Chapter 1: FOSS development model (1.1-1.2)

Josep M. Ribó

October 1, 2010
Chapter 1: FOSS development model

1.1 Proprietary vs. Open Source software development model
   1.1.1 The Cathedral model
   1.1.2 The Bazaar model (FOSS development model)

1.2 A discussion on the FOSS development model
   1.2.1 Criticisms on the FOSS development model
   1.2.2 A discussion
   1.2.3 Circumstances under which Bazaar development may outperform classical SE development
   1.2.4 Towards OSS 2.0

1.3 Statistics about FOSS projects
   1.3.1 A survey about FOSS developers
   1.3.2 Some surveys and statistics about the quality of FOSS projects
1.1.1 Proprietary development model
(The Cathedral model)

Traditional software engineering consists in the development of large software systems:

- By a team of engineers/programmers
- Through a software development model

The development of a software system is called software development project
• The **software development model** defines the **process** that must be followed in order to complete a software development project.

• This process consists of some **tasks** that must be carried out in a valid order and with some specific constraints

  (The valid order and the specific constraints are determined by the model)

• These **tasks** include:

  – Software requirements capture,
  – Specification
  – Design
  – Testing (verification and validation)
  – Configuration management
  – Maintenance-evolution
  – Project management

• These **tasks** should employ formal notations (UML, OCL, use cases, DFDs...)

• Quality is achieved by following the model systematically, and according to the planning, costs and risks estimated by the **project management**
1.1.1 Proprietary development model  
(The Cathedral model)

The software development process is performed, traditionally, in a *cathedral-style* [Ray2000]:

- Centralized project management
- Closed list of developers and beta-testers
- Scarce releases (only of stable versions)
- Strict development cycle (usually, iterative)
- Extensive use of formal documents (including programming guidelines)
- Design and implementation phases carried out by experts in isolation from their users
- It tends to be the case that
  - software is developed to meet the requirements established by a customer
  - developers are not users of that software
1. FOSS development model. Life-cycle

A complete life-cycle for a Free/Open Source Software (FOSS) project may consist of the following:

1. **Preliminary stage**
   
   Selection of a problem of interest and perform some development planning

2. **Prototype stage**
   
   Closed development of a *promising prototype*

3. **Transition stage**
   
   Open the project to the community

4. **Bazaar stage**
   
   Development, debugging and extending that prototype together with the community

5. **Responsibility transfer**
   
   Transfer the responsibility of the project to a competent successor in case of loss of interest

Stages 1 and 2 may also be called *Cathedral stage*
1.1.2 FOSS development model. Life-cycle

An important remark:

A FOSS does not initiate in bazaar-style

When a FOSS project reaches some degree of development and some releases have been published then the project may become public and continue its development in bazaar-style.
Preliminary stage: Steps

**Goal** of the preliminary stage of a FOSS project:

Select a problem of interest and perform some planning

Preliminary stage in a FOSS project ideally consists of three steps:

1. Problem selection
2. Project investigation
3. Project planning

They are explained in the following slides
Preliminary stage: Problem selection

**Goal:**
The FOSS project initiator chooses a problem which is important for him/her
A problem that *scratches an itch*

The starting point of a FOOS project usually is:

- **A problem of interest for a group of developers/company** for which:
  - There is no suitable application or
  - There are only proprietary applications or
  - The existing applications should be highly improved or customized in order to fulfill one’s needs.
  - There are some applications commercialized by a competitor

  **Examples:** Fetchmail, linux, PARI, NTL, GAP, Mono, openoffice, apache

- **A proprietary application that becomes open source**
  (to get more contributions, to become more popular and used...)

  **Examples:** Netscape, Doom, Java technologies, OpenOffice.org
• An application developed by a company which will sell services based on that application

   Example: SendMail pro, StarOffice

• Open source controllers (or other software) for a specific piece of hardware developed by a hardware company

   Examples: Intel, Motorola ...
Preliminary stage: Investigation of related projects

Goal:
Search for some FOSS projects that aim to solve (possibly in a partial way) the same (or a similar) problem

Why:

- We can join that existing FOSS project (and help it progress faster)
- Almost always, it is easier to join an existing project than to start a new one from scratch
- Even if we decide to redesign the application, reusing some modules will save us time

Tip:
Good programmers know what to write. Great ones know what to rewrite (and reuse)
Eric Steven Raymond
Preliminary stage: Investigation of related projects

However.....

Sometimes, we may decide not to join an ongoing FOSS project:

- We dislike or we are not familiar with the programming language or the technology with which that project is being carried out

- We dislike some important aspects of the ongoing project (involved people, architecture, decisions taken...)

Tip:
The starting of a new FOSS project may lead to effort duplication and to the existence of two mediocre projects instead of a bigger and more successful one

Think twice before starting a project from scratch
Preliminary stage: Investigation of related projects

Proposed procedure:


1. If an active FOSS project is found, join the bazaar-style development of that project

2. If an inactive FOSS project is found, try to contact the last coordinator of that project and propose either the project resume or its transfer to you

3. In no such a FOSS project is found, initiate a new FOSS project
Preliminary stage: Project planning

Before starting the design and implementation some planning should be undertaken:

- **Risk analysis:**
  - Is the developers’ motivation and commitment strong enough to initiate a long-term endeavour?
  - Is the project appealing enough to be completed with the existing developers or to attract contributors from the community?
  - Some preliminary schedule

- **Preliminary requirements:**
  - How the new project can differentiate itself from other similar projects
  - Which will be the initial extent of the project

**Tip:**
The more accurate this analysis, the more likely for the project to last long and be appealing for the community
Prototype stage: Steps

The **goal** of the prototype stage of a FOSS project is the development of a first prototype to solve that problem.

Prototype stage in a FOSS project ideally consists of four steps:

1. Prototype requirements
2. Prototype design
3. Prototype implementation
4. Prototype testing

They are explained in the following slides.
Prototype stage: Requirements

**Goal:**
State the requirements of the first application prototype

The requirements of a prototype usually come from:

- The needs of the project initiators
- The insight gained during the investigation of related projects
- If the project is a reimplementation of an existing application (e.g., a proprietary one), the requirements of that application

The traditional tools of software engineering (e.g. use cases, use case diagrams...) can be used in order to complete this phase
Prototype stage: Design

Goal:
Provide an architecture and design of the application

Some design features:

- The design should be extensible
  (new functionalities will probably be added in the future)

- The design should be highly modular

- It should be divided into subsystems provided with clear interfaces
  (a collection of contributors will work concurrently from different geographical locations)

- It should be as simple as possible
  (to encourage cooperation)
Prototype stage: Design

**Tip:**
A structure is stable if cohesion is strong and coupling is low.

**Tip:**
Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.
(Antoine de Saint-Exupéry)

Making the simple complicated is commonplace; making the complicated simple, awesomely simple, that’s creativity.
(Charles Mingus)
Prototype stage: Implementation and testing

**Goal:**
Create a software prototype which shows a *plausible promise* of the future project

- Usually, free software tools are used in the implementation
- Some, often informal, tests are carried out by the developers before showing the prototype
- More thorough tests (involving testing methodologies and tools) are usually not performed in this stage
Transition stage

The **goal** of the transition stage of a FOSS project is to make the project (which up to now has been developed in isolation) public, with the aim of getting contribution from the community.

Transition stage in a FOSS project involves making decisions on, at least, three aspects:

1. Distribution
2. Infrastructure
3. Management style

They are explained in the following slides.
Transition stage: Distribution

**Goal:**
Determine when a prototype should be made public/distributed

It should be a promising prototype

- **A very unstable prototype** with many bugs will not encourage prospective contributors to use it and, eventually, to join the project.

- **A very advanced prototype** may turn down volunteers due to a complex code or to the fact that the required functionality is already implemented.
1. FOSS development model-1.2 FOSS development model. Transition stage

Transition stage: Infrastructure

**Goal:**
In order to make the project public, some infrastructure is necessary.
This infrastructure is usually provided by a public hosting site which assists in collaborative software development.

**Remarks:**

- The project hosting site should provide:
  - Public code access
  - Release download
  - Version management
  - Bug management
  - Communication tools (mailing lists, forums)

- Examples of hosting sites: *Sourceforge, Savannah*...

- Several forums may exist in a FOSS project: *users forum, developers forum, announcements*...

- In FOSS projects, communication tools tend to replace formal documentation

- All communication between contributors of a FOSS project should use these public communication channels
Transition stage: Management style

**Goal:**
The project owners’ control should weaken when the FOSS project is made public

The project owner should become a sort of *head maintainer* who incorporates contributors’ code based on certain criteria

Those criteria should be as much transparent as possible

**Why**
The way in which the FOSS project is controlled by its owners determines its evolution and success

- **Too much control:**
  - Not appealing for prospective contributors
  - Possibility of fork

- **Too permissive:**
  - Poor code may be inserted
  - Design may get chaotic

In 1.4, we will show some examples of how control is performed in several FOSS projects
Bazaar stage. Main features

**Goal:**
Develop and debug an application in a distributed way in collaboration with the community

This is the distributed development style typical of FOSS projects

It has been characterized by Eric Steven Raymond [Ray2000]

**It has the following main features:**

- A small group of core developers and a bigger community of users/contributors (volunteers)
- Concurrent debugging and development by users/contributors
- Peer review
- Early and often releases
- Organization based on a meritocracy and a benevolent dictator
- Keep the modularity and simplicity in the design
The essence of the Bazaar-style development (Linus’s law):

Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone (Eric S. Raymond)

Or...

Given enough eyeballs, all bugs are shallow
Bazaar stage. Core developers and contributors

• A small group of core developers (1-5 is a usual range)
  – They are responsible for the design and development of the project.
  – They have the right to commit changes to the code base repository (see chapter 2)
  – Usually one of them is the project leader/coordinator
    This project leader is usually the benevolent dictator (see below)

• A bigger community of users/contributors (they can be 100s or more...)
  – They are volunteers
  – They use the software and beta-test it
  – They may propose some new requirements
  – They may propose patches
  – They may propose some new development
  – Some very active contributors may eventually become core developers
    One way to become core-developer is to take charge of one project subsystem
  – In general, plain contributors do not have the right to commit changes to the code base repository
Bazaar stage. Core developers and contributors. Summary

Pareto’s law:

- 80% of the code corresponds to 20% if the developers
- 20% of the code corresponds to 80% if the developers
  (including application testers that send issues about the code and/or propose patches)
Bazaar stage. Concurrent development and debugging processes

**Goal:**
Make the FOSS development/debugging process as parallel as possible with as many contributors as possible

**Why:**

- To speed up the development process
- Many eyes can find (and fix) bugs better and faster than just few of them
- Contributors’ ideas are one of the most valuables issues in a FOSS project
Bazaar stage. Concurrent development and debugging processes

Means:

- **Listen to the users**
  - Answer to their questions
  - Encourage them to participate

  *Treat them as co-developers and they will become co-developers (E.S. Raymond)*

- **Listen to the contributors**
  - Incorporate their patches as soon as possible
  - Poll them about design decisions.
  - Recognize their good ideas and incorporate them into the design

- **Issue releases as soon as possible**

Since developers/contributors are often users in a FOSS project, bug detection will typically be easier, deep and more accurate
Goal:

Improve the quality of software by allowing multiple inspections of the already written code

Remarks:

• The practice of peer review has been largely established as a great quality assurance practice

• One benefit of opening the source code to the community is that anyone can review the code and search for mistakes. This allows a large number of code reviews

• Successful FOSS projects benefit from a large base of users and developers who inspect code and identify bugs
Bazaar stage. Early and often releases

Goal:
Issue new releases, incorporating patches to known bugs, as soon as possible

Why:

- Users/contributors do not get distracted with old bugs for which a patch already exists
- The project is alive and this stimulates users/contributors to keep on contributing with the project

Tip:
Make the users/contributors keep the project in their minds
Bazaar stage. Benevolent dictator

**Goal:**
The personal and technical aptitudes of the project leader/coordinator should help the FOSS project progress firmly and smoothly

**Remarks:**

- A FOSS project is a *meritocracy* in which a developer gets credit among his/her peers for his/her technical skills and the value of his/her contributions to the project

- The project leader should have this kind of credit among the project users, contributors and co-developers

- The project leader usually acts as a *benevolent dictator* who has the responsibility for taking a decision in case of a conflict *in which the different parties cannot reach an agreement*
• The project leader should manage this *power* with responsibility since:
  
  – The users and developers are often volunteers and they can abandon the project at will and
  – They may fork the project if many of them think that the leader acts in a irresponsible way

• In the majority of cases, the common sense prevail and the right to fork a project is scarcely used
Bazaar stage. Benevolent dictator

Summary:

The coordinator-leader of the FOSS project should have, at least, the following features:

- The project should be appealing to him/her
- He/she should have communication skills
- He/she should be able to recognize good ideas and incorporate them to the project (even if they are not his/hers)
- He/she should not behave dictatorially
Bazaar stage. Benevolent dictator

Project organizations

- **In small projects** with a small number of core developers (1-3), the project leader often acts as a benevolent dictator.

- **In bigger projects**, it is common to use other sorts of structures:
  - **Perl**: A benevolent dictator that rotates among the core developers.
  - **Apache**: The core developers form a voting committee: “Project Management Committee”.
  - **Linux**: Hierarchy of four levels: ordinary developers, maintainers, trusted lieutenants, and the benevolent dictator. Proposed changes are sent upwards in the hierarchy until they are approved by the benevolent dictator (currently, Linus Torvalds).
1.2.1 Criticisms of the FOSS development model

- Criticisms concerning the FOSS model
- Criticisms concerning the FOSS business model
- Criticisms concerning quality assurance aspects
Criticisms concerning the FOSS model

- The contributors of FOSS projects may be volatile [Ray2001]
- FOSS projects lack a conventional project management [Ray2001]
- Brook’s law [Ray2001]
- FOSS projects lack a formal design process [Fitz2005]
- FOSS development process is not really a new development process [Fug2004, Glass2005]
- FOSS projects depend on modularity. However, too much modularity may lead to coupling problems [Fitz2005, Ruso2005, Scac2002]
- The advantages of FOSS are only clear for those applications for which users are also programmers [Glas2005]
- Version proliferation of FOSS products (specially, Linux) is harmful [Fitz2005]
- Most FOSS projects are unsuccessful
1.2.1 Criticisms of the FOSS development model-The FOSS model

The contributors of FOSS projects may be volatile

**Criticism:**
FOSS projects may have a large number of contributors but they are volatile, not necessarily sustained over time. On the other hand, customers of traditionally developed software can expect sustained investment over time on that software.

**However....**

Some FOSS projects have lasted for many years (e.g., emacs, linux, apache, mozilla...
Brook’s law

Brook’s law:
Adding manpower to a late software project makes it later. Due to:
- Learning process of the newcomers
- Communication overhead among the whole group (a linear increase in the number of developers yields a quadratic increase in the communication complexity): \( n^*(n-1)/2 \)

This effect is reduced in bazaar-style programming since contributors work fairly independently from each other.

The communication overhead applies mostly within the core-developers group since they have to manage the communications coming from the whole group of contributors.
Brook’s law (2)

Example: Apache:

- The role of core developers is to implement new functionalities (10-15 core developers account for 88% of the new code)

- The role of contributors is mainly to submit problems and to propose fixes:
  - 10-15 core developers submitted less than 5% of problems and produced 66% of the fixes (vs. 88% of new code)
  - 1000s of contributors within the Apache community submitted the vast majority of problems

This job can be done in a fairly independent way without incurring in communication overhead
**FOSS projects lack a conventional project management**

**Criticism:**
FOSS development underestimates the productivity-multiplying effect of conventional management

**However....**

According to Raymond [Ray2001], the tasks of a project manager are either (next-to) unnecessary in FOSS projects or performed by the project leader or peers:

**Project manager tasks:**

- To define project goals (FOSS: project leader)
- To monitor the project and its details (FOSS: peer review)
- To motivate people to do boring work (FOSS: There are some contributors for whom the work is not boring)
- To organize people to maximize productivity (FOSS: proficient and highly motivated developers)
- To marshal resources to develop the project (FOSS: Unnecessary)
FOSS projects lack a formal design process
[Fit2005]

Criticisms of the FOSS development model

**Criticisms:**
In the bazaar development style, there is no real formal design process, there is no risk assessment nor measurable goals, no direct monetary incentives for most developers, informal coordination and control, much duplication and parallel control
[Fit2005]

However...

- Major decisions concerning **requirements analysis, design** and also some **risk analysis** are taken before the project gets into bazaar style

  They may be taken using traditional S.E. techniques

- Many successful FOSS projects have started in the context of a software company with assigned staff and with a traditional preliminary design. Afterwards, they have been made public FOSS projects

  This is an important trend in current FOSS
• **Requirements analysis** is easier than in traditional software development since most developers are also users.

• Design of FOSS projects is usually extremely modular in order to make it easy its extension.
FOSS development process is not really a new development process

Criticism:
- Most FOSS projects are carried out by just 1 or 2 developers
- Large FOSS projects (Linux, Apache, Mozilla) have very well structured processes which resemble those of proprietary projects
- Some features of FOSS projects can also be observed in proprietary projects

[Fug2004]

However....

Bazaar-style development is clearly innovative in many aspects:

- Parallel debugging and module development by a mass of voluntary contributors through internet
- Peer review process
- Meritocratic relationship among contributors
1.2.1 Criticisms of the FOSS development model-The FOSS model

- Rapid release launching
- Source code can be studied and improved by anybody
- Bureaucracy is reduced
- Those volunteers that join the project are usually technically competent and motivated

Although some research has to be done yet, many software engineers think that these innovations to traditional SE development model do, in certain contexts, lead to a faster, more reliable and more secure development process and final product.
**FOSS projects and modularity**

**Criticism:**
FOSS projects depend on modularity. However, too much modularity may lead to coupling problems which may affect reuse and maintainability [Fitz2005, Ruso2005, Scac2002]

*A specific example:*
The number of instances of common coupling in the Linux kernel grows exponentially with version number [SJW2002]

This warning must be taken into account when designing/redesigning FOSS projects

It is important to see whether this is just a single example or applies to other FOSS projects
1.2.1 Criticisms of the FOSS development model - The FOSS model

Users-programmers

**Criticism:**
The advantages of FOSS are only clear for those applications for which users are also programmers [Glas2005]

**However...**

Currently, there are many FOSS projects in "higher level" domains such as office suites, ERPs...
1.2.1 Criticisms of the FOSS development model-The FOSS model

Version proliferation

Criticism:
Version proliferation of FOSS products (specially, Linux) is harmful [Fitz2005]
It is difficult to keep different versions of the same product up to date

Indeed.

For this reason, sometimes, only the most popular configurations have been sufficiently tested
Criticisms concerning the FOSS business model

- The FOSS model may not be an effective business model
FOSS model may not be an effective business model

**Criticism:**
The economic model of FOSS is impractical [Glass2005]
It is not clear if companies can survive and make profits by simply selling services [Fug2004]

**However:**

- The possibility of “selling services” of integration and customization of open source software allows the creation of many small-medium companies that provide those services to their customers (currently, there are just package-selling companies that send royalties to Microsoft)

- Some FOSS projects are developed under fundations that raise funds to support these projects (example: Apache)
1.2.1 Criticisms of the FOSS development model-The FOSS business model

- Not all software is sold:
  - Most software developers work in software which is for their own company and which will never be sold [Ray2001c]
  - In addition: A lot of proprietary software is also given for free in order to keep the market position of the producing company (.NET, Internet explorer)

- Some organizations (for-profit companies, Universities ...) provide *developer time* to develop FOSS projects which are of interest to them

  These projects provide prestige to the organization that has supported them

- Sometimes, a number of organizations with a common need support a join effort to create some software product which is needed by all of them (example: Sakai, a FOSS virtual campus developed by many universities)
• Some companies are interested in having open-source applications with proprietary modules built upon (example: Sun sells StarOffice as a proprietary software which incorporates some modules on top of the FOSS OpenOffice)

• Some for-profit organizations have created a FOSS project and sell the license (dual licensing), the support services and the training to their customers (example: MySQL, ZOPE, companies that package Linux distributions).

• Some for-profit companies use a business model which is called *software as a service*

  According to this model, their clients access an application through internet and pay a *subscription* for its hosting and use

  The application can be a FOSS application

• Some hardware companies are interested in developing open source drivers and applications that can run in their hardware devices

• See [Ray2001c]
FOSS business model. Dual licensing

Some software companies offer their products with two (or even more) licenses

A product A can be licensed **both** with:

- **A restrictive free software license (usually, GPL)**

  Thanks to this license, anyone in the community may study the source code of A, modify and redistribute it.

  However, nobody can integrate it into a proprietary application.

- **A proprietary license**

  Those companies that want to integrate A into their own proprietary software products should pay a license fee.
Benefits of multi-licensing:

- Companies get economic profits from selling a software product
- However, the product is still free software and the community may study and redistribute it
- Since it is free software, the company may benefit from the collaboration from the community
- Dual licensing may be also interesting to achieve license compatibility with other products

Difficulties with multi-licensing:

- The company should have the ownership of the whole code in order to apply a dual license to it
- The contributions made by the community may not be licensed with the proprietary license.

The company may ask the contributor to agree on a contributor license agreement
Examples:

- **MySQL AB (MySQL)**
  
  Two licenses: GPL and commercial
  
  http://www.mysql.com/about/legal/licensing/oem/

- **Nokia (Qt)**
  
  Three licenses: GPL, LGPL, commercial
  
  http://qt.nokia.com/products/licensing/licensing#qt-gnu-lgpl-v
FOSS business model

Three types of users:

**EU** End users. No technical skills

**TD** Technology *****

**TC** Technology creators or technology experts

- **Hardware firms. Servers**

  Clients are usually TC

  A free OS:
  
  - A cheaper choice
  - Reuse of Unix programs
  - Customization possible between TC (e.g., Google has more than 40,000 servers with customized versions of Linux).

- **Hardware firms. Notebooks**

  Clients are usually EU

  They may buy FOSS products for price reasons.
1.2.1 Criticisms of the FOSS development model—The FOSS business model

Proprietary software may take advantage of its dominant position and easily regain its marketshare.

Example: Linux initially installed in Asus eee-PC were rapidly changed to Windows-XP.

FOSS

- **Software service and architect companies**

Conclusions:

FOSS business models does not work properly with EU.

FOSS gets its advantage when its business model is based on something else than license prices.
Criticisms concerning quality assurance aspects

Some criticism have come up concerning some quality aspects of FOSS projects:

- Reliability of FOSS projects
- Peer review
- Lack of formal testing (including multiple configurations)
- Bad quality of some FOSS code
- Some code in large FOSS projects may become unsupported
- Lack of documentation
Reliability of FOSS projects

**Criticism:**
The claim that FOSS is more reliable than the proprietary one is naive and unprovable

[Glas2005]

However....

- Many FOSS products have been greatly praised for their reliability (in comparison with the corresponding proprietary alternatives): Linux, emacs, Apache, Tomcat, JBoss, Mozilla, Gimp...

  See, for instance, [Whee2005]

- Equivalent FOSS applications are more reliable than proprietary ones [MKL1995]

- The defect density in successful FOSS projects is equal or lower than in proprietary products [Bou2005]
• Bugs found in FOSS projects are solved quicker than those found in proprietary ones [Bou2005] because:
  – There may be more developers working on it and
  – FOSS projects distribute quick patches to solve bugs instead of *service packages* that contain the patches to a collection of bugs detected in the last few weeks or months.
  *Service packages* are usual in proprietary products.

• Some studies show that Linux is highly reliable (in particular, more reliable than Windows-NT):

  [Link 1](http://www-128.ibm.com/developerworks/linux/library/l-rel/)
  [Link 2](http://web.archive.org/web/20010602081713/www.zdnet.com/sp/stories/issue/0,4537,2387282,00.html)
  [Link 3](http://news.com.com/Security+research+suggests+Linux+has+fewer+flaws/2100-1002_3-5489804.html)

• Sites using Microsoft’s IIS web serving software have over double the time offline than sites using the Apache software:

  [Link 4](http://web.archive.org/web/20011011215009/)
  [Link 5](http://www.syscontrol.ch/e/news/Serversoftware.html)

  However, these studies concern only main projects. Other studies should be carried out to assess the minor ones.
Security of FOSS projects

Criticism:
The availability of source code is a significant security threat to government organizations using FOSS (since hackers can study this code and exploit its vulnerabilities) [Bro2002]

However...

• This sentence promotes an unacceptable approach to software security based on Security by obscurity

• Evidence does not support this claim:
  – More bugs have been found in widely used proprietary systems than in their FOSS counterparts
  – Since the code is publicly available, more developers may work on those bugs and fix them earlier
Peer review in FOSS projects

**Criticism:**
The assumption that all open source code is thoroughly reviewed by its readers is false. Review of source code is likely to be very spotty
[Glas2005]

However...

- The review/test procedures for large FOSS projects is thorough:
  - **Mozilla:** All changes undergo two inspection levels
  - **Apache:** Changes must be reviewed by the group of 15 core developers
Testing and FOSS projects

Criticism:

- 80% of FOSS projects do not produce test plans. Only 40% use test tools [Glas2005]

- There is a lack of attention in many FOSS projects to basic, accepted and mature test techniques (e.g., coverage aspects, regression tests, test suites...) [Zhao2003]

- FOSS projects support multiple configurations. It is difficult and time-consuming to test all of them
Testing and FOSS projects

However...

- Which 80% of FOSS projects do not have test plans. But which 80%??

The vast majority of FOSS projects are initiatives of a few (mostly 1-2) developers with not much help from the community. It is not fair to assess these projects in the same way as the larger ones.

- As projects get larger, testing techniques are more mature

  Example: Mozilla has 6 test teams that maintains test cases and test plans and uses automated testing tools

- An important part of the testing process is developed by the community in FOSS projects (ex: Apache has received bug reports from thousands of different people and hundreds of them have sent patches)
Bad quality of some FOSS code

Some parts of the FOSS code of major projects is bad [Scac2002, Ruso2005]
Heterogenity, lack of coding standards.....

However...

- It is difficult to use this argument against the open-source code, because it simply cannot be compared to the proprietary one.

  Has anybody seen the Windows code???

- Is this situation the rule or the exception in major FOSS projects (in which peer revision is thoroughly performed)?
Unsupported code in FOSS projects

**Criticism:**
Developers that have been responsible for a specific module may abandon the project and that module may not be maintained anymore. [Mich2005]

Although this is certainly a danger, something similar may happen in traditional S.E.

In big projects, somebody else will take over.
Lack of documentation in FOSS projects

Criticism:
Usually, documentation is not an appealing task and contributors tend not to devote time to it.
Even if some volunteers want to contribute in this area, they receive little help from the project developers [Mich2005]

However...
There are many FOSS projects with excellent documentation.
In particular, big projects tend to have quite good documentation (both for developers and for users)
FOSS development model. General conclusions

- **Linux Law applies**
  
  The debugging of a FOSS application can be (and, in fact, it is) distributed among the community without much complexity overhead

- **Specific tools**

  Specific tools that deal with the particular needs of FOSS projects are widely used by these projects:

  - A repository (a forge)
  - A bug manager
  - A version manager
  - Mail lists
• **Reach a critical mass**

Those FOSS projects that do not reach a critical mass may survive but they will suffer from a lack of resources to find and repair defects.

• **Grow a sustainable community**

That community should have members with different roles associated to:

- A small group of **core developers** who:
  * keep modularity
  * ensure peer review
  * take responsibility for the bulk of the coding
- **Contributing developers (active contributors)** who will take responsibility for adding new features (the more modularity, the easier)
- **Bug reporters (passive contributors)** who take responsibility for system testing and bug reporters
- **Users**
• **The emblematic projects may be better**

The emblematic FOSS projects (Linux, Mozilla, Apache, Perl, MySQL, ...) may be better than most of their proprietary equivalents in terms of

– Defect density
– Product reliability
– Product security (if applicable)
– Efficiency in code development and defect repair

See [Zhao2003, Whee2005]

• **The success of the user-developer**

FOSS projects are more successful in those areas that interest developers. That is, for those applications for which developers are also users:

– Requirements are captured faster
– Debugging is also faster because the developers are the users that find the defects and propose a fix for them
– The whole development cycle is faster
• Virtual Project management

FOSS projects usually rely on virtual project management:

Some people, distributed throughout the world, are engaged in a participative management that arise from a decentralized community of developers

These people act as team leader, sub-system manager or system module owner

Versus delegative management that occurs in centralized corporate structures

• Informality

– The use of traditional software engineering tools and methodologies is not paramount in most FOSS projects
  For example: requirements engineering, CASE tools, UML or other modelling languages, testing tools...
– If the group of core developers has fewer than 10-15 members, the process may be not very formal (Apache)
– However, if the group of core developers is bigger then some initiaves should be taken in order to succeed.
  Examples:
  * Follow explicitly a more formal model
  * Divide the project into subprojects
  * Introduce more formality into the inspections
FOSS development model. Conclusions concerning quality

- **Make peer review paramount**
  - Make peer review a main part of your FOSS development process
  - Optimal: 5-6 peer reviewers

- **Release early, release often**
  - New releases that incorporate peer reviewed patches for the bugs and/or new functionalities should be published regularly
  - This maintains the interest of the contributors/users and makes them feel that their work is productive
  - It also maintains a continuous debugging process
  - Often releases get a reduction in the process cycle time


• **Testing**

  – Test in FOSS projects relies on continual improving which is based on:
    
    * release early, release often instead of high quality milestones
    * Peer review
  
  – Users perform the bulk of the testing in most projects (however, Mozilla has dedicated test teams)
  
  – Formal testing usually requires more resources and it is out of the question for small FOSS projects
  
  – In theory, peer review, prototype testing (release early, release often) allow early bug finding

• **Go modular, son**

  – High modularity increases quality
  
  – Small-size components have fewer defects and are more maintainable and evolvable than large modules
• The larger, the better

− The maturity of the development model and the quality assurance strategies of FOSS projects usually depend on the project size.
  Large projects with many users/contributors and an important number of developers tend to be more mature concerning these aspects.
  Some traditional SE practices, tools and methodologies are used in large projects.
− Many FOSS projects (specially small ones) have been abandoned.
− Research ([WMZ2006]) suggest that:
  * rich get richer: The more developers a community already has, the more developers it will attract.
    · Core-peripheral communication structures (core developers-users) communications)
    · Preferential attachment (as project evolves, popular nodes get even more links)
  * Promiscuousness is good: The more developers a community shares with other communities:
    · the more developers of other communities will interact with it and
    · the more information and ideas will flow to both communities
• **Keep the community spirits high**

The project will get more (and more qualified) contributions if:

– It is interesting for developers (i.e., it *scratches an itch*)
– It is highly modular
– It has good documentation
– It involves new challenges for the community (e.g., learning a new technology)
In the last few years, a shift is being observed in the FOSS model [Fitz2006]
Some business models compatible with the FOSS have started to be appreciated by companies:

• Pay for software services

• Market creation

• Taking advantage of the community contribution

As a result, the FOSS model has been undertaken by an increasing number of companies
This fact has brought about a shift in the FOSS projects:

- Many developers working full-time in a FOSS project (paid by the company)
- The project motivation is more a business strategy that involves open source than an *itch worth scratching*
- Projects move from backoffice (operating systems, languages, DBMS, web servers...) to frontoffice (office suites, ERPs...)
- The tasks of requirement analysis and design acquire more importance (requirements are not universally understood for these new kinds of applications)
- The process is less bazaar-like
  
  The core developers are company employees
  
  In any case, the official communication channels where issues are discussed and decisions are taken are still the mailing lists
- Extern contribution is oriented most of all to bug correction
Relationships between firms and open source communities

- Companies may benefit from ideas and products coming from communities
- However, they need a change in its business model
- Companies may change the role from owners of resources to coordinators
Four case studies

Companies engaged in FOSS as main business model but which are different one from another in terms of:

- Products developed
- Services offered
- Business model

Which kinds of relationships they have had with the community?
Business model. MySQL

- Dual license
- Selling services
- Training
Development in community. MySQL

- MySQL has built and maintains a huge community of users and developers
- Code contributions assigned to the firm
- MySQL encourages and contributes to mailing lists, forums and workshops
- Encourages bug reports, patches, some development from the community
- Peer review by the community. Extensive internal peer review
Business model. Cendio

Thin client application (allows remote connection and access to client devices such as mouse, keyboard, file system...)

Proprietary license
Development in community. Cendio

- Cendio searches for FOSS projects whose ideas/code may be integrated into Cendio products
- It has not established its own community
  That is: Cendio uses communities instead of creating its own.
- Non-crucial software is given back to the community (as a return)
- It has to check thoroughly the licenses to avoid legal problems
- Difficult to steer and influence communities
- Difficult to align community interests with company’s own interests.
Business model. Roxen

A programming language and a web server based on it

It combines FOSS and proprietary software

It sells proprietary licenses
Development in community. Roxen

- Created its own FOSS community to receive contributions from external developers in order to speed up software construction

  This model was not successful.

- The community still exists but with fewer members and activity

- The product was not that successful and the company has evolved to more traditional proprietary business model based on selling licenses
Business model. SOT

A linux distribution, a linux server and an office suite mainly oriented to Finnish and Russian markets
• Created its own FOSS community to receive contributions from extern developers in order to speed up software construction

This model was not successful.

• It got not so many contributions

• Different directions between the company and the community

• The product was not that successful and the company evolved to selling extra functionalities

• It did not work either and SOT has given away the code and sells services (customization, training...)

Development in community. SOT
Aspects of relationship firms-communities

- Accessing communities to extend the resource base
- Aligning firm strategies with the community
- Assimilate communities
Accessing communities

**Tactics:**

- Establish new communities
- Identify and use existing communities
Aligning strategies

Tactics:

- Adopting licensing practices that clarifies ownership
- Influencing direction of development
  - Incentives (e.g., salary)
  - Propose interesting challenges
  - Having firm representatives well established in the community to direct its work
Assimilating the work of the community and sharing work with it

Tactics:

- Devote resources to evaluate community source code
- Feed back non strategic source code to the community
Relationships between firms and open source communities

The relationship between firms and communities depends on the type of firm:

- **Firms (and individuals) consisting of end-users that use general-purpose technology**
  - No need of customization
  - Use the software as freeware, to reduce costs
  - No involvement with the community
  - However, their contribution is fundamental in order to spread the use of the FOSS product and technology
  - *Examples of such firms/users:* any firm/user that uses FOSS software without community involvement and without any need of maintenance or customization
  - *Examples of FOSS products involved:* Firefox, open office, Sage...
• **Firms consisting of end-users that need customized technology**

  – They may hire a FLOSS service firm in order to maintain and adapt FLOSS software to their needs.
  – They may propose improvements and new functionalities to the software. However, this is done in an indirect way through the hired service firm.
  – No involvement with the community. They delegate this involvement in the service firm.
  – *Examples of FOSS products involved:* ERPs (ex: Compiere, OpenBravo)...
  – *Examples of such firms/users:* any firm that needs specialized, customized and maintained FOSS software but which does not have the expertise to take part in its development.
Service firms

- Service firms offer services to third parties on FOSS products that (in general) have not been developed by them.
- These services involve:
  * assistance, assurance and adaptation (3A services)
  * integration
  * training
- They depend on the quality maintenance and continuous evolution of those FOSS products.
- A way to ensure that quality is getting involved in the development process with the community.
- Usually, they are involved in bug reporting, bug fixing and even development.
- In any case, they must make important R & D efforts in order to be able to absorb external technology.
- Examples of FOSS products involved: ERPs (ex: Compiere, OpenBravo).
- Examples of such firms/users: Compiere partners
● Firms that produce business packages, software frameworks...

– Communities are a crucial asset for these firms
– Symbiotic relationship between these firms and the communities
  ∗ Communities contribute code, bug reports and new features, which speeds up the development
  ∗ By doing that, the needs of the community are included in the product
– The firms integrate the contributions and control the evolution of the product
– Examples of FOSS products involved: ERPs (ex: Compiere, OpenBravo), Spring, MySQL...
– Examples of such firms/users: Compiere, MySQL...
1.3.1 Statistics about FOSS projects. A survey about FOSS developers

- Survey carried out by the Boston Consulting group about FOSS project contributors
- Survey and statistics concerning quality of FOSS projects
1.3.1 Survey about FOSS contributors

• Motivation

– Intellectually stimulating
– Improve skills
– Code should be open
– Not about defeating proprietary software

• Attitudes towards the community

– Next best thing to having good ideas is recognizing good ideas from others (56%)
– Free software is a matter of liberty, not price (48%)
– When we prepare a program, it’s just like composing poetry or music (42%)

• Weekly time volunteered by contributors

– Less than 1 hour (8%)
– 1-5 hours (30%)
– 6-12 (26%)
– 13-20 (15%)
• **Professional occupation**

  – Programmers and IT professionals: 45%
  – Students 20%
  – Other 18%
  – Academics 6%
Information concerning projects

From a sample of projects from Sourceforge and Freshmeat

Project size:

- Tiny (less than 1,000 lines of code): 10%
- Small (1,000-10,000 lines of code): 53%
- Medium (10,000-100,000 lines of code): 31%
- Large (More than 100,000 lines of code): 6%

Developer number:

- 1 developer: 51%
- 2-5 developers: 43%
- More than 5 dev.: 5%
1.3 Statistics about FOSS projects. 1.3.1 A survey about FOSS developers

**Starting motivation:**

- Meet personal needs: 60%
- To help software community: 28%
- Company needs: 24%

**Devoted time:**

“giving spirit”

- Personal time: 77%
- Partially supported by the employer: 12%
- Full time dedication to open source: 5%
New releases:

- Every month: 43%
- Every 15 days: 29%
- Every week: 11%
- Every 6 months: 7%
- Irregularly: 10%

Configuration management tools:

- Version management: 89%
  All large projects and 50% of tiny ones
- Bug tracking: 61%
- Every week: 11%
- Every 6 months: 7%
- Irregularly: 10%
1.3 Statistics about FOSS projects. 1.3.1 A survey about FOSS developers

**Documentation:**

It does not play a dominant role:

- “To-do list” (inc. list of pending features and open bugs): 84%
- Application build and installation documentation: 62%
- Design documents: 32%
- Release plan document: 20%

**Testing:**

Not so much time and effort devoted as in traditional software engineering (to some extent it is transferred to users)

*Testing techniques:*

- Provide inputs that imitate user behaviour: 68%
- Provide extreme values as inputs: 25%
- Use of assertions: 25%
1.3 Statistics about FOSS projects. 1.3.1 A survey about FOSS developers

- Use of a regression test suite among the large projects: 53%

Testing tools:

- Use of a test suite to support testing: 48%
- Use of a regression test suite among the large projects: 53%

User participation:

- In 20% of the projects, users find 20-40% of the bugs
- In large projects users found 80% of hard bugs
- In tiny projects users found 40% of hard bugs
Conclusions

- User participation very high
- Great use of configuration and bug tracking tools
- Documentation not a high priority
- Informal testing
  (Rely on users)
Survey and statistics concerning the use and quality of FOSS projects

This information comes from several sources. It has been compiled by:

[Whee2005]: David A. Wheeler: Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!
http://www.dwheeler.com/oss_fs_why.html

The article contains much more data. We have just selected some of it

The data is split into the following categories:

- Use of FOSS
- Performancs
- Reliability
- Scalability
- Security
Use of FOSS. Web servers market

Use of FOSS. Web servers market (2)

Use of FOSS. Web servers

Source: Netscraft survey

GNU/Linux is the 2nd web serving OS on the public Internet (counting by physical machine), according to a study by Netcraft surveying March and June 2001.

- Windows 49.2%
- Linux 28.5%
- Solaris 7.6%
- BSD 6.3%
- Other Unix 2.4%
- Other non-Unix 2.5%
- Unknown 3.6%
Use of FOSS. Plans to use FOSS
Source: 2004 InformationWeek survey

- 67% of companies use FOSS products
- 16% expecting to use it in 2005
- 17% No near-term plans to use FOSS products.
Use of FOSS. Email servers and scripting languages for web applications

• **Web servers**

  Source: MailChannel's survey (2007)

  Sendmail and Postfix are the two most popular email servers

  They are FOSS programs

• **Scripting language for web applications**

  – 2002: php was the nb. 1 scripting language for web applications
  – 2008. No statistics found. However, LAMP+JEE have a big piece of the web application development market
Use of FOSS. DBMSs

Source: Survey by research company Evans Data (2003).

- Use of MySQL: Rise 30% over 2002
- Use of Microsoft’s SQL Server and Access databases: Rise 6 percent over 2002
- February 2005: survey of developers and database administrators performed by Evans Data Corp.: 64% of developers and database administrators in US use FOSS DBMSs.

This rise in the use of FOSS DBMSs is due to the following facts:

- Their performance has increased to an acceptable point for corporate enterprise environments.
- These environments have tight budgets for database costs
- They are easier to combine with other development environments
1.3 Statistics about FOSS projects. 1.3.2 Statistics concerning FOSS projects

Performance


FreeBSD and Linux beat Windows XP in most of the tests they performed. Furthermore, in those tests in which Windows XP proved superior, there were more efficient ways of programming those tasks in a Unix-like system.

Example:

Number of CPU cycles needed to "create and start a process:

- 1,032,000 for FreeBSD
- 719,000 for Linux
- 5,376,000 for Windows/XP
Reliability

  
  80% of the top ten most reliable hosting providers ran OSS/FS
  
  - 4: GNU/Linux,
  - 4: FreeBSD
  - 2: Microsoft Windows.

- Source: Experiment by Prof. B. Miller [MKL95]

  Prof. Barton Miller in 1990 and 1995 conducted some experiments consisting in sending random streams of data to proprietary Unix applications and to open source applications

  In both occasions, FOSS applications behaved better when fed with unexpected data
Scalability

Source: Forbes (2005)

GNU/Linux dominates in supercomputing: GNU/Linux is used in 78% of the world’s 500 fastest supercomputers use GNU/Linux, most of the world’s ten fastest supercomputers

- 301: Linux
- 189: Unix
- 2: FreeBSD
- 1: Microsoft Windows
Security

• **Browsers (1)**


Recommendation to use browsers other than Microsoft Internet Explorer (IE) for security reasons.

In particular, Microsoft had failed to patch a critical vulnerability for 9 months

• **Browsers (2)**


- 78% (284/365) of the time in 2006 Internet Explorer was vulnerable to dangerous known attacks, for which no patch to fix it was available
- 2% (9/365) of the time was Mozilla Firefox vulnerable to such attacks
1.3 Statistics about FOSS projects. 1.3.2 Statistics concerning FOSS projects

• **General**

Source: BZ Research survey between more than 600 software development managers on windows vs. Linux security

<table>
<thead>
<tr>
<th></th>
<th>MS Windows Server</th>
<th>Linux</th>
<th>Sun Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very insecure or Insecure</td>
<td>58%</td>
<td>6%</td>
<td>13%</td>
</tr>
<tr>
<td>Secure or very secure</td>
<td>38%</td>
<td>74%</td>
<td>66%</td>
</tr>
</tbody>
</table>

• **General (2)**


_We believe open source software is a necessary requirement to build systems that are more secure.... opening the source of existing systems will at first increase their exposure... However, this exposure (and the associated risk of using the system) can now be determined publicly. With closed source systems the perceived exposure may appear to be low, while the actual exposure... may be much higher. Moreover, because the source is open... the period of increased exposure is short. In the long run, openness of the source will increase its security... [and] it allows users to make a more informed choice about the security of a system..._
References


1.3 Statistics about FOSS projects. 1.3.2 Statistics concerning FOSS projects


Fad? In J. Feller et al. (ed): Perspectives of free and open source software MIT press 2005


