Two case studies of Open Source Software Development: Apache and Mozilla

Audris Mockus, Roy Fielding, and James D Herbsleb

Presented by Jingyue Li

NTNU
Innovation and Creativity

Outline

- Research questions
- Research methods
- Data collection
- Results
- Conclusions/Hypotheses generated
- Final discussions

Research questions$_1$

- *Q1: What was the process used to develop OSS?*
- *Q2: How many people wrote code for new OSS functionality? How many people reported problems? How many people repaired defects?*
- *Q3: Were these functions carried out by distinct groups of people, i.e., did people primarily assume a single role? Did large numbers of people participate somewhat equally in these activities, or did a small number of people do most of the work?*
- *Q4: Where did the code contributors work in the code? Was strict code ownership enforced on a file or module level?*

Research questions$_2$

- *Q5: What is the defect density of OSS code?*
- *Q6: How long did it take to resolve problems? Were higher priority problems resolved faster than low priority problems? Has resolution interval decreased over time?*

Research methods

- First studied the Apache project. Based on the results, a number of hypotheses were framed.
- The second study began after the analyses and hypothesis formations were completed.
- In the second study, the comparable data from Mozilla project were examined.
- Data from Mozilla were used to support for or revise the original hypotheses.
- In both studies, data from five commercial systems were used as comparisons.

Data collection and data analysis

- Apache
  - Developer Email list
  - CVS (Concurrent version control archive), including modification request and modification trace
  - Problem reporting database
- Mozilla
  - Developer Email list
  - CVS
  - Bugzilla
- Commercial project
  - Data from source code control system
  - Data from extended change management system

Results – Apache$_1$

- *Q1: What was the process used to develop Apache?*

  Q1.1 Roles and Responsibilities

  – The Apache Group (AG), the informal organization of core people (part-time work, volunteer) responsible for guiding the development of the Apache

  – AG used email lists exclusively to communicate with each other, and a minimal quorum voting system for resolving conflicts.

  – AG members are people who have contributed for an extended period of time, usually more than 6 months, and are nominated for membership and then voted on by the existing members.

  – Started with 8, has 12 through most of the period, and 25 in 2002, probably more now

Results – Apache$_2$

- *Q1: What was the process used to develop Apache?*

  Q1.2 Identify work to be done

  – Change requests are reported on the developer mailing list, the problem reporting system (BUGDB), and the USENET newsgroups
    - Change requests on the mailing list are given the highest priority.
    - Only one or two developers perform periodic check of BUGDB.
    - Only a few developers ever have time to read news in USENET

  – To keep track of the project status, an agenda file (“STATUS”) is stored in each product’s repository, containing a list of high priority problems, open issues among the developers, and release plans.

  – For a proposed change actually to be made, an AG member must ultimately be persuaded it is needed or desirable.

Results – Apache$_3$

- *Q1: What was the process used to develop Apache?*

  Q1.3 Assigning and performing development work

  – Core developers tend to work on problems which they are most familiar.
  – The core developers obtain an implicit “code ownership” of parts of the server that they are known to have created or to have maintained consistently.
  – Apache software architecture is designed to separate the core functionality of the server from the features (located in modules that can be selectively compiled and configured).
  – New core developers tend to focus on areas without preexisting claims.
  – In many cases, the primary difficulty of identifying a solution at this stage is not finding a solution, but to decide the most appropriate one.

Results – Apache$_4$

- *Q1: What was the process used to develop Apache?*

  Q1.4 Prerelease testing

  – Once a solution has been identified, the developer makes changes to a local copy of the source code and tests the changes on his or her own server.

  – This level of testing is more or less comparable to unit test.

  – No additional testing (e.g., regression, system test) required prior to release, although review is required before or after committing the change

Results – Apache$_5$

- *Q1: What was the process used to develop Apache?*

  Q1.5 Inspections

  – After unit testing, the core developer either commits the changes directly or produces a “patch” and posts it to the developer mailing list for review.
    - Changes to a stable release require review before being committed
    - Changes to development releases are reviewed after the change is committed.

  – If approved, the patch can be committed to the source by any of the developers.

Results – Apache$_6$

- *Q1: What was the process used to develop Apache?*

  Q1.6 Managing releases

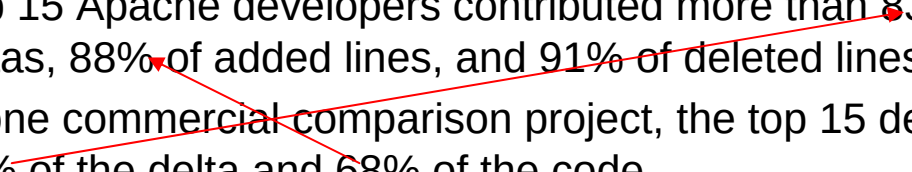
  – When the project nears a product release, one of the core developers volunteers to be the release manager, responsible for:
    - Identifying the critical problems that prevent the release,
    - Determining when those problems have been repaired and the software has reached a stable point
    - Controlling access to the repository so that developers don't inadvertently change things that should not be changed just prior to the release.

Results – Apache$_7$

- *Q2: How many people wrote code for new Apache functionality?*
  - 249 people contributed to 6,092 code submissions
- *How many people reported problems?*
  - Around 3,060 different people submitted 3,975 problem reports,
- *How many people repaired defects?*
  - 182 people contributed to 695 fixes,

Results – Apache[8]

- *Q3: Did people primarily assume a single role? Did large numbers of people participate somewhat equally in these activities, or did a small number of people do most of the work?*
  - Top 15 Apache developers contributed more than 83% of the MRs and deltas, 88% of added lines, and 91% of deleted lines.
  - In one commercial comparison project, the top 15 developers contributed 77% of the delta and 68% of the code.
  - Participation of the wider development community is more significant in defect repair than in the development of new functionality.
  - Wide community of Apache users are the major system tester and problem reporters.



NTNU
Innovation and Creativity

Results – Apache$_9$

- *Q4: Where did the code contributors work in the code? Was strict code ownership enforced on a file or module level?*
  - Out of 42 ".c" files with more than 30 changes, 40 had at least two (and 20 had at least four) developers making more than 10% of the changes.
  - Rather than any single individual writing all the code for a given module, those in the core group have a sufficient level of mutual trust that they contribute code to various modules as needed.

Results – Apache$_{10}$

- *Q5: What is the defect density of Apache code?*

  Comparing to the commercial counterparts:

  – the user-perceived (post-release) defect density of the Apache product is inferior to that of the commercial products

  – the defect density of the Apache code before system test (pre-lease) is much lower.

Results – Apache$_{11}$

- *Q6: How long did it take to resolve problems? Were higher priority problems resolved faster than low priority problems? Has resolution interval decreased over time?*
  - Fifty percent of PRs are resolved within a day, 75% within 42 days, and 90% within 140 days.
  - Core” and “Most Sites” PRs have much faster close times than “OS” and “Major Optional”
  - Important aspect of customer support improved over time, despite the dramatic increase in the number of users.

Mozilla vs. Apache

- Has a process with commercial roots.
- The work in the Mozilla project is much more diverse: it supports many technologies including development tools (CVS, Bugzilla, Bonsai, Tinderbox) that are not part of the Web browser.
- Keep in mind, therefore, that very different results of Mozilla and Apache might be obtained from different hybridization strategies.

Results – Mozilla$_1$

- *Q1: What was the process used to develop Mozilla?*

  Q1.1 Roles and Responsibilities

  – Mozilla is operated by the mozilla.org staff.
  – Only about 4 (out of 12) of the core members spend a significant part of their time writing code. Others have roles dedicated to such things as community QA, milestone releases, and tools support.
  – Some external people (e.g., from Sun Microsystems) are working full-time, for pay, on the project.
  – Adding a new module requires the permission of mozilla.org.
  – Mozilla.org has the ultimate decision-making authority, and retains the right to designate and remove module owners, and to resolve all conflicts that arise.

Results – $Mozilla_2$

- *Q1: What was the process used to develop Mozilla?*

  Q1.2 Identify work to be done

  – Mozilla.org maintains a roadmap document that specifies what will be included and release schedule. However, time is reserved for community members to comment on it.

  – Use Bugzilla to report and manage bugs and enhancement requests.

Results – Mozilla$_3$

- *Q1: What was the process used to develop Mozilla?*

  Q1.3 Assigning and performing development work

  – The mozilla.org members who write browser code appear to focus on areas where they have expertise and where work is most needed to support upcoming releases.

  – Developers can use Bugzilla to request help on a particular change, and to submit their code.

Results – Mozilla$_4$

- *Q1: What was the process used to develop Mozilla?*

  Q1.4 Prerelease testing

  – Mozilla.org performs a daily build, and runs a daily minimal “smoke test” on the build for several major platforms, in order to ensure the build is sufficiently stable to allow development work on it to proceed.

  – If the smoke test identifies bugs, they are posted daily so that developers are aware of any serious problems in the build.

  – Mozilla currently has six product area test teams that take responsibility for testing various parts or aspects of the product, such as standards compliance, mail/news client, and internationalization.

  – The test teams maintain test cases and test plans, and other materials such as guidelines for verifying bugs and troubleshooting guides.

Results – Mozilla$_5$

- *Q1: What was the process used to develop Mozilla?*

  Q1.5 Inspections

  – Mozilla uses two stages of code inspections:
    - By module owners who review a patch in the context of the module
    - By a smaller designated group (referred to as super-reviewers) who review a patch for its interaction with the code base as a whole before it is checked in.

Results – $Mozilla_6$

- *Q1: What was the process used to develop Mozilla?*

  Q1.6 Managing releases

  – Mozilla runs a continuous build process
  – Produces binaries nightly and issues “Milestones” approximately monthly.
  – Milestone decisions are made by a designated group, known as drivers@mozilla.org, with input from the community.

Results – Mozilla$_7$

- *Q2: How many people wrote code for new Mozilla functionality? How many people reported problems? How many people repaired defects?*
  - Much larger than Apache
  - 6,837 people reported about 58,000 PRs, and 1,403 people reported 11,616 PRs that can be traced to changes to the code.
  - Outside participants tend, on average, to contribute fewer changes and less code relative to internal participants.
  - Much larger external participation may be found in problem reporting. About 95% of the 6,873 people who created PRs were external, and they reported 53% of the 58,000 PRs.

Results – Mozilla$_8$

- *Q3: Did people primarily assume a single role? Did large numbers of people participate somewhat equally in these activities, or did a small number of people do most of the work?*
  - Mozilla development had much larger core groups relative to the total number of participants.
  - The problem reporting participation was very uniform in Apache, but contributions vary substantially in Mozilla, with 50% of PRs reported by just 113 people, with the top person reporting over 1,000 PRs (compared to Apache, where the top reporter submitted only 32 PRs).

Results – Mozilla$_9$

- *Q4: Where did the code contributors work in the code? Was strict code ownership enforced on a file or module level?*
  - In apache, there is no clear ownership pattern. The owner of a specific code equals the person who contributed most to the code
  - In Mozilla, on the other hand, code ownership is enforced, called despot. The module owner is responsible for:
    - Fielding bug reports, enhancement requests, patch submissions, and so on.
    - The owner should facilitate good development community.
    - Before code is checked in to the CVS repository it must be reviewed by the appropriate module owner and possibly peers.

Results – $Mozilla_{10}$

- *Q5: What is the defect density of Mozilla code?*
  - *Pre-release defect density is lower than Apache and commercial ones*
  - *Post-release defect density is lower than commercial ones*

Results – $Mozilla_{11}$

- *Q6: How long did it take to resolve problems? Were higher priority problems resolved faster than low priority problems? Has resolution interval decreased over time?*
  - Median resolution interval is much longer than for Apache. It means that mandatory inspection of changes in Mozilla almost doubles the PR resolution interval.
  - There is a significant relationship between interval and priority. These results appear to indicate that Mozilla participants were generally sensitive to PR priority.

Hypotheses generated from the two case studies$_1$

- ***Hypothesis* 1a**: Open source developments will have a core of developers who control the code base, and will create approximately 80% or more of the new functionality. If this core group uses only informal ad hoc means of coordinating their work, the group will be no larger than 10 to 15 people

- ***Hypothesis* 2a**: If a project is so large that more than 10 to 15 people are required to complete 80% of the code in the desired time frame, then other mechanisms, rather than just informal ad hoc arrangements, will be required in order to coordinate the work. These mechanisms may include one or more of the following: explicit development processes, individual or group code ownership, and required inspections

Hypotheses generated from the two case studies$_2$

- ***Hypothesis* 3**: In successful open source developments, a group larger by an order of magnitude than the core will repair defects, and a yet larger group (by another order of magnitude) will report problems

- ***Hypothesis* 4**: Open source developments that have a strong core of developers but never achieve large numbers of contributors beyond that core will be able to create new functionality but will fail because of a lack of resources devoted to finding and repairing defects

Hypotheses generated from the two case studies$_3$

- ***Hypothesis* 5**: Defect density in open source releases will generally be lower than commercial code that has only been feature-tested, that is, received a comparable level of testing

- ***Hypothesis* 6**: In successful open source developments, the developers will also be users of the software

- ***Hypothesis* 7**: OSS developments exhibit very rapid responses to customer problems

Final discussions$_1$

- Two kinds of coordination
  - Explicit mechanisms include such things as interface specifications, processes, plans, staffing profiles, and reviews
  - Communication-only approach (does not scale)
- Apache adopts an approach to coordination that seems to work extremely well for a small project
  - The server itself is kept small
  - Any functionality beyond the basic server is added by means of various ancillary projects that interact with Apache only through Apache's well-defined interface

Final discussions$_2$

- Control over the interface is asymmetric, in that the external projects must generally be designed to what Apache provides
- The coordination concerns of Apache are thus sharply limited by the stable asymmetrically controlled interface
- Thus, coordination can be successfully handled by a small core team using primarily implicit mechanisms
- The benefit of the larger open source community for Apache is primarily in those areas where coordination is much less of an issue

Final discussions$_3$

- The Mozilla approach has some of the Apache-style OSS benefits:
  - The community has taken over a significant portion of the bug finding and fixing helping with these low-interdependency tasks
- However, the Mozilla modules are not as independent from one another as the Apache server is from its ancillary projects.
  - Because of the interdependence among modules, considerable effort (i.e., inspections) needs to be spent in order to ensure that the interdependencies do not cause problems
  - Therefore, the relatively free-wheeling Apache style of communication and implicit coordination is likely not feasible in Mozilla.
  - Mozilla core teams must have more formal means of coordinating their work

Final discussions$_4$

- Overall, in OSS projects, low post-release defect density and high productivity stem from effective use of the open source community for the low-interdependence bug finding and fixing tasks
- Defect density and productivity both seem to benefit from recruiting an open source community of testers and bug fixers

Final discussions$_5$

- How core team in Apache (and, we assume, many other OSS projects) is formed may be another of the keys to their success.
  - They will try to choose something that is both badly needed and where they have some specific interest
  - They must demonstrate a high level of capability
  - They must also convince the existing core team that they are responsible and productive
- Commercial development
  - Assignments are given out that may or may not correspond to a developer's interests or perceptions of what is needed

The power of gifts: organizing social relationships in open source communities

By Magnus Bergquist & Jan Ljungberg
The Viktoria Institute, Göteborg, Sweden

Open source communities

Empirical data collected over a period of 2 years

- Texts and papers
- Scanning news groups and discussions etc

Actors and Stakeholders

- Owner (founder, assigned or preserver)
- Core developers
- Developers
- Defect repair
- Tester (end users)
- User support (mundane)

Open source development

Exchange culture vs gift cultures

characterized by

Scarcity or abundance

Gifts

- The practice is culturally defined and socially determining activity
- ”Giving a gift brings forth a demand for returning a gift, either another object or, in a more symbolic fashion, forces of power connected to the objects”
- ”To give away something is to express an advantegeous position in relation to the recipient”

Gifts

- The Internet
- Reproduction cost ~$0
- Non-deminishing value
- The social character of gifts still doesn't disappear

Gifts vs Commodity transactions

- The giving of gifts should not involve explicit bargaining
- ”By giving away, the giver shows superiority and the receiver becomes dependent. There is an obligation to repay the gift in the future.”

Decision making

- Closed circle of developers?
- Who gets in?
- Flaming

Reputation and Status

Giving to the community is what makes the individual a hero in the eyes of others

Important:

- Leadership
- Common goals

Toward an Understanding of the Motivation of Open Source Software Developers

Yunwen Ye & Kouichi Kishida

Friday, 26. September, 2008

Overview

1. Introduction
2. OSS Communities
3. An Example – The GIMP Project
4. Legitimate Peripheral Participation
5. Learning as the Motivation
6. Discussions
7. Summary

OSS Communities

- Anyone can be a contributor
  - No strict seperation between users & developers
- Community of practice
  - Common goal/interest
- Regeneration
  - Original developers leave, new developers join

Roles

- Project Leader
- Core Member
- Active Developer
- Peripheral Developer
- Bug Fixer
- Bug Reporter
- Reader
- Passive User

GIMP

- Image editor
- Developed by two students until they graduated
- 20 month pause
- Leadership assumed by a different developer
- Enough core developers to take over the task after he left.

Mailing list activity

No. of emails	No. of members	Total no. of emails
> 200	5	2237
101 - 200	8	1197
51 - 100	10	695
26 - 50	29	1061
11 - 25	47	741
5 - 10	73	471
3 - 4	107	352
2	134	267
1	502	502
Total	915	7525

Code contributions

No. of contributions	No. of contributors	Breakdown of the contributors according to their defined roles in the GIMP community				
		Core Members	Active Developers	Peripheral Developers	Bug Fixers	Bug Reporters
>250	3	3	0	0	0	0
101-250	4	0	4	0	0	0
51-100	11	0	10	1	0	0
21-50	15	1	12	2	0	0
3-20	47	0	21	26	0	0
1-2	82	0	0	82	0	0
Not credited in change log	25	0	0	0	10	15
Total	**197**	**4**	**47**	**111**	**10**	**15**

Legitimate Peripheral Participation

- Knowing-in-action
- Learning by participation
- Peripherally participate by doing small and easy tasks
  - Legitimate participation, access to knowledge from masters

Learning as the Motivation

- «Scratching a personal itch»
  - Need new functionality
- Often not the case
- Is the motivation to learn something new?
- Creating a software system = knowledge cronstuction

OSS Initiators

- Explorative learning
  - Find new ways of things
  - Overcoming existing problems
  - Doing things better
- Learning by doing
  - Learn more about a topic by doing practical tasks

Later Participants

- OSS projects often initiated by skilled developers
- Good help in learning and understanding programming.
- Learn from fixing bugs, maintenance, ++
- Level of participation can increase as skill level increases

Social Aspects

- Technical supremacy highly appreciated
- Reputation
  - Attention
  - Trust
  - Influence

Control structure

- Open developement / Open release
- Process
  - Closed
  - Transparent
  - Open
- Different degree of participation possible

Building a community

- Focus on community building
- Encourage newcommers
- Be responsive to questions
- Make the project attractive
  - Language
  - Modularity

Why Open Source software can succeed

Andrea Bonaccorsi, Cristina Rossi [2003]

Laboratory of Economics and Management,
Sant'Anna School of Advanced Studies,
Italy

Presented by Gunnar Rangøy

Open Source from an economic view

- Open Source analysed as a process innovation.
  - A new and revolutionary process based on access to the source.
- Can be formulated in terms of Schumpeters's three-phase axis of innovative process
  - Invention
  - Innovation
  - diffusion

Theoretical puzzles for economic analysis

- Why do programmers write Open Source codes if no one pays them to do it?
- How do hundreds of people dotted around the world manage to effectively co-ordinate with each other in the absence of any hierarchical structure based on the ownership of assets, producing programmes consisting of millions of lines of code?
- Why is it that Open Source programmes are becoming so widespread in a world dominated by Microsoft-imposed standards, and more generally by proprietary standards?

Motivation

- “The aim of this paper is to explain what at first sight would seem “nonsense” by combining recent developments in economic analysis such as the theory of collective action and the theory of technological diffusion in the presence of network externalities.”

A revolutionary idea

- Stallmann proposed the Free Software Foundation in 1984
  - There should be unrestricted access to computer programming codes
- Philosophy
  - When programmers are allowed to work freely on the source the collaboration helps top correct errors and adoption to different needs and platforms.

“I don’t know who these crazy people are who want to write, read and even revise all that code without being paid anything for it at all”

Glass, R., 1999. Of Open Source, Linux and Hype.

Mass of users

- Heterogeneous
  - Large group of users incapable of developing
  - Hobby programmers
    - Limited contribution; insufficient to explain the enormous results achieved
  - «Hacker culture»
    - The heirs of the «Real Programmers»

Motivation

- Why work for free?
  - Altruism can at most explain people writing software in their spare time

Intellectual gratification

- Similar to a scientific discovery
  - Sharing results helps to improve them through feedback and to get recognition and prestige
  - Same thing applies to sharing code
- Involving other elements than financial remuneration

Creativity

- Hackers regard programming as an form of art
- Pleasure of creativity
  - Creativity is lost in the commercial world
    - Deadlines
    - Transformed into a assembly line

Reliable

- Commercial software perceived as not reliable
- “Given enough eyeballs, all bugs are shallow”. Raymond (1999)

- Provides prestige and visibility to be noticed by software firms
- Signalling of quality of human capital
- Projects arise to fill an unfilled market

Open Source movement

- Profound change in the nature of the software good
  - Assumed the characteristics of the collective good
- How is free riding avoided?
  - Incentive to contribute does not get smaller as the the number of contributors grows

Open source projects

- Definition

  Any group of people developing software and providing their results to the public under an Open Source license

- Successful projects usually starts with someone has a need for a particular kind of software
- Increase in participation allows the project to gain momentum

Open source projects [cont]

- The lifecycle of a successful is shaped by
  - A widely accepted leadership
  - Effective co-ordination mechanism
- Authority of project leaders arises naturally
  - Selects the best fit solution
  - Does not force participants to do any particular task
- If no one volunteers the job is neglected
  - If the job is fundamental the project is going to stop
- In successful projects tasks are done by several participants

Object orientation

- Object oriented is a innovation in managing complexity
  - Programs is organised in modules
  - Modules can be reused
- OO matches the Open Source production process
- The key to the success of Linux is its modularity (Torvalds)

Communication

- World-wide contributions
- Face to face and phone is rare
- Asynchronous
  - Email
  - Ftp
  - Newsgroups
- Storage of the produced knowledge

Conventions

- Common notion of calidy
- Each project has conventions
- Tacit rules

Licences

- Most important governance structure
- GPL copyleft licence
  - Viral nature
  - Maximizes circualtion
  - Can't hide the code

Non-sexy work

- Mundane but necessary tasks
  - Development of graphical interfaces
  - Compilation of technical manuals
- Difficult to explain motivation

Hybrid business models

- Companies produce free software
  - Charing additional services instead of licences
- Open source licences that do not follow the extreme GPL format

Diffusion

- Subject to path dependence and lock-in
- Users within group more important than the total number
- Advocacy has exponential growth
- Critical Mass

Simulations

- Model implemented using the SWARM-platform

$$F = \alpha \cdot (1 - \rho) \cdot IV + [(1 - \alpha) \cdot NE + \Delta \cdot CO]$$

- depends on initial variables
- «A robust general result is that under many plausible conditions commercial software and Open Source software are likely to coexist even in the limit»

Conclusion

One of the main implications of our analysis and of the simulation exercise is that under many plausible situations commercial software and Open Source are likely to coexist in the future.