Software Engineering Topics for Community Driven Projects

Matthew Monaco

University of Colorado at Boulder
matthew.monaco@colorado.edu

Friday, March 23rd, 2012

Outline

1. Main Topics Overview
2. Development Models
3. Legal
4. Hosted Solutions
5. Notable Projects
6. Development Environments
7. Summary
8. References

Outline

1. Main Topics Overview
2. Development Models
3. Legal
4. Hosted Solutions
5. Notable Projects
6. Development Environments
7. Summary
8. References

Overview

- Community driven projects imply open source software
- Because they are inherently transparent, there are projects which exemplify almost any software engineering style conceivable
- Therefore, they are a *great* learning tool
- New projects can emulate ones the participants find interesting, or have even taken part in before
- Mistakes are also typically public, so they may be learned from

Status

- Community-driven projects have a wide range of characteristics
- Status
  - Maintenance — Feature complete or a more modern alternative exists.
  - Fork — Some parties had a different idea about the direction of the original project.
  - Immature — Testing new ideas or attempting to compete with an (inferior) project
  - Stable — Widely used but still accruing new features
  - Dead — May still be used, may receive the occasional patch

Focus

- Focus and Motivation
  - Security — For some projects, security is the modus operandi. (Those in the OpenBSD community, for example).
  - Improvements — New ways to look at problems with well-defined solutions. For example, distributed version control systems over the older, centralized paradigm.
  - Feature Rich — Some projects attempt to do a lot, and tend to accept new features quite willingly. Office suites are a good example of this.
  - Singular Focus — On the other hand, some projects have a very specific goal and most changes tend to be bug fixes, enhancements to *existing* features, or changes required to stay relevant (new authentication mechanisms).
  - Education — Many projects simply start as a desire to learn, one of the most famous being the Linux kernel.
  - Developer Needs — A lot of projects start as tools to do a job for which none exists.

Organization

- Pure democracy — Anyone may contribute, no real owner
- Dictatorship — A single project owner who takes contributions at his or her sole discretion (others may fork if the license permits)
- Federated — A single owner or small group of owners who delegate responsibility for subcomponents to others (the Linux kernel)
- Mixed — Because many projects are distributed in nature, there are typically isolated development groups. For example IBM, Oracle, Apple, Intel, and individual users all have their own private organization around X.Org development and their contributions are then adopted by a single maintainer

Testing

- The testing requirements for a community project vary greatly
  - Even within the same project!
- Often, projects which provide device drivers are highly dependant on hardware for testing. But volunteers don't have access to every variation
- In such situations, new versions are released publicly knowing that even widely used hardware may not have been tested. Because of this, short release cycles are becoming very popular
- Individual corporations may have varying standards for the contributions that they produce and contribute to a project
- Some projects provide unit tests with a release but these often go overlooked

Release Cycles

- As we'll see soon, there is a very wide range of release cycles for community projects
- These range from permanent development status (must build from VCS)
- To highly stable projects with releases only needed to address bugs and conform to changing technology
- To rapidly released software
- This last one is often the most effective for community projects because it means developers quickly get feedback for their work
- Additionally, rapid release cycles mean that features are released in relative isolation so regressions are easier to track down

Outline

1. Main Topics Overview
2. Development Models
3. Legal
4. Hosted Solutions
5. Notable Projects
6. Development Environments
7. Summary
8. References

Centralized VCS

- The version control system a project uses has the potential to play a large role in how the project organizes itself
- RCS is one of the original revision control systems. It was primarily meant for use on single files (no collaboration)
- CVS is a structured extension of RCS which allows collaboration via a CVS server. Users must *check out* files they want to work on.
- SVN is another centralized revision control system which is quite popular. Like CVS, users rely on a centralized server

Distributed VCS

- Distributed version control systems allow a lot more flexibility for most projects, especially community-based projects
- Developers can be much more independent
- They may share code in small (even private) groups while working on tasks, without permanently creating revision history
- There is no single point of failure (unless there is only a single copy of a repository)

Git

- Git [27] is a distributed revision control system initially
- Developed by Linus Torvalds in 2005
- Motivation — Free use of BitKeeper was withdrawn by the product's owner
- Goals — Speed. The Linux kernel is very large so day-to-day operations would be tedious if too slow
- Later Goals — Ease of use. Since git has become so refined and polished, it has seen an *enormous* adoption rate [20]

Git *continued*

- Git clearly does a lot right
- It has lead to much faster code development because it cuts down on collaboration overhead
- Through Git, it is very easy to isolate and test individual feature changes
- Bugs can be found quickly with `git bisect`
- Projects can easily fork one another if different philosophies emerge
- While certainly not a "silver bullet," Git (and some other distributed VCSs) have done wonders for software engineering

Git *continued*

- One reason for such a high adoption rate for git is that it doesn't require a project to change its organizational structure
- It can emulate the workflow used by any other VCS
  - Centralized — a master repository can act like an SVN or CVS server
  - Individual — git is useful even for personal work because no server setup is required
  - Hierarchical — (see the Linux discussion below)
- Once project groups are more comfortable with git, they can begin to ease their way into more efficient workflows
- Git proves that inertia is a major component of a software engineering project
- It began as a difficult to use tool, but as interest grew, enthusiasm has led to a very polished VCS interface

Outline

1. Main Topics Overview
2. Development Models
3. Legal
4. Hosted Solutions
5. Notable Projects
6. Development Environments
7. Summary
8. References

A project's contributors, life-cycle, and proliferation can be greatly affected by the license chosen. Let's take a look at the common ones and what they mean for a project.

Glossary

- There are a few terms in the software world which are misunderstood or used interchangeably when they shouldn't
- Open Source — The source code of a project is publicly available
  - No indication of license
  - No indication of cost
- Gratis — Free to use, often referred to as "free as in beer" or freeware. Gratis software may be installed by individuals but not redistributed or modified
- Libre — Free of restrictions, often referred to as "free as in speech." These projects are often released under fairly permissive licenses and are free to be redistributed, modified, etc

Glossary *continued*

- Copyleft — Redistribution and modification allowed, but must continue to be released under the same (or compatible) licenses. (This is a play on 'copyright')
- Copyfree — Very permissive, may be modified and redistributed under a new license
- GPL — A copyleft license
- LGPL — Similar to GPL, but allows projects to be linked with incompatible licenses
- MIT — A copyfree license

GPL

- Written by the Free Software Foundation [14]
- GPLv1 published in 1989 [14]
  - Copyleft
  - Redistributed software may be released under the same or even more permissive licenses
- GPLv2 published in 1991 [15]
  - If for some reason a project is not allowed to distribute under the GPL, then they may not redistribute GPL components at all.
  - Accompanied by the LGPLv2 (*see below*)
- GPLv3 published in 2007 [17]
  - Revised definition of source code
  - Disallowed hardware restrictions to override software restrictions
  - Nullifies DRM (*digital rights management*)

LGPL

- The Lesser GPL licenses LGPLv2 [16] and LGPLv3 [18] are companions to the GPLv2 [15] and GPLv3 [17] licenses.
- The GPL licenses stipulate that
  - A GPL'd work may be modified in private
  - If released, a derived project must be under a compatible license
- But "derived" is somewhat ambiguous
- The FSF holds that even dynamically linking to a GPL'd shared library is a derived work.
- The FSF released the LGPL licenses to allow a project with a non-free license to link to one under the LGPL
- However, the FSF still strongly encourages the use of the GPL
- Also referred to as the *Library* GPL

GPL *continued*

- The FSF is focused on the widespread use and development for free and open source software
- This sometimes makes it difficult for interaction with proprietary project
- The Linux Kernel
  - Released under the GPLv2 [15]
  - A monolithic software project — a (deliberately [9]) unstable internal API and ABI
  - Much easier to keep code in the main kernel tree because the person responsible for an API change is also responsible for fixes to all of its uses
  - But in-tree modules must be licensed under the GPL (or similar)
  - To get around this some companies release...
    - Small in-tree kernel module to do the bare minimum
    - The rest of the driver sits in userspace (typically at a performance loss)

GPL *continued*

- Alternatively, some projects provide an out of tree kernel module
- This means the driver doesn't come with Linux out-of-the-box, but does reside in the kernel once installed
- Nvidia, for example
  - Graphics driver is out-of-tree
  - Support for old hardware gets dropped frequently because of the maintenance overhead
- Additionally, out-of-tree and non-GPL kernel modules have a limited in-kernel API. Some of the most useful features of the kernel require a module to be released under the GPL

Apache

- The Apache license [13] is a GPL-compatible license used by projects under the Apache Software Foundation
- Most notably, the Apache HTTP server
- Unlike the GPL, this license is not copyleft
- If a project is released with GPL and Apache components, the project as a whole must be released under the GPL (becase it is stricter about freedom)

MIT

- The MIT license [21] is a GPL-compatible license
- Originally published by The Massachusetts Institute of Technology
- Copyfree
- Very short (relieving!)
- Simply states that the software is released without warranty, outlines what a user may do with the software, and stipulates that redistributing MIT-licensed software must include the license itself
- (But derived works do not have to be MIT themselves)

BSD

- The final widespread license we'll discuss is the BSD license [23]
- Originally published by The Regents of the University of California
- There are actually a few variations
- The most important and common restriction is that redistributing BSD licensed software must contain the original copyright notice

Organizations

- Releasing software under a free license isn't just a philosophy, it affects how software projects are actually developed
- Because many community driven projects aren't owned by any one person, and are developed by volunteers, a number of organizations have emerged to help protect the rights of developers
- The Free Software Foundation (FSF) [25] was started by Richard Stallman in 1985.
  - Primary purpose is to promote the use a free software
  - Gives endorsement to licenses which are intended to be "free"

Organizations *continued*

- The Open Source Initiative is similar to the FSF
  - Formed in 1998 by Bruce Perens and Eric Raymond
  - Approves licenses (does not provide its own)
  - Basically the FSF with a more palatable image to some
- These, and other organizations, maintain funding to help defend small projects against legal pressure by large, well-funded, organizations

Licencing *continued*

- Another interesting example is AMD (formerly ATI)
- AMD provides a proprietary driver for their graphics cards known as Catalyst [11]
  - Very good performance
  - Bad interoperability with common desktop technologies
  - Support for updated software lags behind releases (such as X.Org)
- AMD also contributes to and fully supports an open source driver for their graphics hardware known as xf86-video-ati [12]
  - Decent performance
  - Works very well on most Linux computers
  - Support for new hardware and updated dependencies is bleeding edge
- Companies need to figure out what is best for their customers. Sometimes this means providing alternatives

Outline

1. Main Topics Overview
2. Development Models
3. Legal
4. Hosted Solutions
5. Notable Projects
6. Development Environments
7. Summary
8. References

Online Platforms

- There are quite a few online platforms which combine essential features of any product into easy-to-use packages
  - Features — Bug tracker, mailing list, public source code repository, public release archives, wikis, web pages, question and answer services, and build services
  - Examples — Sourceforge, Launchpad, GNU Savannah, Github, BerliOS, BitBucket
- The most flexible of these allow for the use of most revision control systems
- They are at the heart of many projects because the foster very easy and rapid communication with quick feedback from users and testers

Bug Trackers

- The most widely used is Bugzilla [2]
- Bug trackers are a flexible way for projects to
  - Track planned features
  - Build dependency lists and prioritize changes
  - Communicate with users about regressions
  - Communicate with users about feature requests
- There are hosted Bugzilla services which even allow projects to place bug dependencies on other projects

OpenSUSE Build System

- Some community projects employ automatic build systems
- However, they typically have some sort of corporate backing to sponsor the infrastructure
- A good example of this is the OpenSUSE Build System [7]
- Packages are built automatically for all specified architectures as they are updated
- Package owners get feedback about failures as a result of changes very quickly
- Users can track where the distribution is headed and prepare their own projects ahead of releases

Outline

1. Main Topics Overview
2. Development Models
3. Legal
4. Hosted Solutions
5. Notable Projects
6. Development Environments
7. Summary
8. References

Linux

- Initial Motivation — Education, A free UNIX-like alternative [26]
- License — GPLv2 [15]
- Contributors — Thousands, from independent hobbyists to large corporations
- Organizational Structure — Federated [6], Linus Torvalds provides the main repository. He receives changes from the maintainers of the major subsystems, who in turn correspond with multiple sub-maintainers (and so on).
- Release Cycle
  - A release candidate around once per week
  - A full release every two to three months (depending on the number of release candidates — subject to Linus' discretion)
  - Bug fixes to stable versions for a few months
  - Every few releases are tagged "long-term" and will be supported for a few years

*BSD

- A number of modern variations (FreeBSD, OpenBSD, NetBSD,...)
- Motivation — Necessity. BSD dates back to the 1970's when modern UNIX systems were just emerging
- License — BSD
- Contributors — Many, FreeBSD, for example [4]
- Release Schedule — Slow. Free and OpenBSD are meant to be extremely stable. New features and new hardware support takes a long time as everything must be thoroughly tested

Firefox

- Developed by the Mozilla Corporation
- Motivation — Originally to provide a light-weight, standards-compliant, open source web browser
- License — MPL (a GPL-like license customized by Mozilla)
- Contributors — Many, but mostly under the umbrella of the Mozilla Corporation
- Organization — A number of "steering" committees and elected board members
- Release Schedule — Monthly [24]. In 2011 Firefox made a somewhat inauspicious switch to a rapid release cycle. Many claim this was a move to compete with Google Chrome's version number.
- However, from a software engineering standpoint, short release cycles allow Mozilla to get feedback and fix bugs on new features more easily
- Otherwise, it may not be obvious which new work actually caused the bug

Linux Distributions

- A Linux distribution can be an enormous software engineering undertaking in-and-of itself!
- Most distributions need to coordinate compatible versions of thousands of independent projects
- Some also provide rolling releases in which the system is meant to be updated frequently
- Let's look at one fixed release cycle distribution and one rolling release distribution...

Ubuntu

- Ubuntu is a fork of Debain by Canonical, owned by Mark Shuttleworth
- Motivation — Provide a free, open source Linux distribution for desktops with the goal of easy-of-use and predictable release schedules
- License (for distribution-specific projects) — Mostly GPL
- Contributors — Employees of Canonical
- Business Model — Free product, paid support
- Release Schedule — Six Months, but users may upgrade some components on their own
- Ubuntu is somewhat "introverted." It has been criticized for not trying to push contributions back to upstream projects
- However from an SE standpoint, this may not always be practical as Ubuntu's end goal is its *own* users

Arch Linux

- A rolling release distribution [1]
- Users are expected to have the latest of everything
- Attempting to hold on to a particular version of a project my require a lot of manual intervention
- Highly transparent, avoids making customizations to upstream projects
- A lot of work is required to release widely used shared libraries
- So as a software engineering project, Arch is mainly responsible for package management and system initialization
- I feel compelled to note: my personal favorite

X.Org

- The X.Org server is an implementation of the X11 protocol, which has its roots back to the 70's
- The X.Org project itself began in 1984
- The codebase is quite large and most changes are done with backward compatibility in mind (an enormous accidental difficulty)
- X.Org is managed by an elected board of directors.
- Most contributions come from Intel, Apple, Red Hat, Oracle, etc
- Recently began a much quicker release cycle

X.Org *continued*

- Another recent development for X.Org was the move to a modular architecture [10].
- This was done to help updates to individual components
- Previously, to test a new feature for a video card or input device, one would need to recompile the entire X.Org stack
- This has allowed for a faster release cycle in part because average users can more easily test new features and provide feedback

Openbox

- Openbox [19] is a window manager which works on its own or as a replacement inside common desktop environments
- Motivation — A rewrite and improvement of Blackbox
- License — GPLv2 [15]
- Release Cycle — Slow. Openbox is primarily developed by a single developer, in his spare time. So releases occur sporadically
- Conformity — Openbox needs to conform to many standards, the largest of these are X libraries, and Freedesktop.org specifications so that it will work well in other desktop environments

Outline

1. Main Topics Overview
2. Development Models
3. Legal
4. Hosted Solutions
5. Notable Projects
6. Development Environments
7. Summary
8. References

- Development environments are an import aspect to software development
- Well designed ones should have no bearing on how a project is actually managed
- A group member *should* be able to choose whatever he or she is comfortable with
- We'll briefly look at the most popular ones

Vim

- Vim [22] is an improved version of the ancient vi
- It aids software development most by staying out of the user's way
- Provides very efficient mechanisms for editing and managing text files
- Extensible [8] — Vim can be extended in many ways, which has greatly contributed to its staying power

Emacs

- Like Vim, Emacs [5] is widely used
- It offers its users a little bit of a different approach...
- Emacs is quite heavy-weight but is highly extensible
- It can be used as a simple text editor, or a full-blown IDE

Eclipse

- Eclipse [3] is a complete integrated development environment
- Primarily geared toward Java
- Eclipse tries to do it all for the user (rather than the do one thing and do it right approach)
- Like Vim and Emacs it is also highly extensible
- It provides features such as
  - API reference
  - Built-in build systems
  - VCS integration
  - Debugging capabilities

Outline

1. Main Topics Overview
2. Development Models
3. Legal
4. Hosted Solutions
5. Notable Projects
6. Development Environments
7. Summary
8. References

Summary

- Following community and open source projects is a great learning tool
- When beginning a new project, chances are there is an existing project out there with a model that makes sense to adopt
- This includes learning from others' mistakes
- The information required to learn from a community project is almost always readily available through mailing list archives, bug trackers, and VCS history

Outline

1. Main Topics Overview
2. Development Models
3. Legal
4. Hosted Solutions
5. Notable Projects
6. Development Environments
7. Summary
8. References

References I


Arch linux.
http://www.archlinux.org.


Bugzilla.
http://www.bugzilla.org.


Eclipse.
http://www.eclipse.org/.


Freebsd.
http://www.freebsd.org.


Gnu emacs.
http://www.gnu.org/software/emacs.

References II

Linux maintainers.
`http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=blob;f=MAINTAINERS;hb=master`.

Opensuse build service.
`https://build.opensuse.org`.

Vim scripts.
`http://www.vim.org/scripts/index.php`.

Can we move device drivers into user-space?
`https://lkml.org/lkml/2012/2/22/569`, February 2012.

References III


Paul Anderson, Alan Coopersmith, Egber Eich, Adam Jackson, Kevin Martin, and Keith Packard.
X.org modularization proposal.
http://www.x.org/wiki/ModularizationProposal.


Advanced Micro Devices.
Catalyst display driver.
http://www.amd.com/us/products/technologies/amd-catalyst/pages/catalyst.aspx.


Advanced Micro Devices and Community.
xf86-input-ati.
http://cgit.freedesktop.org/xorg/driver/xf86-video-ati/.

References IV


Apache Software Foundation.
Apache license v2.0.
http://apache.org/licenses/LICENSE-2.0, January 2004.


The Free Software Foundation.
Gnu general public license v1.0.
http://www.gnu.org/licenses/gpl-1.0.html, February 1989.


The Free Software Foundation.
Gnu general public license v2.0.
http://www.gnu.org/licenses/gpl-2.0.html, June 1991.


The Free Software Foundation.
Gnu lesser general public license v2.1.
http://www.gnu.org/licenses/lgpl-2.1.html, February 1999.

References V

The Free Software Foundation.
Gnu general public license v3.0.
http://www.gnu.org/licenses/gpl-3.0.html, June 2007.

The Free Software Foundation.
Gnu lesser general public license v3.0.
http://www.gnu.org/licenses/lgpl-3.0.html, June 2007.

Dana Jansens.
Openbox.
urlhttp://www.openbox.org.

References VI

Paul Krill.
Torvald's git: The 'it' technology for software version control.
urlhttp://www.infoworld.com/d/application-development/torvalds-git-the-it-technology-software-version-control-167799, July 2011.

MIT.
The mit license (mit).
http://www.opensource.org/licenses/MIT, 1988.

Bram Moolenaar.
Vim.
http://www.vim.org.

References VII

Regents of the University of California.
Bsd license.
http://www.opensource.org/licenses/BSD-2-Clause.

The Mozilla Project.
New channels for firefox rapid releases.
http://blog.mozilla.com/blog/2011/04/13/new-channels-for-firefox-rapid-releases/, April 2011.

Richard Stallman.
The free software foundation.
http://www.fsf.org, 1985.

References VIII


Linus Torvalds.
What would you like to see most in minix?
http://groups.google.com/group/comp.os.minix/msg/b813d52cbc5a044b, August 1991.


Linus Torvalds, Junio Hamano, and et al.
Git - fast version control system.
http://git-scm.com, April 2005.