TOSW 0.2.2

The Open Source Way

Creating and nurturing communities of contributors

Community Architecture

TOSW 0.2.2 The Open Source Way
Creating and nurturing communities of contributors
Edition 1

Author Community Architecture team@theopensourceway.org

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588 Research Triangle Park, NC 27709 USA

This guide is for helping people to understand how to and how not to engage with community over projects such as software, content, marketing, art, infrastructure, standards, and so forth. It contains knowledge distilled from years of Red Hat experience.

Appendix A. Revision History

Revision 0.2 Karsten Wade *kwade@redhat.com*

First conversion of 0.2 material to Publican

Introduction

This book provides an introduction to creating and nurturing communities of contributors. This is a core part of Red Hat's mission, as manifested in the many communities where Red Hat is an active contributor.

Red Hat employs contributors who work directly in upstream development on all parts of the open source architecture stack, from the Linux kernel to the thinnest skin of Ajax4Jsf. How these people interact with their communities is the first part of how Red Hat influences the growth of community.

The qualities of the interaction of each individual contributor are as different as any human-to-human interaction. Over the course of time, these many interactions distill to a set of guiding principles that help us with the community growth and nurturing.

Red Hat has collective community wisdom.

We also have particular skill and luck in being thought and action leaders in many open communities. It has all been done in an open, ad hoc fashion, and it hasn't been written down before.

So this book is a capturing of that distilled knowledge. It is also an active document, as that knowledge grows and changes, so do the contents of this book.

This book is small and written on a wiki to facilitate collaboration and growth. If you have ideas to improve the *The Open Source Way*, you are encouraged to just start writing it down. Use a page's "discussion" tab to access the talk page.

1.1. What this book is

1. A book that:
    a. ... describes a principle
    b. ... explains how to implement it
    c. ... and gives real world examples.
2. A thin reference guide to richer, thicker works that explain Big Principles in Lots of Fancy Words.
3. An open how-to manual, initially authored by the Red Hat *Community Architecture*[1] team.
4. Useful for much more than just code -- art/design, documentation, marketing, translation, IT, and so on

1.2. What it is not

1. Comprehensive for the sake of inclusion.
    - We are focused on slim and useful.
2. Overly repetitive of content already said and available elsewhere as free and open.
    - Build on the work of others via the power of reference.
3. Boring.

1.3. What it is influenced by

There are some very comprehensive, thorough, and useful works, such as:

- *Producing Open Source Software*[2]
- *Cultivating Communities of Practice*[3].

More influencers and good reads are found in *Chapter 9, Books you should read* and *Chapter 11, Data and references*.

1.4. Essential terminology or 'Read this even if you think you know what it means'

This section is growing ...

This section is growing for a while as the *The Open Source Way* is written and terms need defining.

1.4.1. Community

When people talk about community, they may be talking about very different things:

- People who *use* something
- People who *like* something
- People who *advocate* for a technology
- People who *enable others* to use something
- People who *contribute to improve* something
- ... and so on.

When we talk about community in the *The Open Source Way*, we mean the community of contributors who are a superset to all other communities. The ones who, by getting things done, make it possible for many, many more to get much, much more done.

This is the group of people who form intentionally and spontaneously around something important to them. It includes the people who use or benefit from the project, those who participate and share the project to wider audiences, and the contributors who are essential to growth and survival.

Contributors are the oxygen. Without it, the animal chokes and dies.

The community that contributes to improve something ...

The community that contributes to improve something is the one that defines the technology and the next generation of users, advocates, and enablers.

1.4.2. Tactics

This is derived from military terminology. The common example is a group of soldiers who work with a squad leader to take and secure a hill.

Tactics are the ideas, plans, methods, and means used to accomplish a goal. That is, tactics are the maneuvers used to achieve an objective that is set by strategy.

In Fedora, the strict yet clever *packaging guidelines*[4] are a strong-arm tactic that enforces the Fedora strategy of good packaging of only free and open source software.

1.4.3. Strategy

Strategy focuses on everything from which hill to secure, to deciding which miltary campaigns to engage in.

A term from military science, strategy is the focus of the command group. Strategy focuses on setting goals and which groups can obtain the goals.

Once the "who, what, where, when, and why" is decided, tactics takes over as the "how".

The Fedora strategy of creating and supporting only FLOSS is derived from the culture of the community combined with a pragmatic view of the legal and business landscape.

1.4.4. Planets and blogs

A *blog*[5] is a mix of personal and project writing that comes from a participant or contributor.

The idea of a *blog planet* is to show the width and depth of personalities in a community, and what they are doing.

It intentionally allows for personal expression. In other words, the content is not all community specific. It is sometimes wildly off topic.

The *Fedora Planet*[6] is a good example, with blogs aggregated from willing contributors. People add themselves to the planet, and it is a mix of languages, topics, skill levels, and project interest. All aspects of the Fedora Project are covered there on a daily basis, with a wide variety of contributors from Packaging, Art and Design, Marketing, upstream development, and so on.

1.4.5. Leaderless organizations

A leaderless organization is *decentralized*, meaning it does not rely upon a central authority for leadership, strategy, or tactics. Being decentralized makes it easier to heal, faster to respond and innovate, and more able to grow in scale.

An example of a large leaderless organization is the Internet. Not just the set of agreements that makes the structure work, but all the way down to the wire protocols that route around damage in the network, such as when a backhoe slices through a fiber link between two ISPs.

In projects, *Wikipedia*[7] is a decentralized organization, while the *Encyclopaedia Brittanica*[8] is a classic centralized organization.

[4] http://fedoraproject.org/wiki/Packaging
[5] http://en.wikipedia.org/wiki/Blog
[6] http://planet.fedoraproject.org
[7] http://wikipedia.org
[8] http://en.wikipedia.org/wiki/Encyclop%C3%A6dia_Britannica

The idea of the leaderless organization is put forth in the book *The Starfish and The Spider*[9].

1.4.6. Version control

Version control is the comfortable insurance that means you can sleep soundly at night.

A *version control system (VCS)* keeps track of the differences in versions of content. Properly implemented, it is nearly disaster proof.

When a document is saved in to a VCS, the difference between the current saved copy and the one just before it is stored. The same occurred at all previous saves. This makes it possible to restore a document to any point in its save history. It is also possible to return to a previous save point and start a new pathway of saves from there. In code this is called *branching*, with the original code in the central *trunk*.

When you work with a web-based tool to write, edit, and display a document, you are using version control. All content management systems and wikis use a form of version control to capture and display the differences between edits.

Version control provides noisy reporting. Anyone who is interested or able may watch the stream of changes, or derive reports from them. Watching and commenting on version saves as they occur is a hallmark of the open source methodology.

1.4.7. Content v. code repositories

Code is a specialized form of content. There are many similarities between repositories of code and content. The major difference is in the tools.

Code repository tools focus on centralized or decentralized development of the code. The ability to return to a point in time and branch from there is essential.

Content repository tools also include the ability to return to previous points in a documents save history. They also include management tools, such as team automation in MediaWiki, and workflow, publishing, and lifecycle components in a content management system (CMS).

1.4.8. Open collaboration tools

These are examples of open collaboration tools used in the open source communities. The same or similar tools apply to any online community of practice.

- What is a wiki?
- What is a mailing list?
- What is IRC?

1.4.9. Open marketing

This is marketing done entirely in the open, no secret discussions on brand tactics behind the Wizard's curtain. You talk about your strengths, weaknesses, brand position, and so forth as an ongoing open discussion.

[9] http://www.starfishandspider.com/

Central to this are social media tools. Some are used for discussions, some are for information dispersal, but any vector is a potential for learning and spreading the word.

- Blogging planet or otherwise aggregated feed of all contributors
- Publicly displayed and discussed content and code committing
- Always being watched
  - All mailing list traffic
  - All IRC logs
  - All voice sessions logged and available
  - 100% totally accountable discussions
  - Radically transparent

1.4.9.1. Fundamental rule - do not bash competitors

In fact, ignore them when talking about your project. Why waste the attention you've got on your project to point attention at something else?

1.4.9.2. Fundamental rule - be radically transparent

There is no such thing as too much information. Everything you can legally say, you should be saying.

Don't use this as an excuse to be verbose or disorganized.

1.4.9.3. Fundamental rule - explain what you know and how you know it

In the general category of being honest, spend some time with revealing sources and means. Bad numbers are everywhere in software and communities. Make sure your numbers are real and provable.

This also falls in the general methodology of thinking like a scientist.

1.4.10. Science

We may all be social artists in our communities, but in the end, our understanding of this is all driven by science.

There is an active feedback loop happening right now:

1. Scientific method wildly successful and bringing forth new, life altering ideas and innovations ...
2. Scientific method is a direct parent of the open source methodology ...
3. Open source way practiced in business/private sector ...
4. Influences academia ...
5. Influences public/government ...
6. Funds science and supports innovation ...

7. Iterating through more ideas for the Great Ones ...

8. (Back to top)

The open source way is breaking through to help make new ideas happen ... in organizations that brought us the ideas of free government and free, public education ... ideas that were revolutionary and evolutionary in the 19th century ... that fueled a century of innovation ... and are now getting refueled by new ways of openly collaborating on infrastructures of freedom.

Communities of practice

2.1. Introduction

> The idea that an organization is a constellation of "communities of practice" is a genuine breakthrough, and that overused word "breakthrough" is merited. It is an idea that had profound implications for what it takes to run a successful organization in our frenetic, chaotic times. In this book, Wenger lays the groundwork for the kind of thinking that will be necessary for any surviving organization in the 21st century. Wenger and the Institute for Research on Learning are defining the cutting edge. And they are right! Pay attention! Please!"
>
> —Tom Peters, about Etienne Wenger's book *Communities of Practice*

2.2. What is a Community of Practice?

It seems to be a commonly held opinion that principles of the open source way are limited to the practice of software development.

In fact, the open source way is an instance of a "community of practice", which exist in varying forms all around us. Etienne Wenger, the leading theorist of communities of practice, [http://www.ewenger.com/theory/ defines] the term as follows:

> Communities of practice are formed by people who engage in a process of collective learning in a shared domain of human endeavor: a tribe learning to survive, a band of artists seeking new forms of expression, a group of engineers working on similar problems, a clique of pupils defining their identity in the school, a network of surgeons exploring novel techniques, a gathering of first-time managers helping each other cope. In a nutshell: **Communities of practice are groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly.**
>
> —Etienne Wenger

2.3. Elements of the Community of Practice

Every community of practice consists of three structural elements:

- *Domain*. The domain is the area of knowledge that interests the community. In free software, the domain is usually a particular technical problem that needs to be solved.
- *Community.* The community is the set of people who care enough about the domain to give their own time to participate. In free software, even though domain may be very specific, interested community members can come from anywhere that's connected to the Internet -- which is one of the factors that makes the community software development model so powerful.
- *Practice.* The practice is the way that work is done, by the community, to further their goals in regard to the domain. All frameworks, tools, ideas, stories, documents, legal entities, code, and so forth, are all part of the practice. It's the work, and all the tools used to get the work done.

2.4. Principles for Cultivating Communities of Practice

In his book *Cultivating Communities of Practice*, Etienne Wenger proposes seven principles for successfully cultivating communities of practice. Anyone who is responsible for moving a community forward towards its goals should consider reading it.

2.4.1. Design for evolution

When starting a new community effort, it's difficult to know what form it will take. Volunteers are not employees; they can only be influenced, never ordered. Some may take passionately to the proposed project; others may only be able to give some of their time.

Be ready for that. Concentrate on simple lists that encourage accountability. Know who's doing what. Maintain a flexibility of structure, so that if someone to have an idea that takes your effort in a completely different direction, you don't have to change all your rules. Only have as much "governance" as you need at the time, never more.

Also: volunteers leave. It's a fact of life. Make it easy to hand off tasks to newcomers, and work to generate newcomers, so that your old-timers don't feel obliged to continue doing work that they no longer have the time or inclination to do.

At *teachingopensource.org*[1], there are a number of working groups. Some of them are active; some of them are not. Creating a new working group is deliberately kept simple; identify a "leader" in charge of moving things forward, and claim a wiki page, and you're all set. Working groups are evaluated periodically by the entire group, and if it's generally agreed that no progress is being made -- often because the leader doesn't have the time required -- the working group either finds a new leader, or is disbanded. Simple process, until more complex process is needed.

2.4.2. Open a dialogue between inside and outside perspectives

In any community there are always "insiders" -- i.e., the people who understand the problem space very well, the people who are at the core of the domain space -- and people who are "outsiders", but have enthusiasm, some domain knowledge, and a willingness to help. A successful community uses both of these perspectives effectively, because it's the outsiders who are most likely to impart new energy and new perspectives.

There are different kinds of insiders and outsiders. A company's employees can be the insiders, and the customers can be the outsiders. One particular team can be the insiders, and other employees can be the outsiders. Beyond companies, longtime members of a community become insiders over time, and newcomers almost always start as outsiders.

For example, when Red Hat first started the Fedora project, many of the critical decisions were taken by a committee composed only of Red Hat engineers. A number of critical tasks were assigned to "insiders", and these tasks languished. In the meantime the "outsiders" struggled to find meaningful tasks to help the project's growth. The public perception looked a lot like *this*[2]. The dialogue between the groups made it clear that externalizing the tools and processes were key to community growth.

[1] http://teachingopensource.org
[2] http://lwn.net/Articles/83360/

2.4.3. Invite different levels of participation

Often, the hard tasks at the center of a domain can only be tackled by the insiders -- but if only the hardest tasks get focus, and only by experts, then those new to the domain do not have opportunities to gain expertise.

One of the important concepts espoused by Wenger is *legitimate peripheral participation*. It is, essentially, the idea of apprenticeship; the key difference is that, rather than being apprenticed to an individual, a new practitioner can be apprenticed to the entire community of practice.

In a volunteer community especially, people who join are going to be invested in learning more and doing more, and it's important to identify work that matches the newbie's skills, invites them to stretch those skills, and provides people who can help them develop those skills as required.

As an example, learning to package software is not easy -- but some software is easier to package than others. Newbie Fedora packagers are invited to learn how to package fonts, since they are simple and very uniform in how they are packaged. There are also lots and lots of fonts out there that need to be packaged. It's a perfect "on-ramp" for newbie packagers, and because there are so many fonts to be packaged, there are plenty of opportunities for newbies to be immediately useful.

Experts are great, but they're hard to find.

Experts are great, but they're hard to find. The way to find more experts is to invest in a process that continually creates more experts.

2.4.4. Develop both public and private community spaces

Avoiding the cabal mentality does **not** mean having every single conversation in public, ever. Transparency is great, but it is unreasonable to expect everyone to be comfortable with full transparency, all the time.

Also, one-to-one communication builds intimacy and trust that multiway communication cannot. Especially as new participants are building confidence, it's vitally important to build private relationships between individuals. Certainly it is appropriate to encourage important conversations to be moved into public forums, especially conversations about actions that will affect others. But private chats are important too, and often useful for eliciting insights that help move the more public conversations forward.

In particular, private conversations can be critical to resolving conflicts within communities, and community leaders who are seen as responsible for the community's health should be comfortable taking the parties aside. There have been many examples in the Fedora community of conflicts that were better resolved in private, and there will be more in the future.

2.4.5. Focus on value

No volunteer wants to spend time on work that nobody values. Therefore, encourage community members to express the value that they receive from the community, and to reflect on the value that they provide.

Also understand that not everyone's notion of value needs to agree; so long as participants do not actually detract by participating, they should feel free to add value in whatever way they see fit. Core participants frequently do not value a set of contributions initially, and only come to understand

and appreciate that value later. Even contributions that are wildly experimental and far from the mainstream, and may not seem at all valuable, should be respected and encouraged.

Example: There was a time, not so long ago, when many central members of the Fedora community saw a Live CD as a waste of effort. The Live CD is now one the most important deliverables of the entire Fedora community. But it was clear to the initial contributors that a good Live CD was critically valuable, and the community's embrace of that work led to a critical mass of contributors to focus on, and solve, the problem.

2.4.6. Combine familiarity and excitement

Stable and familiar working processes are vital, because people need tasks to focus their day-to-day work.

Still, people can not thrive on heads-down tasks alone. Exciting new challenges create opportunities to energize old friends and attract new ones, and give volunteers an important sense that they are all wrapped up in a great and important challenge. This excitement is crucially important to keep volunteers motivated on the daily work.

Example needed.

2.4.7. Create a rhythm for the community

The pace of engagement is crucially important in a community of doers.

Moving too quickly and demanding too much, too soon, can leave volunteers frantic and feeling like they can't keep up. Moving too slowly can lose volunteers who do not see enough activity to hold their own interest.

The weekly IRC meeting has been a hallmark of most successful Fedora projects. For some projects, a weekly meeting may be too much, and for other projects, a weekly meeting may only be a way to checkpoint activity that is going on constantly. Either way, building and maintaining a sense of rhythm is crucial for a healthy community.

2.5. Learn more

To learn more about how to cultivate communities of practice, read Etienne Wenger's book on the subject:

http://www.amazon.com/Cultivating-Communities-Practice-Etienne-Wenger/dp/1578513308

How to loosely organize a community

This chapter explains the basic structure of a community that is succeeding and evolving.

We often call this "community growth", with the caveat that healthy growth is not boundless growth.

Control the growth -- you water and feed, but you also prune.

If you look at an open source project with a long history, you see these guiding principles of how to loosely organize a community. Going through the history of the BSD project, you see they discovered and iterated through everything from mailing lists to open access code repositories. Their 30+ year lifespan can be viewed as a stretched out version and variation of the Fedora Project's 15 year lifespan from Red Hat Linux to Fedora Linux.

3.1. Community soil - true-isms to grow with

You have one of two goals in your community building plans.

1. You want to create and be central to the success of a community effort.
2. You want to be a catalyst in a community that includes you but is not relying upon your individual or team efforts for survival.

Arguably, the second goal is the preferred goal for any Red Hat community activity that leverages community growth to continuously increase the value of our investment, while not having to increase the actual investment.

When we look at the most successful Red Hat products, they come from projects where Red Hat's relative investment over time remains flat or decreases in comparison to the ongoing value. The Linux kernel forms the core of the Red Hat Enterprise Linux distribution, but Red Hat does not employ the greatest number of kernel contributors, just the ones that do the most work that matters to Red Hat as well as the kernel community.

When we look at Red Hat projects that fail, the early investment was high and stayed high throughout the project's lifespan while the community continued to never appear.

Regardless of which goal you have, the methodology is the same.

3.1.1. Initial building the soil or 'Get it going'

3.1.1.1. Practice radical transparency from day zero

Avoid private discussions

Making important decisions in private is like spraying contributor repellant on your project.[1]

1. From the earliest project moments all discussions and content (documents) need to be openly available and version controlled.
2. All discussions and decisions must be done in the open and be archived.

3. Wiki page changes (such as [[Special:RecentChanges]]) and code commit logs to the mailing list until it becomes annoying.

Do not fall in to the mistake of doing private email discussions to "get things started quickly." You can have all of the needed collaboration tools available within 2 hours and $US15/mon under a custom domain on any number of Linux-based hosting services (at a bare, bare minimum.)

Do not underestimate the importance of the right set of tools available at community start or continuance.

Example needed

3.1.1.2. Get things started immediately with the simplest and most open communication methods available plus a meeting time

Focus first on enabling communication, then people can self-organize around the work that needs to get done.

Don't try to get everything ready before revealing to the world. Other people want to help get it ready; it won't be any fun if you do all the work before they get there.

1. Make sure that meeting happens regularly.
2. Use it to discuss tasks, leave tactics and strategy for the mailing list.
    - This ensures the widest participation in the important guiding activities (tactics and strategy).
    - #* The meetings become a regular check point/drum beat to ensure progress is made and things are not forgotten.

Example needed

3.1.1.3. Start open marketing soonest

You aren't working on a stealth-mode start-up, you are trying to grow an open project. Make appropriate-sized noise at the earliest opportunities.

Remember that “it is better to do than to seem”. Don't just talk about ideas -- talk about ideas and do stuff about them, too.

1. At least the leadership needs to blog on relevant technical planets (GNOME, Fedora, etc.)
2. Social media information feeds aggregated on a page for reference
    a. If you are like many people, you are the most excited at the beginning of a new venture. That is when you want to set your tone of voice and participation bar, to give yourself the highest open marketing standards to maintain throughout the community life.

Example needed

3.1.1.4. Quiet projects stay quiet. Noisy projects get noisier.

At this point, you have things moving in some direction, even if it is only three of you discussing things loudly in public.

What is going to draw new people are:

- Interesting parts of the project that match their passions.
- The project itself inspires passion in people.
- Some other passion drives people toward the project.

People find out about this through your rigorously open marketing what you are working on.

What are developers working on and why? Expose every part and reason in a well organized wiki page.

What supportive pieces are not owned or in danger of being orphaned? Make a list and keep it updated.

Example needed

3.1.1.5. Tasks, tasks, tasks or 'Project management matters'

The first item on your task list should be, "Write initial task list, prioritize, assign dates and doers."

An updated task list says, "We're getting things done, here's where you can help."

Think of it like putting a few euros and coins in a guitar case when busking (playing for money in public), or seeding a tip (gratuity) jar with a few dollar bills. Entice people to fill in the blanks of a few "Unassigned" items without making it totally scary by being too empty.

One simple project management method is to assign around 50% of the unassigned tasks to the project leader, who then has a lot to delegate when asked.

Too empty is scary

Too empty is scary. Too full makes it look too late to help. Balance your task list, have at least 60% of your tasks assigned.

If you have more unassigned tasks than 40%, take out some tasks, or mark them as a lower priority. It is a sign that you are reaching too far at this stage. You aren't going to do them anyway, and they can be added back later if still viable.

Example needed

3.1.2. Ongoing soil support principles or 'Get out of the way'

We presume you are looking to get the most value from your community growth, meaning you understand you want it to scale beyond your ability to be the central person. You are seeking a leaderless organization, or as near as you can make it.

3.1.2.1. Turn over leadership tasks to natural community leaders

As you lower the barriers to participation, as contributors arise from the participants, natural leaders arise from the contributors.

You can tell a natural community leader:

- They listen when people talk.

- They talk, people listen.
- They get stuff done as much as or more than talking.
- They inspire other people.
- They demonstrate a sense of ownership of part of or the entire project.
- They are eager to own a task or project from the beginning.
- They work and play well with others.
- They show common sense.
- They do not run with scissors.
- They tend not *to feed the trolls*[2].

Take that person aside and say, "Hey, you seem to be in charge of/most interested in Foo, can you be the project leader there?" At worst, they say no. At best, you just helped give birth to another community leader. Congratulations!

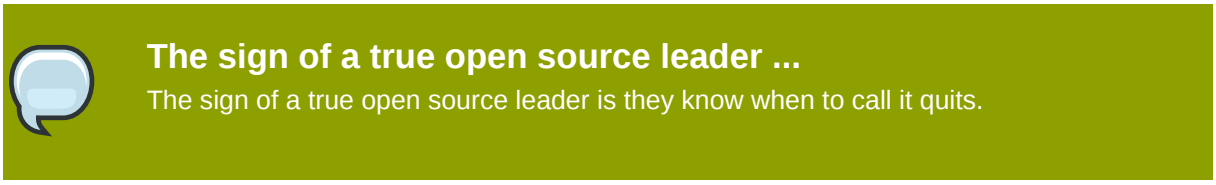
When Fedora decided to migrate to MediaWiki, the Infrastructure team got an interesting email. Ian Weller wrote, "If you need any help, give me a shout." Turns out he was a minor participant/contributor in Fedora but was passionate about MediaWiki. Within a few weeks, he was debugging migration scripts, writing how-tos, and leading various Fedora teams in best-practices about using MediaWiki. After a highly successful migration and subsequent build-out, Ian was named 'Wiki Czar' in January 2009, *nominated by a member of the Community Architecture team*[3].

3.1.2.2. Turn over project leaders regularly

Often a single person sits in the center of a group of contributors, acting as a go-between, arbiter, soother of feelings, and go-to person. We call such people "project leaders". They are your best friends. They might even be you!

If they are you, you need a six month exit plan.

You may not pull the trigger right away, but you need to be prepared to.



At the *Moosewood Restaurant*[4], the collective *runs the kitchen using a rotational scheme*[5]. A kitchen worker rotates roles:

1. One week as the head chef -- plan menus, order food, touch each plate before it goes to a diner
2. One week as the sous chef -- right-hand to the head chef, responsible for food before it leaves the kitchen/line

[3] http://lists.fedoraproject.org/pipermail/fedora-wiki/2009-January/000089.html
[4] http://www.moosewoodrestaurant.com/aboutus.html
[5] http://search.barnesandnoble.com/Sundays-At-Moosewood-Restaurant/Moosewood-Collective/e/9780671679903

3. One week as a line or prep chef -- fixed work station area, run the process and work hard
4. One week as a dish washer -- nothing like seeing up close what people don't eat
5. Back to the top

There is no job in the world that a fresh mind and perspective cannot gain from.

Most of us are not Linus Torvalds ...

Most of us are not Linus Torvalds. Don't be afraid to find a leader to replace you.

3.2. Community building tools - just enough to get the job done

3.2.1. Initial tooling or 'Get it going'

3.2.1.1. Set up a mailing list first

An open collaboration needs a relatively low barrier to entry communication method that is:

1. Open to all to read;
2. Open to many to write to;
3. Subscribable;
4. Openly archived.

A mailing list fulfills this need.

Bonus for doing it on open infrastructure using open source software.

That is a recursive use of the value of the open source way as experienced through quality software.

Mailman[6] is the de facto standard.

`Example needed.`

3.2.1.2. You need a version controlled repository for content - code and documentation and art and etc.

For more information on version control, read *Section 1.4.6, "Version control"*

Version control is the insurance that makes you and your contributor community bold.

1. This is a code repository (git, Subversion) and a document system that has the lowest barrier to entry (wiki).

[6] http://list.org

2. Look at existing best-of-breed hosting, e.g. fedorahosted.org.

3. Making giving access to this as easy as possible; do not let the administration fall between the cracks.

Example needed

3.2.1.3. Use lightweight, open collaboration tools - wikis, mailing lists, IRC, version control, bug trackers - and give out access

To quickly gain momentum, a series of small and useful tools always trumps a monolithic, inappropriate, hard to use tool.

People are familiar with certain tools already -- give the people what they want.

The choice here does not need to be open source. If you choose a non-open solution, you incur additional risk for whatever opportunity you are trying to capture. Make sure that a move to fully open tools is part of your roadmap for the project.

Some people will show up to participate no matter what tool you choose. Another group will participate only if the tool is open source, with some preferring popular tools. Why not choose an open, popular solution and capture all groups?

1. Default open subscription is the rule

2. Spread admin rights to anyone responsible; try to pair with people outside of your organization

3. Encourage people to be bold

   - Don't be afraid to roll back bad decisions, that is what version control is for

4. Be bold yourself

Example needed

3.2.2. Ongoing tools or 'Get out of the way'

3.2.2.1. Improve your infrastructure to improve your project

The community needs to keep finding and building tools, processes, and especially automation.

Automated testing, automated building, automated error checking, lint scanners run on code and content, wiki patrolling, etc.

You do not need to always build all this yourself. For example, projects hosted on fedorahosted.org gain everytime a new project is hosted or Fedora Infrastructure adds features to the web apps or the *Fedora community panel*[7].

Example needed

[7] http://fedoraproject.org/wiki/MyFedora

Stuff everyone knows and forgets anyway

Use this chapter to remind yourself and everyone else.

4.1. Embrace failure

Fail spectacularly, and don't forget to take notes.

There is a lot of crowd wisdom at work here. It's basic physics, too -- if you charge ahead and fall on your face, you at least made headway. Like feeling your way around in the dark, each touch, bump, and stub of the toe teaches you more about the environment.

When you drive your processes and technology to the brink of failure, repeatedly, you get very comfortable somewhere most people don't even want to go. You also get to set a new standard for what is comfortable for your community, giving you new room to expand even further.

Example needed

4.2. Communities require care and feeding to get started ...

Imagine if you grow a garden and are very ambitious. The first year you overplant and are unwilling to thin out the seedlings. You end up with overcrowded, unhealthy plants.

Implementation needed

By giving good and regular care, your communities ...

Example needed

4.3. ... And communities need to be left to grow and evolve

Principle needed

Implementation needed

Example needed

4.4. Remember what you learned in Kindergarten

- Play nice
- Share
- Don't run with scissors

Seriously, you would be surprised at how many of the most basic rules are broken that our parents and teachers thought they taught us many years ago.

Implementation needed

Example needed

4.5. Take extra extra extra care to have all discussions in the open

Principle needed

Implementation needed

Example needed

4.5.1. Radically visible meetings at all times

Any private interactions, from the hallway to email/irc to phone calls, are a risk to the project. At the minimum, you must be mindful of reporting back the results of direct conversation.

However, this isn't really good enough. A summary of a meeting never shows the reasoned discussion, effectively cutting the community out of the decision making process.

Implementation needed

There is a reason mailing lists and open chat networks are the baseline for all communication in successful open source projects.

Example needed

4.5.2. No decision point is too small to pre-announce to a mailing list

While we can grow trust in the community about technical or other decisions, there are common problem circumstances we can avoid:

- The corporate sponsor and staff are always more suspect, so need to take extra care to share the decision making process
- We cannot guess what is important to contributors, nor why. Presuming they don't care about a decision is a bad idea.

Implementation needed

The method is to take all decisions to the open forums, until people start complaining that a particular class of decisions can happen in another way/place. This is called, "Building trust by proof through annoying transparency."

Example needed

4.5.3. How to let a mailing list run itself

A mailing list is the core of any active community that gets things done. It is the equivalent to having everyone in the same office all the time. The functionality has not been duplicated, replicated, or replaced with any other technology in more than twenty years. News service is arguably better, but mailing lists are ubiquitous.

It's not hard to run a mailing list, in fact the best ones really run themselves. In order to get it to that state:

1. Everything must be discussed on the list.
2. If you break rule 1, you must make sure that whatever the discussion was, the details and results are published on the list
    - This happens often, so it is incumbent that decisions come back to the list from IRC, phone calls, f2f meetings

4.6. Take even more care to do all design and decisions in the open

Principle needed

Implementation needed

Example needed

4.7. Use version control for your content as well as code - everyone watches the changes

Being able to roll back work encourages people to be bold and innovative. One of the currencies of project work is simply making a change to a codebase or a document as an explanation itself. Changes are self-evident in the commit itself. This is why all changes must be accompanied with a comment or summary that explains the reasons behind the change.

Equally important is that all project members watch these changes. Responding to the changes on a mailing list, bug report, or wiki discussion page keeps the discussion associated with the node of origin.

Example needed

For example, major design decisions need to ...

4.8. Choose open tools that can be extended

Don't choose tools just because your small team knows them the best. Be careful of choosing something because you are in a hurry and need something quickly.

When it comes to tools, talk with trusted community leaders about what you should use. Even if you choose something out of step with other community members, you will at least know why you did and be prepared to mitigate risk.

It is risky to choose closed tools ...

It is risky to choose closed tools that a community cannot easily scale itself.

Implementation needed

Example needed

4.8.1. Make sure everyone has an equal and clear method for access to write-commit to open tools

Principle needed

With version control under code and content, you can open access to more people. For example, don't require people to go through a lengthy process or to "prove" themselves with X number of patches. Give people access to whatever they need to make a difference, even if they don't use the access.

Wikipedia is a prime example here. They use the power of the community size to police bad content changes, rather than limiting the community size behind rules of what it takes to write content changes.

More people want to do good for your project ...

More people want to do good for your project than want to do bad. Don't punish the do-gooders because of the potential evil someone might do.

4.8.2. Tie this together with open content

Principle needed

Don't forget to document code, processes, methods, reasons, etc.

Implementation needed

Example needed

4.9. Focus on healthy and open community interaction

Principle needed

Implementation needed

Example needed

4.9.1. Make governance as clear as possible

Principle needed

Implementation needed

Example needed

4.9.2. Use your lawyers well while avoiding too much legalese

Principle needed

Implementation needed

Example needed

4.9.3. Do not let poisonous people bog down the community

Principle needed

Implementation needed

Example needed

4.9.4. Communicators need to communicate - do not let liaisons go silent

Principle needed

Implementation needed

Example needed

4.9.5. Disable poisonous communication

Principle needed

Implementation needed

Example needed

4.10. Seek consensus - use voting as a last resort

Most decisions in an open community are not decided by a vote. As in a well-working democracy, decisions are handled by the experts and knowledgeable people who are in charge at the will of the people around them.

Voting is best left for deciding who is in charge and the occasional very contentious issue. Don't you wish it weren't contentious? Wish you could go back and make a consensus?

Example needed

4.11. Reassign your benevolent dictators while evolving toward a consensus-based democracy

http://producingoss.com/en/consensus-democracy.html

Principle needed

Implementation needed

Example needed

4.12. Do not forget to release early and release often

Principle needed

Implementation needed

Example needed

4.12.1. Release early and release often is for more than just code

Every idea deserves the light of day as early as possible.

A common mistake is to wait to release a document, process, marketing plan, etc. until it is "just right". People who otherwise fully understand how not to do this in code forget when it comes to other parts of a project.

Apply this rule:

- If a piece of information is confidential, keep it private;
- Otherwise, get it out in the public soonest ...

... using techniques such as these ...

- Bounce the idea off the mailing list(s)
- Make a wiki page
- Add to another wiki page
- File a ticket in a tracking system
- Make a non-coding prototype
  - *Wireform* web page in a wysiwyg editor
  - Inkscape or GIMP mock-up/collage
  - Bar napkin and pen, scanned or photographed

Example needed

4.13. Evolve with the growth of your audience and contributors

It's easy to say, "We designed this for such-and-such a reason for this particular audience." It's not so easy to be the only creator and maintainer, though. If you want to grow beyond just you and your niche group, you have to be willing to accept the influx of new ideas and directions that comes with open collaboration.

Implementation needed

Example needed

What your business does correctly when practicing the open source way

This chapter is a pat on the back for what you do correctly.

It is also your reminder -- don't forget to keep on doing these parts that work well.

Continuous improvement means we are always building on what we have learned before. It is not a single state that you achieve to remain static.

For example, the course of the Fedora Project as a source for a Linux distribution demonstrates the spirit of continuous improvement in the open. Simply converting the community Linux distro to a non-commercial project was not enough. It took six releases of Fedora to get over the hump of enabling the wider community to contribute without inhibition or barriers.

5.1. Identifies and focuses on key future technology

The technology you defined two years ago, perfected a year ago, and are making money on today, is going to be a commodity a year or two from now. This is one constant we can rely upon.

Open source communities force your organization to keep moving, to never get comfortable in one spot for too long. This is an effect of the rapid way open source software can advance, and it works to your advantage to always have little brothers and sisters nipping at your heels.

Early in the 2000s, Red Hat provided the industry's best system management for Linux via RHN. Because that project was internal only and so was limited in its future by exactly how many people Red Hat hired to work on it, there was a natural limit to how long RHN would remain in a leadership position.

Although arguably Red Hat was able to keep RHN relevant for an additional few years by not engaging in external system management projects, the resulting situation in 2009 is quite opposite. RHN has now drifted backward in relevancy compared to open source alternatives, and the open source tools far outstrip RHN capabilities where it matters the most to customers.

The technologies in the best position to replace RHN include Nagios, Puppet, Cobbler, Func, and Yum -- the first two were started by Red Hat engineers and maintain strongly influenced by Red Hat needs, while Yum was made successful by being the better and open dependency resolver that the RHN tool 'up2date' was not. It took a few years longer in this case, but Red Hat is getting to the next level by re-focusing on the key technology.

5.2. Funds key open source development

You may have the power of the pocket book. By applying funding intelligently, you can be instrumental in the initiation and growth of important open source projects.

The best implementations of this idea are when an initial high investment is followed by a low and ongoing investment that remains relatively level or linear in progression. As the project grows, your relative investment grows at a fraction of the rate of the overall project; this is where the low-investment, high-return of open source contribution occurs.

Another interesting implementation is when you can direct "other" organization's funds at open source.

By learning from when your investment does not pay off, such as ongoing high-investment with not enough return, you gain value from mistakes made.

For example, the NSA partnered with Red Hat to work on getting SELinux upstream in the Linux kernel. The NSA goal was to have Red Hat bake SELinux in to a commercial off-the-shelf (COTS) package that US Government agencies could procur. While Red Hat was making a technology bet that included hiring key and unique developers from the communities, it was also helping the NSA to direct their efforts in the open community.

Back to the idea of spending your own money, there are a number of key technologies and projects that stand-alone now, but owe some initial lifeblood to market interests. At Red Hat, these include: GNOME; XEN virtualization and the advancement of virt under Linux (libvirt, oVirt, etc.); RPM.

For a more comprehensive list, refer to *http://fedoraproject.org/wiki/Red_Hat_contributions*.

5.3. Makes mistakes and learns from them

Principle needed

Implementation needed

Example needed

5.4. Is daring about applying the open source way to non-engineering teams

Principle needed

Implementation needed

Example needed

From design thinking to IT's Genome project, the open collaboration methodologies are used throughout Red Hat to set strategy and achieve goals.

5.5. Works hard to preserve culture in a fast growing company

Principle needed

Implementation needed

Example needed

5.6. Takes people who think they understand the open source way and really teaches them how it works

Principle needed

Implementation needed

Example needed

Including discussion around open source in new hire training is an example of this.

Rethinking how departments are structured and combined, and how they collaborate, is another way of teaching and permeating a culture of open source.

5.7. Has a strong brand and steers that consistently through community interactions at all levels

Principle needed

Implementation needed

Example needed

From CxO interactions to customer and free community relationships, there is a remarkable consistency to the Red Hat brand. This is due to more than a strong campaign by the Brand team. It also requires the brand message to live inside of Red Hat associates. It comes from people associating the open source way with the way Red Hat does business.

What your business does wrong when practicing the open source way

If you belong to an organization, such as a business, you may apply the open source way to how you do business. How you plan long-term strategy and how you execute short-term tactics. This is one way that the open source way is as set of principles applied to differing disciplines.

For more of an overview on applying the open source way to business practices, read *Chapter 7, Business the open source way*.

In this chapter, various common mistakes are discussed. Red Hat shares some examples, do you have any others from your team or other organizations that apply?

6.1. Burns projects and losses momentum in technology leadership by ignoring community fundamentals

One of the challenges of doing business the open source way is the balance of experience and expectations in your employees and shareholders. Accurately practicing the principles of open source requires diligence and patience. Sometimes other instincts and requirements cross the pathway of essential community principles.

Putting your community principles first in action and in risk analysis is essential to accurate practice of the open source way.

Ironically, even the best organizations are not always the best at what they do. Sometimes we break every rule in *The Open Source Way*.

`Example needed`

6.2. Breaks fundamental rules of community

In fact, every rule is being broken right now, somewhere inside of our favorite organizations.

Somewhere project team members are designing in the hallway instead of on a mailing list.

A team leader is wondering if they really need to go through the trouble of getting community input on their work.

Someone in a sales organization is failing to show customers the value of being involved in the design and development of the software core to running the customer's business.

For example, there is a common practice of having two mailing lists for open projects. One is an internal list that too often gets used for development discussions before bringing them in to the open. The other is the public discussion and development list.

Similarly, when dealing with ISV partners, the partner team makes tools, content, and collaborates all on a back-channel.

The problem is, community members either know or can feel there is work happening before conversations get to the public list.

6.3. Fails to learn from mistakes

Our ability to institutionally learn from past mistakes can be hobbled. This is particularly challenging when the open source way is not ingrained in to all aspects of your business; this is rather common, keeping open source as a development methodology but otherwise running the business in a traditional fashion.

The Open Source Way is one part of fixing this. We need roadmaps even when we think we know all the routes by heart.

As with many of these "does it right, does it wrong" items, there is an irony at work here. You may have learned serious lessons from past mistakes, yet still see many of the same ones repeated continuously.

You find this when the organization is not listening to the memory keepers, who attempt to speak up and remind that this has been tried before.

For example, trying to solve problems with closed, proprietary software because of a desire to get something running quickly. Experience shows that, just as we try to teach our customers, over the medium to long term, that decision is going to cost more than engaging with an open solution from the start.

6.4. Misses the opportunity to adopt and grow key or future open source technology for your own IT

Even open source businesses have a history that is riddled with decisions to use a piece of proprietary technology for internal or external purposes. In some cases there was no open source choice to adopt and the cost of starting from the beginning was considered prohibitive.

There are, however, as many cases where the open source software could have been adopted and fixed up to the needed enterprise standard by using the open source methodology.

`Example needed`

6.5. Does not include community strategy in all new projects

Even a project that is only for internal focus and consumption can benefit from considering and including its community.

In other words, we can adopt open source methodologies for everything from our internal communication to brand marketing.

At the Google headquarters, there are brief articles posted on the bathroom walls, along with encouraging signs. The articles are short and technical, detailing how to use and test a particular piece of Google technology. The other signs and branding encourage Googlers to beta test their own software. This is sometimes called *eating your own dog food*[1], that is, using the products and methods yourself that you sell and encourage your customer communities to use.

[1] http://en.wikipedia.org/wiki/Eat_one%27s_own_dog_food

6.6. Separates community from business

Like many other companies, your business distinguishes clearly between the worlds of "development" where you support and work with communities, and "business", where you have adopted hierarchical controlled structures.

This mirrors a traditional business approach, where interacting with communities members is analagous with customer service, a sub-branch without real power to change the rest of the business.

Sometimes from a historical evolution, this separation is comprehensible. Unfortunately, we can't always explain these histories so they makes sense in the modern context where open community organizations act in a more integrated manner.

A key community growth initiative that will benefit the bottom line is to make growing various types of communities (customers, sales, partners, etc.) a top-level strategic goal. Agree that you need to include community interaction as part of the feedback loop that improves your business and processes.

Think of every way that "customer" and "customer service" have integrated with various parts of successful customer-focused businesses. Then substitute "community" and "community enablement".

There are examples out there, where companies successfully integrate community principles into their business and gain benefits from their customer and partner contributions as they do from development.

A good exercise is the comparison of community integration on the web sites between *http://www.redhat.com* and *http://www.jboss.com JBoss* on one hand and *http://www.oracle.com Oracle*, *http://www.sap.com SAP*, *http://www.novell.com Novell*, and *http://www.ibm.com IBM* on the other hand. You can see the strength of the stand-alone community, but in the former you don't see obvious community offered for customers and partners. In the larger, traditional, proprietary companies, they have put forth the face of having learned they need to be a leading member in the community of their customers and partners.

A similar exercise can be done with smaller pure and mixed open source ISVs. The community is forefront and listening to/interacting with users is a key to their growth.

Ironically, applying the open source way via community principles is sometimes more evident in companies that do not practice open source development. Twitter, for example, builds on open technologies, has a rapid and iterative approach to incorporating user ideas, and otherwise shows a high level of listening to and learning from end-users that is similar to how open source listens to end-users.

Business the open source way

This chapter is an overview of how a business can use the open source way to success in growing revenue and brand. As in other chapters, principles are explained, implementation is described, and examples are provided.

Resources: *http://darkmattermatters.com*

Who else to watch

This section generally lists individuals and organizations external to Red Hat that are useful to watch their approach to community.

A sustainable community underpinning is continuous improvement. To improve, we learn from each other.

For that reason, this book needs "your" help. Add to these lists.

We could list names and organizations endlessly. Try to pick the ones that matter.

8.1. People

- Endless list?

8.2. Groups

- Not quite such an endless list ...

Books you should read

Suggested books of various community experts.

Producing OSS

Author

Karl Fogel

Link

http://www.amazon.com/Producing-Open-Source-Software-Successful/dp/0596007590

Website

http://producingoss.com/

ISBN

ISBN-10: 0596007590 and ISBN-13: 978-0596007591

The Starfish and the Spider

Authors

Ori Brafman and Rod A. Beckstrom

Link

http://www.amazon.com/Starfish-Spider-Unstoppable-Leaderless-Organizations/dp/1591841437

Website

http://www.starfishandspider.com/

ISBN

ISBN-10: 1591841836 and ISBN-13: 978-1591841838

Groundswell

Authors

Josh Bernoff and Charlene Li

Website

http://www.forrester.com/Groundswell

ISBN

ISBN-10: 1422125009 and ISBN-13: 978-1422125007

Wikinomics

Authors

Don Tapscott and Anthony D. Williams

Website

http://www.wikinomics.com/book

ISBN

ISBN-10: 184354637X and ISBN-13: 978-1843546375

Cultivating Communities of Practice

Authors

Etienne Wenger, Richard A. McDermott, William M. Snyder

Link

http://www.amazon.com/Cultivating-Communities-Practice-Managing-Knowledge/dp/B000SEOINI/ref=dp_kinw_strp_1

ASIN

ASIN: B000SEOINI

Community Architecture dashboard

Red Hat's Community Architecture team supports our executive leadership team by providing a dashboard-view of community health information. The communities on the dashboard list vary depending on interest and need. For example, we are not actively watching the health of our Linux kernel or GCC contributions. This is because those communities are self-sustaining and extremely healthy. Instead, the dashboard watches important, struggling, and nascent community efforts that either the team or the corporate leadership team are interested in tracking.

Current aspects of the dashboard:

- Data gathered through *EKG*[1]

Data and references

This chapter contains copies or links to data and references used in this book.

11.1. References

These are online and offline references that support conclusions drawn elsewhere.

- *Interview with Seneca College's David Humphrey*[1], where he gives a succinct list of success factors.
- *Josh Berkus' Ten Ways to Destroy Your Community*[2], with some replies to Rodrigues' rebuttal (below.)
  - *Colin Charles post on the presentation*[3]
  - *Zack Urlocker post on the presentation*[4]
  - *Savio Rodrigues rebuttal post on the presentation*[5]
- *The top ten rules*[6] from *The Starfish and The Spider*[7].
- *Open Source Triple Play*[8]
- *How to get your code into an open source project*[9]

11.2. Data

This section contains copies of relevant data for immediate reference, as well as links and sources.

How to tell if a FLOSS project is doomed to FAIL

(This was originally written by *Tom 'spot' Callaway*[1] and is used here under the *CC BY SA 3.0*[2] license. The work *How you know your Free or Open Source Software Project is doomed to FAIL (or at least, held back fro])m success)*[3] originally appeared at this URL:)

http://spot.livejournal.com/308370.html

This was inspired by my recent efforts to look at *Chromium*[4], but these are just some of the red flags I generally have observed over the years written down.

Size

- The source code is more than 100 MB. [+5 points of FAIL]
- If the source code also exceeds 100 MB when it is compressed [+5 points of FAIL]

Source Control

- There is no publicly available source control (e.g. cvs, svn, bzr, git) [+10 points of FAIL]
- There is publicly available source control, but:
- There is no web viewer for it [+5 points of FAIL]
- There is no documentation on how to use it for new users [+5 points of FAIL]
- You've written your own source control for this code [+30 points of FAIL]
- You don't actually use the existing source control [+50 points of FAIL]

Building From Source

- There is no documentation on how to build from source [+20 points of FAIL]
- If documentation exists on how to build from source, but it doesn't work [+10 points of FAIL]
- Your source is configured with a handwritten shell script [+10 points of FAIL]
- Your source is configured editing flat text config files [+20 points of FAIL]
- Your source is configured by editing code header files manually [+30 points of FAIL]
- Your source isn't configurable [+50 points of FAIL]
- Your source builds using something that isn't GNU Make [+10 points of FAIL]
- Your source only builds with third-party proprietary build tools [+50 points of FAIL]

[1] http://fedoraproject.org/wiki/User:Spot
[2] http://creativecommons.org/licenses/by-sa/3.0/
[3] http://spot.livejournal.com/308370.html
[4] http://code.google.com/chromium/

- You've written your own build tool for this code [+100 points of FAIL]

Bundling

- Your source only comes with other code projects that it depends on [+20 points of FAIL]
- If your source code cannot be built without first building the bundled code bits [+10 points of FAIL]
- If you have modified those other bundled code bits [+40 points of FAIL]

Libraries

- Your code only builds static libraries [+20 points of FAIL]
- Your code can build shared libraries, but only unversioned ones [+20 points of FAIL]
- Your source does not try to use system libraries if present [+20 points of FAIL]

System Install

- Your code tries to install into /opt or /usr/local [+10 points of FAIL]
- Your code has no "make install" [+20 points of FAIL]
- Your code doesn't work outside of the source directory [+30 points of FAIL]

Code Oddities

- Your code uses Windows line breaks ("DOS format" files) [+5 points of FAIL]
- Your code depends on specific compiler feature functionality [+20 points of FAIL]
- Your code depends on specific compiler bugs [+50 points of FAIL]
- Your code depends on Microsoft Visual Anything [+100 points of FAIL]

Communication

- Your project does not announce releases on a mailing list [+5 points of FAIL]
- Your project does not have a mailing list [+10 points of FAIL]
- Your project does not have a bug tracker [+20 points of FAIL]
- Your project does not have a website [+50 points of FAIL]
- Your project is sourceforge vaporware [+100 points of FAIL]

Releases

- Your project does not do sanely versioned releases (Major, Minor) [+10 points of FAIL]
- Your project does not do versioned releases [+20 points of FAIL]
- Your project does not do releases [+50 points of FAIL]
- Your project only does releases as attachments in web forum posts [+100 points of FAIL]

- Your releases are only in .zip format [+5 points of FAIL]
- Your releases are only in OSX .zip format [+10 points of FAIL]
- Your releases are only in .rar format [+20 points of FAIL]
- Your releases are only in .arj format [+50 points of FAIL]
- Your releases are only in an encapsulation format that you invented. [+100 points of FAIL]
- Your release does not unpack into a versioned top-level directory (e.g. glibc-2.4.2/) [+10 points of FAIL]
- Your release does not unpack into a top-level directory (e.g. glibc/) [+25 points of FAIL]
- Your release unpacks into an absurd number of directories (e.g. home/johndoe/glibc-svn/tarball/glibc/src/) [+50 points of FAIL]

History

- Your code is a fork of another project [+10 points of FAIL]
- Your primary developers were not involved with the parent project [+50 points of FAIL]
- Until open sourcing it, your code was proprietary for:
- 1-2 years [+10 points of FAIL]
- 3-5 years [+20 points of FAIL]
- 6-10 years [+30 points of FAIL]
- 10+ years [+50 points of FAIL]

Licensing

- Your code does not have per-file licensing [+10 points of FAIL]
- Your code contains inherent license incompatibilities [+20 points of FAIL]
- Your code does not have any notice of licensing intent [+30 points of FAIL]
- Your code doesn't include a copy of the license text [+50 points of FAIL]
- Your code doesn't have a license [+100 points of FAIL]

Documentation

- Your code doesn't have a changelog [+10 points of FAIL]
- Your code doesn't have any documentation [+20 points of FAIL]
- Your website doesn't have any documentation [+30 points of FAIL]

FAIL METER

- 0 points of FAIL: Perfect! All signs point to success!

- 5-25 points of FAIL: You're probably doing okay, but you could be better.
- 30-60 points of FAIL: Babies cry when your code is downloaded
- 65-90 points of FAIL: Kittens die when your code is downloaded
- 95-130 points of FAIL: HONK HONK. THE FAILBOAT HAS ARRIVED!
- 135+ points of FAIL: So much fail, your code should have its own reality TV show.

Appendix

Content in this section is supportive materials that does not belong in another section or chapter.

Example needed.

Index