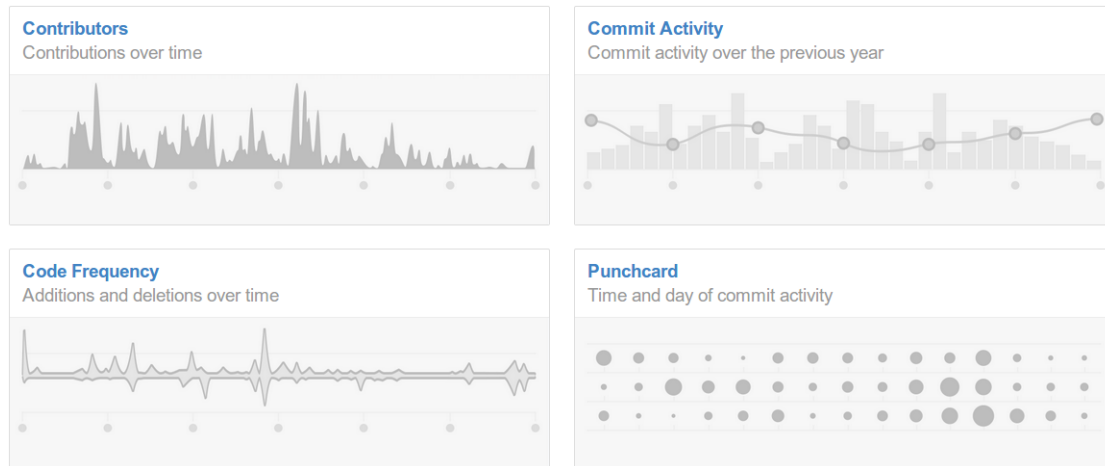


Figure 4: GitHub contributors graphs

5.2 Communication strategy and channels

In communication staging we have to accomplish with two main groups: *strategy* and *channels*.

Starting with *Jono Bacon*[9] introduction to *Communicating Clearly* chapter:

"Community is absolutely about understanding the ether. Our notes are the processes, governance, tools, and methods in which we work together."

⁷<http://kerneltrap.org/node/444>

The notes we dont play are the subtle nuances in how we pull these notes together and share them with one another. The space between the notes is communication.”

Stands as the centrepiece of his chapter, the most important thing in communication within a community are *“the notes you do not play”*, citing a quote from Eric Clapton. That is, when we know which is the vehicle we need to introduce for proper communication. It is more important that we miss something than overcharge the channels

Therefore, we will divide the process into two phases. Using the analogy of the roads and driving through them we have:

- Create the highways - Channels: such as mailing lists, forums, and discussion channels.
- Encourage great driving - Strategy: providing a consistent example of simple approaches to communication that make the community easier to understand and more pleasurable for everyone involved)

Thinking about our project, *Greenhouse* is a technical project for technical users. This is the first step to choose an appropriate channel. We have developers in our company and first users (that could became developers) thus we want to separate the communication between two channels:

- Mailing lists: Create different mailing lists for each type of subscriber: developer, user.
- Forums: Forum to get an easy human enter door to the project for a newbie developer and not expecting to be the main communication channel.

5.2.1 Mailing Lists

Be clear, specific and human (the most important). In Mailing Lists we have to be in touch with the people interested to contribute to the project, not only the short term upstream, we are focused in long term upstream. Thus we have to be organized, focused and good mates. *Mailing Lists* will be public and this has to be in the whole project, because of transparency we can create a trustful community around the project.

Starting with mailing list, we need a clear language, follow Netiquette⁸ basic guidelines and respect everyones opinion.

Avoiding the main problems in Mailing List such *bikeshed*⁹ and over technique discussions which can drive away new community members.

⁸<http://tools.ietf.org/html/rfc1855>

⁹The gist is that while nobody will question the details of a large and complex project (e.g. a nuclear reactor), for a simple thing as a bikeshed everybody will loudly add their two cents as to exactly how it should be built/painted/constructed

Mailing Lists have to be maintained and guided by the community thus we have to be clear and describe process, guides and documentation.

5.2.2 Forums

The *Forums* create a friendly user interface for a newbie users to get in touch with community and little problems that couldn't fit well in our *Mailing Lists*. Gives the opportunity to search and reply easily. To the new user or the user that only is looking for one determinate question, such as FAQs¹⁰ or version release posts.

Forums maintenance could be assumed by a developer (in the beginning) and other forum users that increase its participation and rank by meritocracy inside the *Forum*. The purpose of the forum is to be handled by community to introduce a friendly-human face to newbie users.

After chose these "*highways*" and "*Encourage great driving*" we need documentation, clear documentation related to this tools. How to get started and how to choose the right "*highway*". This will guide the user (pre-community user) into the channels and facilities of community.

- Where are the tools ?
- How to start. Firs search, is not necessary to be register to search for information. Transparency and increasing engagement with the community interacting.
- Guides and rules to write, showing samples. Friendly and human samples. For the administrator and the basic user.
- FAQs - Resolved issues, HowTo's, help and where is.

Furthermore we have to use blogs, create and produce content related to the community, personal blogs, technical blogs, articles, stories about the community, talks and success cases. This could be another highway to spread information about the community and the project.

If you are looking for IRC channel, you won't find it. We started this section referring us to the appointment of Jono Bacon "*the notes you do not play*" therefore we decided not to create an IRC Channel for the community because we can't manage it whit our resources. Ff the community will want one we will be happy and could try to afford this cost in the future.

I want to include a reference to Issue Tracker System (ITS) in this section as a *highway* and *encouraged driving* element. The *ITS* is important in a community, in our case a technical community not focused in an end-user. We include an *ITS* to manage the project in the community, communications, RFC - *Request For Comments*, Doodles,

¹⁰Frequent Ask Questions

discussions, documentation, references and of course *Netiquette* related to how to use an *ITS* which is explained widely in Technology Section6.

This tool is the jump for the user after the *Forums* and *Mailing Lists* to contribute to the development aimed by the members of the community. Following Netiquette guidelines to post a bug into the *ITS* and stay tuned with issue updates, test the patches and help developers to fix any bug.

Because of is another communication channel with development team I chose to reference the *ITS* inside this section.

5.3 Managing volunteers and attracting new users

6 Technology

Commodity.

6.1 Technical Infrastructure needed

Rationality, critical analysis, Development plan (good practices for source code development) and roadmap.

7 Business scalability

Metasploit and MySQL.

7.1 Evolution

Teams, Volunteers, Expansion. Where , How, Which mechanisms ?

7.2 Emphasis

Integration, Upstream.

References

- [1] Philip H. Albert,
Dual Licensing: Having Your Cake and Eating It Too,
<http://www.linuxinsider.com/story/38172.html>

- [2] Milking The GNU,
Dual-licensing: revoking the GPL,
<http://blog.milkingthegnu.org/2008/05/dual-licensing.html>
- [3] Milking The GNU,
Dual-licensing is unfair and community debilitating,
<http://blog.milkingthegnu.org/2008/05/exisiting-dual.html>
- [4] StackOverflow,
MIT GPL Dual-license in commercial software,
<http://stackoverflow.com/questions/3336161/mit-gpl-dual-license-in-commercial-software>
- [5] Producing OSS,
Dual Licensing Schemes,
<http://producingoss.com/en/dual-licensing.html>
- [6] Elena Blanco,
Dual-Licensing As A Business Model,
<http://www.oss-watch.ac.uk/resources/duallicence2>
- [7] Brian Behlendorf,
Open Source as a Business Strategy,
<http://oreilly.com/openbook/opensources/book/brian.html>
- [8] Martin Fowler,
FeatureBranch,
<http://martinfowler.com/bliki/FeatureBranch.html>
- [9] Jono Bacon,
The Art of Community,
<http://www.artofcommunityonline.org/>