Chapter 3: Analysis of FOSS projects

Josep M. Ribó

March 9, 2010
Chapter 3: Analysis of FOSS projects

3.1 Types of FOSS projects

3.2 Analysis of FOSS projects
   1. Project identity card
   2. Project analysis from the repository
   3. Project analysis with software tools
   4. Websites that analyze FOSS projects

3.3 Other types of analysis
   3.3.1 Apache
   3.3.2 Mozilla
   3.3.3 Conclusions of these two case studies
2.1 Types of FOSS projects

Traditionally, 5 kinds of FOSS projects are considered:

1. Solo
2. Monarchist
3. Community
4. Corporate
5. Foundation

**General criteria:**

- The bigger the number (1..5) the greater the formality
- If projects evolve, they evolve upwards
1.4.1 Types of FOSS projects. Solo projects

- **Description:**
  - 90% of all FOSS projects
  - 1-2 developers responsible for the whole projects
  - They are open
  - Sometimes they are forks of other projects
  - Support is not guaranteed (developers may lose interest or want to keep on the development by themselves)

- **Examples:** Plenty of examples in sourceforge
1.4.1 Types of FOSS projects. Monarchist projects

• **Description:**

  – They evolve from a solo project that has been successful and has gained a large community of contributors
  – Usually one project leaders and a few “lieutenants”
  – Decisions are taken by the project leader and his/her lieutenants
  – Disputes are settled by the project leader
  – More formal than solo projects. They use formal mail lists, bug tracker, version system

• **Examples:** Perl, Python, Bittorrent
1.4.1 Types of FOSS projects. Community projects

• **Description:**
  - The project is ruled by a significant number of contributors. Decisions are taken democratically or by consensus of those contributors (sometimes a committee does exist).
  - Contributions are thoroughly reviewed by the contributors.
  - Very active tools (mail lists, bug trackers...).
  - Communities focus on recruitment to advance the project as much as possible.

• **Examples:** PostgreSQL, Linux Terminal Server
1.4.1 Types of FOSS projects. Corporate projects

• **Description:**
  - Code that was released by a private company and still is intertwined with it
  - The majority of programmers are company employees
  - Strategic decisions about the project are taken by the company
  - Difficult to participate actively in a corporate project (formal process of application, assignment of rights to the company...)
  - External contributors tend to focus on add-ons to the project and bug finding
  - Dual licensing
    It’s also worth noting that Corporate projects are alone in having an organizational model which requires a particular license. Almost universally, they either use the GPL or LGPL (in order to compel contribute-back) or a Mozilla-like license that maximizes credit for the company.

• **Examples:** MySQL, Firebird, Red Hat Fedora, JBoss, OpenOffice...
1.4.1 Types of FOSS projects. Foundation projects

• Description:
  – A Foundation is a non-profit formal organization that supports the development of a FOSS project
  – Projects that are critical to various large companies who use the Foundation to protect their mutual interests and ensure themselves a voice.
  – Community projects that seek for the legal advantages of foundations and that need to hire staff
  – Decision-making process is completely formalized
  – Difficult to participate actively in a foundation project (formal process of application ... )
  – Systematic fundraising (user donations are encouraged)

• Examples:
  – Apache Software Foundation (Tomcat, Apache, Struts ...)
  – Uber-Foundation (SpamAssassin)
  – Mozilla Foundation
  – Gnome
  – THe GNU project
  – The Free Software Foundation
I propose three levels of FOSS project analysis:

1. Project identity card [BLM2008]

2. Project analysis using the information obtained from the repository (forge) [BLM2008]

3. Project analysis using software tools on the version control system [RGC2003]
3.2 Analysis of Foss projects. 1. Project id. card

Project identity card

It is composed by the essential attributes that constitute a FOSS project.

These are the attributes:

- Project name
- Repository
- Homepage: project URL
- Project description: What is the purpose of the project?
- License
- Standalone vs. Part of a project family
- Functional description:
  Software type (Web Server, Operative System, ERP, ...)
- Type of software organization that produces it
  Company, foundation, spontaneous ...
3.2 Analysis of Foss projects. 1. Project id. card

• **Cost for the users**
  
  – Entirely free,
  – Pay for services and features,
  – Pay for everything

• **Size of current development team** *(people who have written in the repository in the last 6 months)*
  
  – 0 (abandoned project),
  – 1 - 10,
  – 11-50,
  – more than 50

• **Community size**
  
  Small: less than 51, Medium: 51-250, Large: more than 250

• **Programming language**
3.2 Analysis of Foss projects. 1. Project id. card

- **Infrastructure/software engineering tools** (no. of categories covered from the following infrastructure list)
  - some use (0-4), extensive use (5-7)
  - continuous integration
  - build tools
  - automated code documentation
  - testing
  - version control
  - issue/bug tracking system
  - static code analysis

- **Innovation**
  - Traditional FOSS application (e.g., before 2003),
  - Emerging application contexts for FOSS, which there were only proprietary solutions before 2003 (e.g., Open Office)

- **Age**
  - Before 1998 (definition of OSS)
  - 1998 - 2003
  - Later than 2004

See examples at [BLM2008]
Project analysis using the project repository (forge)

Taking a look at the project repository, a lot of information about the project may be obtained. This information can be structured in the following topics:

- **Functional requirements**
  - Is there a detailed feature list?
  - Are there release notes?

- **Reliability**
  - Development status
  - Minor/patch releases
  - Critical bug number
  - Bug response time
  - Test process

- **Maintainability**
  - Coding standards, programming style guide (Yes/No)
  - Maintainance releases (Yes/No/ How many)

- **User community**
3.2 Analysis of Foss projects. 2. Using the project repository

- User mailing list (Yes/No/How many)
- Dimension of the user community
- Number of patches/releases
- Number of patches/releases last month

- **Contributor community**
  - Technical mailing list (Yes/No/How many)
  - Dimension of the contributor community
  - Contributors’ origin and status (employees of the company that develops the project, bazaar-like hackers ...)

- **Documentation**
  - Documentation mailing list (Yes/No)
  - User manual (Yes/No)
  - *Getting started* guide (Yes/No)
  - FAQ (Yes/No)
  - Technical documentation (Yes/No/Which: tech. manual, javadocs-like documents...)
  - Installation guide (Yes/No)

- **Distribution channel**
  - Source code download (Yes/No)
  - Binary code download (Yes/No)
  - Repository access (anonymous, read-only, read-write, impossible...)


3.2 Analysis of Foss projects. 2. Using the project repository

• **Project organization and contribution policy**

  – Who takes the decisions concerning the project evolution?
  – Which is the role of the community in this decision-taking process?
  – Is it easy to participate actively?
  – Does it provide a dual license?

See [BLM2008] for a complete reference (with more topics) and examples
3.2 Analysis of FOSS projects source code using software tools

The source code of FOSS projects provides a lot of information in order to analyze those projects.

There are several software tools that are meant to do such analysis:

- CODD
  It gets information about authorship.

- SLOCCount [SLOCCount]
  It counts the lines of code of the project and it applies to this result the COCOMO model

- CVSSAnalY [CVSSAnalY]
  It analyses information about committers, files and modules

In the following few slides we will introduce these applications.
3.2 Analysis of Foss projects. 3. Using software tools

SLOCCount and COCOMO model

The COCOMO model

COCOMO is a model to estimate the effort, the cost and the schedule of a software project.

COCOMO offers three granularity analysis levels:

- **Basic COCOMO**
  It provides a rough idea of the effort, cost and schedule of a software project as a function of the project size (expressed in lines of code).

- **Intermediate COCOMO**
  It takes into account software attributes (reliability, experience, project complexity...).

- **Detailed COCOMO**
  In addition to intermediate COCOMO, it takes into account the different development phases (analysis, design, implementation, test...):

We will focus on **basic COCOMO**
Basic COCOMO is a model that computes software development effort, cost and schedule as a function of project lines of code.

The computation of basic COCOMO depends on the family of software projects to which it is applied:

- **Organic projects**: Small and simple projects carried out by a small team of experienced engineers/programmers. Few (software, hardware or operational) constraints apply to the project.

- **Semi-detached projects**: Medium size and complexity projects carried out by a team of developers with a mixture of experience levels. Some constraints may apply to the project.

- **Embedded projects**: Projects with important software, hardware and operational constraints.

The computation of the COCOMO values is performed according to the formules in the next slide:
• **Effort (E in Persons-month):**

\[ E = a(KLOC)^b \]

KLOC: Thousands of lines of code that the project has

1 person-month is the amount of work done by 1 person in 1 month if working full time

• **Schedule (D: development time):**

\[ D = cE^d \]

• **Cost (P: Number of persons):**

\[ P = \frac{E}{D} \]

The parameters \( a, b, c, d \) depend on the type of projects according to the following table:

<table>
<thead>
<tr>
<th>Software project</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>2.4</td>
<td>1.05</td>
<td>2.5</td>
<td>0.38</td>
</tr>
<tr>
<td>Semi-detached</td>
<td>3.0</td>
<td>1.12</td>
<td>2.5</td>
<td>0.35</td>
</tr>
<tr>
<td>Embedded</td>
<td>3.6</td>
<td>1.20</td>
<td>2.5</td>
<td>0.32</td>
</tr>
</tbody>
</table>
SLOCCCount

SLOCCount is an application that is able to measure the lines of code (SLOC) of a software project

http://www.dwheeler.com/sloccount/

- Many programming languages are automatically identified by SLOCCount
- It also estimates the effort, cost and schedule of the project based on the basic COCOMO model
- It can also use more elaborate COCOMO models if more information about the project is given to it
- The parameters to perform the calculations can be changed by using the options:
  - --effort a b
  - --schedule c d
  - --personcost cost
- The option --filecount counts files instead of lines of code
CVSAnalY is an application that performs statistical analysis of a CVS/Subversion repository.

In particular, it provides committing information and also project size and modules information.

http://cvsanaly.tigris.org/research.html

[RKG2004]

- CVSAnalY creates a database in MySQL designed as follows:
  - **Table Files**
    * file_id (Primary key)
    * name
    * creationDate
    * filetype
    * size
    * module_id (foreign key)
    * ...
  - **Table Modules**
3.2 Analysis of Foss projects. 3. Using software tools

* module_id (Primary key)
* module (name)

- **Table** Commiters
  * committer_id (Primary key)
  * commiter (name)

- **Table** Log
  * commit_id (Primary key)
  * commiter_id
  * file_type
  * revision
  * ...

```
+commiter_id
| 1 |
+module_id
| 1 |
+file_id
| 1 |
```
In addition, CVSAnalY generates graphics for many statistical aspects concerning committing and project/module size.

The command:

```
$ cvsanaly --driver mysql --database testMetro
       --user josepma --password whatever
       --run-plugin graphs --output-dir ./graphs
```

generates a database called testMetro and a collection of graphics which are put in the `./graphs` folder.

Even if the MySQL DBMS is not up and running, CVSAnalY scans the CVS/Subversion repository of the project and generates a file with the SQL instructions necessary to create the above described database.
Information obtained from CVSAnalY and SLOCCount

Using those tools, a lot of information can be drawn from the subversion/CVS repositories.

We will concentrate in the following:

- **Committer’s information**
  - Commits vs. time chart (for each committer)
  - Commits vs. time chart (for each module)
  - Committers vs. time chart (for each module)
  - Commits vs. module chart
  - Committers vs. module chart
  - LOCs for each committer chart
  - Files for each committer chart

**Tools:** CVSAnalY and CODD

- **Project size and economic value**
  - LOCs (lines of code) for the project and basic COCOMO model results
  - LOCs in time for each module chart
  - Files in time for each module chart

**Tools:** SLOCCount, CVSAnalY
Websites that analyze FOSS projects

- **http://flossmetrics.org**
  - Quantitative analysis of FOSS repositories
  - Uses CVSAly
  - Web interface: Melquiades

- **http://flossmole.org**
  Similar to flossmetrics

- **http://cia.vc**
  Event counter (commits, messages...) on FOSS projects at real time

- **http://ohloh.net**
  User-friendly analysis of FOSS projects (commits, languages, contributors...)

  A dynamic view of the activity of a FOSS project throughout time
There are many more FOSS project analysis proposals in the literature.

In the following few slides we outline a case study of the projects:

1. Apache
2. Mozilla

Some conclusions about the FOSS development model are included at the end of the analysis.

See [Mock2005] for a complete description.
1. The Apache project

- **Web:**
  
  http://httpd.apache.org

- **Objective:**
  
  The Apache HTTP Server is an open-source HTTP server for modern operating systems including UNIX, MS-Windows, Macintosh and Netware. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards. Apache has been the most popular web server on the Internet since April of 1996.

- **License:**
  
  Apache License 2.0
• **Number of core developers:**
  
  – 6-9: The AG members that are developing in a given week and
  – Those developers who are about to be nominated to AG members

• **Number of users/contributors:**
  
  – Problems have been submitted by more than 3000 people (2005)
  – Code contributions have been made by almost 400 people (2005)
  – 15 core developers contributed 83% of the code but fixed only 66% of the defects
  – The rest of the defects were fixed by the rest of 400 code contributors
• **Project organization:**

  – The Apache Group (AG) consisted entirely of volunteers. They were 25 (in 2005)
  – AG members are developers who have contributed for an extended period of time (e.g., 6 months)
  – The core developers is a subset of the AG.
  – AG members have commit access to CVS
  – AG members can vote for the inclusion of changes (usually, votes are restricted to major changes)
  – A minimal quorum voting system for resolving conflicts

• **Development process:**

  No formally defined development process

  Usually a core developer iterates through a series of tasks:
  
  – Discovering an existing problem/new functionality to be implemented
  – Determining which volunteers will work on that problem
  – Design a solution
    The alternative solutions (if they exist) are sent to the developers list in order to get feedback from the group
  – Developing and testing the code within the local copy of the source
  – Presenting the code changes to the AG for review
  – Committing the code and documentation to the repository
3.3 Other approaches to FOSS project analysis. 1. The Apache project

- **Testing:**
  - Prerelease testing (unit testing)
    Developer tests the code that he/she has developed
    This test is informal
    *No additional tests are required prior to release*
  - Inspections
    After prerelease testing:
    * The code is posted to the developers mailing list for review (in the case of a stable release) or
    * The code is committed to CVS and reviewed afterwards by the AG (in the case of development releases)

- **Releases:**
  - Before a product release, one of the core developers takes the role of *release manager*
  - He/she sorts out all the issues that may prevent the release and helps the product reach a stable point
  - The role of *release manager* is rotatory

- **Documentation:**
  .xml
  There is a subproject that aims to create the documentation and to translate it to different languages

- **Version control tool:**
3.3 Other approaches to FOSS project analysis. 1. The Apache project

Subversion

- **Bug tracking tool:**
  
  Bugzilla

- **Communication mechanism:**
  
  Communication in the Apache project is done by means of e-mail lists, exclusively.

  The Apache project maintains 12 mailing lists. Some of them are the following:
  
  - User Support and Discussion
  - Apache HTTP Server Development Main Discussion List new features and patches for bugs are discussed
  - Apache HTTP Server Bug Reports List
    Reporting bugs (usually with no patch)
  - Apache HTTP Server Documentation Project
  - GUI Management Interface for the Apache Web Server
  - Mailing List for Testers of Apache HTTP Server Release Candidates

- **Foundation:**
  
  http://www.apache.org
2. The Mozilla project

- **Web:**
  
  http://www.mozilla.org

- **Objective:**
  
  A collection of projects aimed to develop a variety of products (firefox, thunderbird, Bugzilla, Bonsai...)

  The flagship project of Mozilla is the internet browser **firefox**

- **License:**
  
  Mozilla public license

- **Number of core developers:**
  
  Mozilla.org has 12 full-time members with the roles of coding and coordination

  Other non-Mozilla.org contributors act as module owners (78) and leaders of the test teams (6)
3.3 Other approaches to FOSS project analysis. 2. The Mozilla project

- **Number of users/contributors:**
  - 486 people contributed code (2005)
  - Almost 7000 people reported problems (95% of which are external people)
  - External people reported 53% of the problem reports
  - 1400 people reported problems that eventually lead to code changes
  - External participation has increased gradually over time
  - Most code changes have been developed by mozilla.org members and module owners
• **Project organization:**
  
  – Mozilla is operated by the mozilla.org staff (12 members in 2005)
  – They coordinate, guide the project and provide access to CVS
    They are a sort of “benevolent dictator”
  – Decision-making authority for a specific module is delegated to the module owner
  – Adding a new module requires the permission of Mozilla.org
  – Mozilla.org has the ultimate decision-making authority
    They are responsible for resolving all conflicts that occur
    They may designate and remove module owners

• **Development process:**
  
  – There exist a roadmap that contains the work to be done
  – mozilla.org members develop code for their expertise areas
• **Testing:**
  
  – Prerelease testing
    6 product area test teams who take responsibility for testing aspects such as internationalization, standards compliance, mail client...
    Test teams maintain test cases and test plans
  
  – Inspections
    Two stages of code inspections:
    * Module owners review patches concerning a specific module
    * Superreviewers review the interactions of that patch with the code base as a whole before it is checked in
  
  – Community contributions
    Fixes by the community are submitted as attachments to bugzilla problem reports
3.3 Other approaches to FOSS project analysis. 2. The Mozilla project

- **Releases:**
  - Automatic nightly builds
  - Milestones releases, approximately monthly.
    They are responsible for a group: drivers@mozilla.org

- **Version control tool:**
  CVS

- **Bug tracking tool:**
  Bugzilla

- **Communication mechanism:**
  A collection of newsgroups and mailing lists (each one mirroring its mate)
  They are accessible from
  - news.mozilla.org
  - lists.mozilla.org

- **Foundation:**
  http://www.mozilla.org
Conclusions of these two case studies

These two case studies suggest some conclusions concerning FOSS projects that are consistent with what has been presented in section 1.2 and 1.3.

However, bear in mind that these conclusions are only based in two case studies, therefore, they may not be true in all cases.

For these reason, [Mock2005] has called them hypothesis:

**Hypothesis 1:**

Open source developments will have a core of developers who control the code base, and will create approximately 80 percent 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.

This hypothesis is consistent with Pareto’s law.
Hypothesis 2:

If a project is so large that more than 10 to 15 people are required to complete 80 percent of the code in the desired time frame, then other mechanisms, rather than just informal ad hoc arrangements, will be required 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.

These mechanisms are based on splitting the project into modules or subprojects and give the ownership of each module/subproject to one developer or group of developers. In addition, other control measures (as inspections) may be required.

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.

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.

The fact that developers are experts in the domain of the software they write helps in achieving reliable and high-quality applications.

What will happen when FOSS projects are written for domains in which developers are not expert or are not sufficient motivated???

Hypothesis 7:

OSS developments exhibit very rapid responses to customer problems.
References


