To appear in Proc. Tenth Int. Software Metrics Symposium (Metrics 2004).

Open Source Software Development: A Case Study of FreeBSD

Trung Dinh-Trong and James M. Bieman
Software Assurance Laboratory
Computer Science Department
Colorado State University
Fort Collins, CO 80523 USA
{trungdt,bieman}@cs.colostate.edu

Abstract

A common claim is that open source software development produces higher quality software at lower cost than traditional commercial development. To validate such claims, researchers have conducted case studies of "successful" open source development projects. This case study of the FreeBSD project provides further understanding of open source development. The FreeBSD development process is fairly well-defined with proscribed methods for determining developer responsibilities, dealing with enhancements and defects, and for managing releases. Compared to the Apache project, FreeBSD uses a smaller set of core developers that implement a smaller portion of the system, and uses a more well-defined testing process. FreeBSD and Apache have a similar ratio of core developers to (1) people involved in adapting and debugging the system, and (2) people who report problems. Both systems have similar defect densities, and the developers are also users in both systems.

1 Introduction

Open source software (OSS) development receives much attention in both the trade press and from researchers [2, 3, 9, 10]. OSS systems share a number of characteristics [12]. These systems are developed by a large number of volunteer contributors. Some OSS projects are supported by commercial companies with paid participants in addition to many volunteers. Participants have the freedom to work on any part of the project. There are no assignments and deadlines. In general, developers do not create a system-level design, a project plan, or lists of deliverables.

Proponents claim that the quality of the OSS development is equivalent or even superior to traditional commercial development and that "many companies are drawn by the low cost and high quality of open source software" [14]. Reasons given for the advantages of the open source projects tend to relate to the notion of "freedom". Anybody can have a copy of the program and can contribute to the improvement of the system [6, 17], so that OSS developments "directly leads to more robust software and more diverse business models" [20]. Also, OSS developers can work without interference and in their own time, resulting in great creativity [14]. Overall, many claim that OSS is developed faster, cheaper, and the resulting systems are more reliable [6, 12, 20].

Others challenge the value of OSS, and do not believe in its long term success. Possible weaknesses of OSS development include a lack of a formal process [18], poor design and architecture [1, 14], and development tools (such as CVS) that are not comparable to those used in the commercial community [18].

Several research studies have evaluated claims about OSS. Schach et al. examine 365 versions of the Linux kernel and report that the number of lines of code has increased linearly with the version number, while the number of common couplings has increased exponentially [17]. These results suggest that Linux will become difficult to maintain, unless it is restructured.

Godfrey and Tu also studied the evolution of Linux by examining 96 kernel version [6]. They found out that although Linux is very large (over two millions lines of code), it has been growing at a "super-linear" rate for several years. Given that the growth of large commercial systems tends to slow down when the systems become larger, Godfrey and Tu's results suggest that OSS systems have a growth rate that is much greater than that of traditional systems.

Mockus et al. propose that key requirements for the success of an OSS project can be expressed as seven hypotheses [12]. These hypotheses were first developed through an empirical study of the Apache project, a *pure* OSS project — a project without major commercial support. After conducting the second study on a greater project, Mozilla, which is supported by a company, Netscape, the authors refined the hypotheses.

We can determine whether or not the hypotheses represent general rules by examining other open source systems. This paper reports the result of a case study that repeats the Mockus et al. study on a different OSS development project, the FreeBSD project. FreeBSD was selected for this study because it is a "successful" OSS project — the project is an order of magnitude older than either Apache or Mozilla, and the project website shows a list of about 100 software vendors who offer commercial product and/or services for FreeBSD. In addition, information concerning the FreeBSD development process is open to the public through an email archive, a bug database, and a CVS repository. We examine enough information to assess five out of six hypotheses posed in Mockus et al.'s work.

2 The Hypotheses

The objective of the Mockus et al. case studies of the Mozilla and Apache projects was to understand the processes that are used to develop successful OSS and to compare their effectiveness with commercial development [12]. Mockus et al. found that the Apache project was managed by an informal organization consisted entirely of volunteers. Every Apache developer had at least one other job, so that they could not work full-time on the project. On the other hand, the Mozilla project was managed by a commercial company, Netscape, and some of the developers worked on the project full-time and for pay. Nevertheless, the processes used in these two projects had many traits in common. The identification of these common traits led to seven hypotheses about successful OSS development:

H1: "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."

H2: "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."

H3: "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."

H4: "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."

H5 "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."

H6: "In successful open source developments, the developers will also be users of the software."

H7: "OSS developments exhibit very rapid responses to customer problems."

3 Study Method

3.1 Research Questions

To conduct the experiments about OSS development process, Mockus et al. [12] answered the following questions about the Mozilla and Apache and their development processes:

1. "What were the processes used to develop Apache and Mozilla?"
2. "How many people wrote code for new functionality? How many people reported problems? How many people repaired defects?"
3. "Were these functions carried out by distinct groups of people, that is, 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?"
4. "Where did the code contributors work in the code? Was strict code ownership enforced on a file or module level?"
5. "What is the defect density of Apache and Mozilla code?"
6. "How long did it take to resolve problems? Were high priority problems resolved faster than low priority problems? Has resolution interval decreased over time?"

1. How many roles are involved in coding (for example, changing .c and .h files)? (I know of three roles: “developers” (AKA “committers”), “core developers” and “Release Engineer team”.)
2. How does one become a “developer” or “committer”?
3. How does one become a “core developer”? When and how do you vote for new developers?
4. How does a normal person contribute code? Does he/she need to submit his/her code to a committer?
5. How does the Release Engineering Team check the code?
6. How does a “committer” decide to commit a new piece of code to the *Current* code base? How does he decide that this code is stable enough to put into *Stable* code base?
7. What is the difference between the role of a “developer” and a “core developer”? What privileges does a “core developer” have that a “developer” does not have?
8. How does one decide what to do next when fixing a bug and when adding new functionality?

Figure 1. Questions sent to each of the FreeBSD Core Team members.

We sought answers to the same questions concerning the FreeBSD project. Out of these six questions, we obtained data to answer the first five. To answer the questions about the development process, we studied the documents provided in FreeBSD project website [16]. To help answer our questions, one member of the Core Team (this term is used in FreeBSD to refer to the core developers) provided us a hidden Web address of the “FreeBSD internal pages” (it is hidden in the sense that we cannot find any way to navigate to this page from the FreeBSD home page). The FreeBSD internal pages provided useful information about the guideline and requirements for the FreeBSD *committers*. In the FreeBSD project, *committers* play a role that is similar to *developers* in the Mozilla and Apache project. In addition, committers may be elected to the *Core Team*. We also sent each member of the current (at the time) nine Core Team members a set of questions, which are displayed in Figure 1. Four Core Team members responded with their answers.

3.2 Data Sources

In order to answer the quantitative research questions at the beginning of Section 3.1 (questions one through four), we obtained the necessary data from the project CVS repository, which includes the bug report database as well as the email archive. The CVS repository contains all of the code and related documentation that is committed to the project from 1993 until the present. The bug report database contains information describing all reported problems, as well as the status (such as fixed, under test, or open) of each problem. Each bug report is called a PR, and assigned a reference number. The email archive contains every email message exchanged between the developers since 1994. Due to the nature of the open source software, the locations of the developers are distributed world wide, and they rarely meet with each other. Developers generally exchange information about the project via email. According to Mockus et al. [12], email archives record all information about an OSS project. However, the main disadvantage of using email archives as a primary source for information is that the format is usually informal.

CVS Repository. FreeBSD, like many OSS projects, uses a Concurrent Version Control Archive (CVS) as the version control tool. Whenever a developer needs to change the code base, he or she can check out the corresponding file, make the change and check the file back into the CVS. CVS not only stores the latest version of the code base, but also stores the history of the code that is changed [5].

FreeBSD developers maintain two branches of the code: the *Current* branch contains all of the ongoing projects (many are under test and not ready to be released) related to FreeBSD, and the *Stable* branch, which is the official released version of FreeBSD. In this research, we retrieved information about FreeBSD code from the Current branch. The FreeBSD CVS repository is available to the public and anybody can make a mirror copy of it. We retrieved a copy of the CVS in early April, 2003 and used the CVS command “log” to retrieve information about all updates made to the code base (the “src” directory) from the start of the repository until the day we downloaded it. Each update includes the time of the update, the corresponding file, the number of lines deleted and added, the login name of the developer who committed the change, and a short description of the change. The particular problem (PR) that was fixed is reported in the PR database, which almost always contains the PR reference number. We wrote a program to scan the log to record developer login names and count the number of updates made by each developer. We assume that each of the developers uses just one login name to commit the code. We distinguish between the code updated to fix problems, and the code updated or added to implement a new feature.

We assume that entries that contain the string "PR" represent code to fix problems, and the other entries represent code that implements new features. Each change to a file is a *delta*.

Developer Email Archive. The FreeBSD project maintains many different email lists for various purposes. We studied all of the email messages that were sent to freebsd-bugs@FreeBSD.ORG to report problems. Out of 36,118 PRs recorded in the database, 16,115 were also recorded in this email list. We used this list to extract names of those who reported problems and the number of problem reported by each person. Although we did not study the exhaustive list of PRs, the retrieved data was still enough for us to test the hypotheses 3, which relates to the number of people that report problems.

Bug Report Database. The FreeBSD project records every reported problem using a GNATS database. Further information about the GNATS database is available from the GNU website [15]. Each report contains a description of the problem, the name of the reporter, the reported date and other information. For our research, we only retrieved the total number of bugs reported. Other information describing bugs is retrieved from the Developer email archive.

3.3 Data For Commercial Projects

For this paper, we reused the data describing commercial projects that was provided in the Mockus et al. study [12]. These projects are denoted as projects A, C, D and E. According to Mockus et al., "project A involved software for a network element in an optical backbone net work" and "projects C, D and E represent Operations Administration and Maintenance support software for telecommunication products". Mockus et al. also claim that the processes used to develop these systems was very well-defined.

4 Results

First we examine the collected data to answer the research questions, then we evaluate the hypotheses.

4.1 Answers to the Research Questions

4.1.1 Q1: "What was the process used to develop FreeBSD?"

FreeBSD is an operating system derived from BSD UNIX, the version of UNIX developed at the University of California, Berkeley. According to the FreeBSD developers [16], FreeBSD can run on x86 compatible, DEC Alpha, IA-64, PC-98 and UltraSPARC architectures. As described by Godfrey and Tu [6], an OSS project can be forked into an alternative OSS project when a subset of developers are unhappy with the "official" or main branch. The BDS Unix project is an example of this phenomenon, which forked into FreeBSD, OpenBSD and NetBSD. The FreeBSD project started in 1993. At the time of this study in 2003, there were 35 released versions (from 1.0 to 5.0).

FreeBSD maintains two branches of its code base. The *Current* branch consists of on-going projects, which need to be tested and are still unstable. The *Stable* branch is mature and comparably well-tested; releases are formed from the Stable branch.

Roles and Responsibilities. Contributors to the code base play one of three main roles: *Core Team member*, *committer*, and *contributor*. In a *pure OSS process* every developer (including Core Team members) and contributor is a volunteer and most likely has a paid job. Thus, most volunteers contribute to the FreeBSD project part-time, perhaps during nights or weekends. The Core Team is a small group of senior developers who are responsible for deciding the overall goals and direction of the project. The Core Team assigns privileges to other developers and resolves conflicts between developers. Core Team members are also developers — they contribute code to the project. Usually, a Core Team member may also have to manage some other specific areas such as documentation, release coordination, source repository and GNATS database.

When the FreeBSD project began, the Core Team consisted of thirteen members. According to the current by-laws of the project, the Core Team consists of seven to nine members who are elected to two year terms by active committers. Any active committer (active within the latest twelve months) can be a candidate for the membership of the Core Team. An early election is called if the number of Core Team members drops below seven.

Committers are developers who have the authority to commit changes to the project CVS repository. According to the by-laws of the project, a committer must be active within the past 18 months. Otherwise, the Core Team can revoke the committer's privileges. An active contributor can be nominated to be a committer by an existing committer. The Core Team can award committer privileges to a candidate. A new committer is assigned a mentor, who supervises the new committer until he or she is deemed to be trustable and reliable.

Contributors are people who want to contribute to the project, but do not have committer privileges. They usually begin to contribute by registering on the project mailing lists so that they can be informed about the activities. Contributors may test the code, report problems, and also suggest solutions.

Identification of work to be done. There are two main tasks that need to be done in any project: developing new features and fixing defects. Although it is the responsibility of the Core Team to decide the direction of the project, it is hardly happened in reality. Instead, according to Core Team members, individual committers usually determine their own project, for example, adding a feature. Sometimes, committers may form teams to work on large projects.

Project defects reported by contributors are tracked using the GNATS database. There are three ways to report a problem: (1) use the *send-pr* command of FreeBSD, (2) use a web-based submission form provided in the FreeBSD web page [16], or (3) send an email to Freebsd-bugs@FreeBSD.org. If one of the first two methods are used, the PR will be automatically added to the GNATS database. The third method (sending email) is less preferable, because a committer must personally process the email — he or she must manually read the message and add a PR to the database. Also, emailed problem reports may be ignored because of the huge volume of messages received each day.

A PR has the following fields: reference number, responsible committer, submitted date, severity, reporter name, state, and description. A PR may be in one of several states: open (just submitted, no effort to fix it yet), analyzed, feedback, patched, suspended or closed (the bug is fixed or cannot be fixed at all). The FreeBSD webpage also provides a guideline for problem reporters, to help them to make the description as informative as possible.

Assigning and performing development work. A committer can search through the open PRs in the bug report database and assign a PR to himself, or to another committer that should be able to solve the problem. Many PRs may have solutions suggested by the person reporting the defect. Contributors can scan the PR database and propose the solutions to open PRs. Although contributors do not have access to change the code base, they can test solutions in their own copy of the code, and send the solution to the corresponding assigned committer. A contributor may also send a solution to the email list as a follow-up message. The committer responsible for the PR can communicate with the bug reporter and all interested contributors to discuss the problem and possible solutions. A committer may solve the problem directly or use a solution proposed by a contributor. After testing a proposed solution, the committer can insert the solution to the Current code base. Sometimes a committer will insert a solution to the Stable code base directly, if the problem does not exist in the Current code base. If, over time, no new defects related to the fix are reported, the committer can close the problem.

To implement new features, a committer (or a team of committers) writes code, tests it and then adds the code to the Current code base. Before the release date of a new Stable release, a committer can decide to merge their new code to the Stable version.

Testing. Committers must test their own code (with the help of interested contributors) before they can commit their code to the Current branch. The thoroughness of testing depends on the judgment and the expertise of a committer. Also, before merging the code to the Stable branch, a committer can perform a process called *merge from current (MFC)*. After developing new code, committers set a countdown period and ask other developers and contributors to test the code. If no new defect found at the end of the countdown, a committer may assume that the code is acceptable.

Another form of testing may be considered a form of system test. Before releasing a new version, a release candidate is introduced to the committers and contributors. The release candidate is tested and fixed until a Release Engineer Team decides that the system is ready. However, no committer is assigned to be a tester; volunteers test the release candidates.

Code inspection. A committer may want to commit a piece of code to a file or portion of the system that is the responsibility of another committer, the *active maintainer*. The active maintainer must review and approved the new code before it is added to the code base.

A developer can determine the active maintainers of a location by using the CVS command log, which will indicate who is currently changing the code. Committers may assigned themselves as active maintainers of a location by putting their name in a README file or a makefile.

A committer that plans to make a significant change to the code is expected to ask some other committers to review the code. Committers in the Release Engineer Team review code 30 days before a release date.

Managing releases. The Release Engineer Team manages FreeBSD releases. A Core Team member volunteer is the chief of the team. The other members of the team are volunteers selected from the committers. A new version of FreeBSD is released every four months using the following timetable:

- 45 days before the release date: the Release Engineer Team announces to every developer that they have a 15 day period to integrate their changes to the STABLE branch.

- During the 15-day period: committers will perform the MFC for their code.

- 30 days left: the Release Engineer Team announces a 15-day *code slush* period, during which the team will review the added code to the previous release. During the code slush period only limited changes are allowed such as bug fixes, documentation update, security-related fixes, minor changes to device drivers, and other changes that are approved by the Release Engineer Team.
- 15-days left: the code base enters a *code freeze* period. During code freeze, a release candidate is built every week and distributed for a widespread testing, until the final release is ready. The only changes are allowed during the code freeze period are serious bug fixes and security repairs.

Comparing the FreeBSD process to that of Apache and Mozilla. It is more appropriate to compare FreeBSD with Apache rather than Mozilla, since both Apache and FreeBSD are "pure OSS" projects. In contrast, Mozilla is a hybrid project — it is supported by a commercial company with paid participants. The process used to develop FreeBSD is very similar to that of Apache. Both projects use the same or very similar (1) developer roles, (2) concepts of code ownership, and (3) mechanisms to assign tasks to developers. However, the FreeBSD project has a more well-defined testing process than the one used in the Apache project. FreeBSD includes a form of system testing during the "code freeze" period, while Apache does not.

4.1.2 Q2: "How many people wrote code for new functionality? How many people reported problems? How many people repaired defects?"

To determine the number of people involved in writing code for FreeBSD, we used the CVS "log" command to retrieve the user names of the committers who update the code to the *src* directory. The *src* directory contains the code in the Current branch of FreeBSD; it does not include any application code provided by third parties. A total of 354 committers added code to the *src* directory from 1993 to April 2003.

Following the steps in Mockus et al. case study [16], we counted the number of distinct people who contribute code to fix defects and the number of people who contributed code for new features. The *src* directory contains source code (files with *.h* or *.c* extensions), text files such as *README* files and shell scripts. A total of 224 committers checked in 11,406 deltas to fix problems. Among these deltas, 5,893 deltas are source code files checked-in by 197 committers. 337 committers checked in 516,540 deltas for new features; 301,969 of these deltas are source code files checked in by 290 committers.

An examination of the archive of the email list freebsd-bugs@FreeBSD.ORG determined the names of contributors who reported problems. A total of 6082 unique individuals (based on names) reported 16,115 problems. The email list probably does not include all bug reporters, since there are 36,118 PRs in the GNATS database. Because we did not find an exhaustive list of bug reporters, we conclude that there are at least 6,082 bug reporters. This is enough data for us to evaluate the corresponding hypothesis (hypothesis 3).

4.1.3 Q3: "Were these functions carried out by distinct groups of people, that is, 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?"

The results from Mockus et al. [12], indicate that a small group of less than 15 committers will commit more than 80% of the new source code (code for new features). However, our results, shown in Figure 2, show that the top 15 committers contribute only 56% of the deltas to new source code; it took the 50 top committers to contribute 80%. We also found that a total of 36 people were members of the Core Team at some period, and 36 top developers (not all of them are in the Core Team) contributed about 75% of new source code. Note that this data is for the deltas that affected source code (*.h* and *.c* files). We performed the same analysis on the deltas that affected all files (not just source code) in the *src* directory and got similar results.

Figure 3 shows the cumulative distribution of the source code changes that were checked-in to fix defects. The top 15 contributors checked-in about 40% of the deltas, and the top 50 developers contributed about 70% of the fixes. This result is somewhat similar to the Apache case study [12]. A small number of committers added most of the new features, but the effort required to fix defects is more evenly distributed.

Among the 6,082 individual reporters reporting 16,115 defects, the top 15 reporters reported between 49 and 100 problems each, which represents 0.6% of the PRs. There were 3,370 reporters who reported one bug, 1875 reporters who reported two bugs, and 447 who reported three.

4.1.4 Q4: "Where did the code contributors work in the code? Was strict code ownership enforced on a file or module level?"

The study of the Apache project [12] suggests that there is no strict code-ownership involved in OSS developments. The result of our study strongly supports this suggestion. Our study shows that among 26,048 *.c* and *.h* files, only 30% of the files were modified by one committer, 25% by two committers, 15% by three committers, and 8% by ten or more committers. One file was changed by 74 developers.

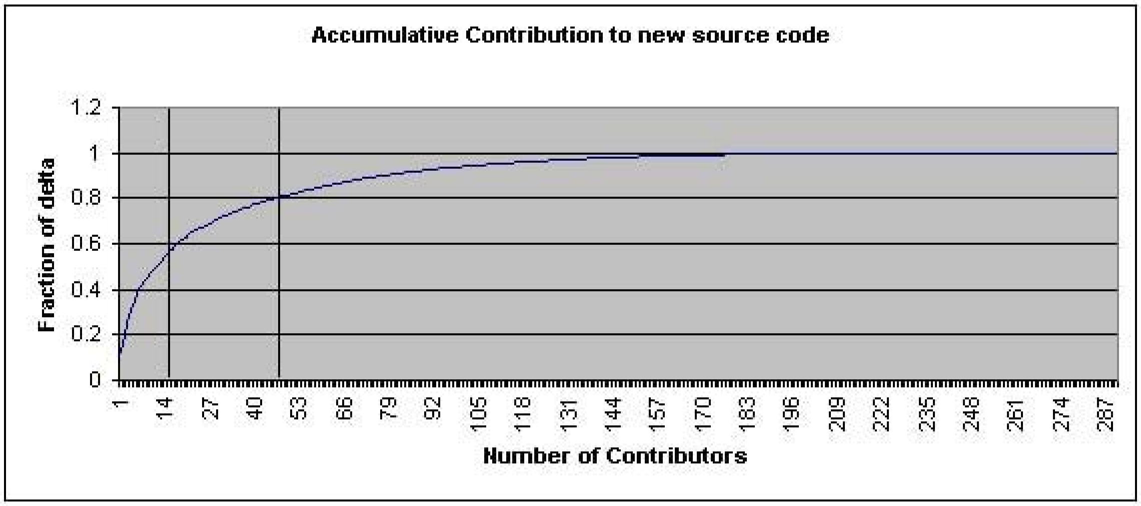

Figure 2. Distribution among developers (committers) of source code deltas to add new features.

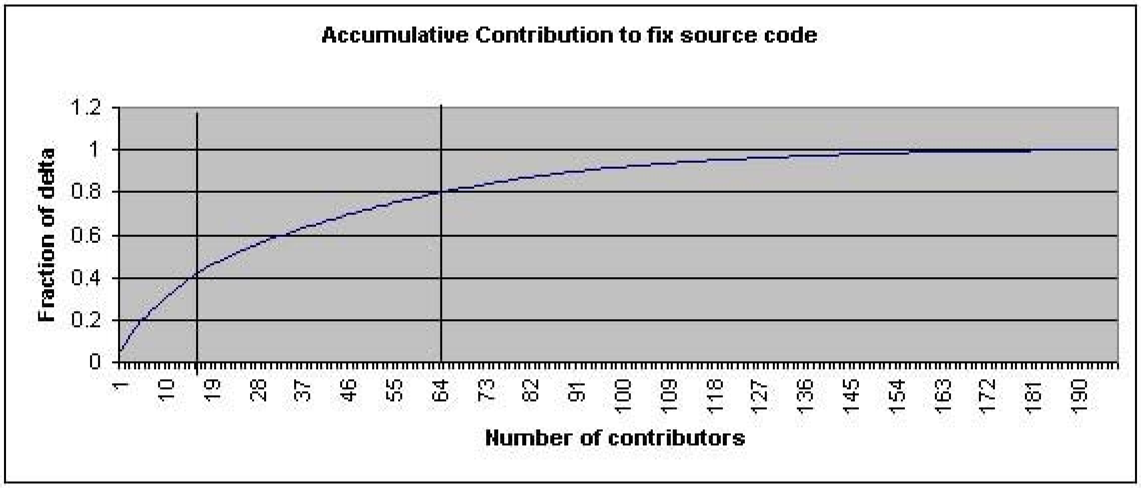

Figure 3. Distribution among developers (committers) of source code deltas to fix errors.

In fact, every committer has the privilege to make any change to any file in the system. Code ownership in FreeBSD does not exist. Instead, FreeBSD committers are only required to respect each other by asking for code review before committing code to files that are actively maintained by other committers.

4.1.5 Q5: What is the defect density of FreeBSD code?

Following the approach used by Mockus et al. [12], we measure the number of defects per thousand lines of code added and per thousand deltas. The result is shown in Table 1. We also compare the result with the defect densities in Apache and the four commercial software systems as reported by Mockus et al. The four commercial projects are denoted as projects A, C, D and E. For the commercial software systems, we show both the defect density after release and after the feature test. There is no data for the post-feature defects in Project A. The results indicate that the defect density of FreeBSD (and also Apache) is smaller than the commercial systems after the feature tests.

4.2 Evaluating the Hypotheses

We examine each hypothesis concerning successful OSS projects in order:

H1: A core of 10 to 15 developers in an OSS project will control the code base, and create approximately 80% or more of the new functionality.

In FreeBSD, a total of 36 people were members of the Core Team at some time over the period studied. These

Table 1. Defect densities in FreeBSD, Apache, and four commercial systems.

Measure	FreeBSD	Apache	A	C	D	E
Post-release defects/KLOC	3.35	2.64	0.11	0.1	0.7	0.1
Post-release defects/Kdelta	68.39	40.8	4.3	14	28	10
Post-feature defects/KLOC	3.35	2.64	$*$	5.7	6.0	6.9
Post-feature defects/Kdelta	68.39	40.8	$*$	164	196	256

Core Team members contributed 75% of the new functionality deltas. The core team contained 13 members at the beginning of the project; later the team size was restricted to between seven and nine members. We might infer that the larger sized group was unwieldy, and that is why the size of the Core Team was reduced.

Thus, the size of the Core Team was generally smaller than that given in H1, and the Core Team members contributed slightly less of the functionality. We suggest that H1 is overly proscriptive. A more realistic hypothesis is that there will be a core of fifteen or fewer developers at any one time, and this core will contribute most of the new functionality.

H2: In projects where more than 15 people contribute 80% of the code, some formal arrangements will be used to coordinated the work.

FreeBSD had a smaller set of core developers, so we cannot evaluate H2. We do note that FreeBSD does use a set of rules for determining the set of committers and core developers, and there are guidelines for "assigning" tasks, testing, and inspections. However, these rules appear to be informal.

H3: A group that is much larger than the core will repair defects, and an even larger group will report problems.

The FreeBSD project was consistent with the relationship in H3 between the relative size of Core Developers, those who repair defects, and those who report problems.

H4: OSS projects without many contributors, in addition to the core, may create new functionality, but will fail because of a lack of defect discovery and repair capability.

Since FreeBSD did have many contributors, we could not evaluate H4.

H5 Defect density in OSS releases will be lower than commercial code that has only been feature-tested.

The results from FreeBSD are consistent with H 5. However, we note that the four commercial projects A, C, D and E were chosen by Mockus et al. because they were deemed as comparable to Apache, but Apache may be not comparable to FreeBSD. Thus, our informal judgement is that we cannot reject the null hypothesis for H5.

H6: Developers will be users of the software.

The developers of FreeBSD were clearly users, thus supporting H6.

H7: There will be rapid responses to customer problems.

Unfortunately, we do not have enough data yet to evaluate H7.

5 Threats to Validity

Like most case studies there are threats to validity. We assess four types of threats to validity: construct validity, content validity, internal validity and external validity. Construct validity refers to the meaningfulness of measurements [7, 13] — do the measures actually quantify what we want them to? To validate the meaningfulness of measurements, we need to show that the measurements are consistent with an empirical relation system, which is an intuitive ordering of entities in terms of the attribute of interest [4, 8, 11]. The variables in this study include counts of defects, deltas, and the size of the different project groups match those used in the Apache study. A count of the deltas in the code base is an intuitive measure of the relative contribution of project members, and a count of defects is an intuitive indicator of code quality. However, not all deltas or defects are equal, but the large number of deltas and defects should minimize the impact of the variability of delta size or defect severity. Counts of the number of members in the different OSS development groups do not appear to represent any threat to construct validity.

Content validity refers to the "representativeness or sampling adequacy of the content... of a measuring instrument" [7]. The content validity of this research depends on whether the individual measures of deltas and defects adequately cover the notion of the relative contribution of developers and code quality respectively. The count of deltas quantifies only one aspect of relative contribution. We only look at one quality attribute, defects. It is always difficult obtaining quantitative indicators of all aspects of quality. One real concern is that the qualitative understanding of the process used is based on informal dialogue with only a subset of Core Developers. A representative sample of all Core Developers and committers might offer different insights. Also, there is an implicit judgement in this research. That is that all of the OSS projects involved (FreeBSD, Apache, and Mozilla) may be considered successful OSS developments. One may always debate such judgements.

Internal validity focuses on cause and effect relationships. The notion of one thing leading to another is applicable here and causality is critical to internal validity. This study did not really lend itself to a statistical analysis of correlation between variables. In a sense, we did not have a control — an OSS system that is a failure. In addition, the hypotheses were expressed as necessary conditions for success, but they are not sufficient conditions. A project may satisfy all of the organizational conditions, yet fail due to other external reasons. For example, there may turn out to be little user interest in the OSS product. Thus, a study of failed project would shed little light on the hypotheses. Ultimately we can find conclusive evidence only when the hypotheses can be clearly rejected — when a data from a successful OSS project contradicts the hypotheses. An intuitive argument does support a causal relationship between OSS project organization and success.

External validity refers to how well the study results can be generalized beyond the study data. An adequate study should be valid for the population of interest [19]. A general problem with case studies is that they may or may not apply to other projects. One objective of this project is to add another piece of evidence to that collected by Mockus et al. [12]. Thus, this study has reduced the threats to the validity of the earlier study.

There are some specific threats to the validity of this research. There is a lack of information about the commercial systems. In order to evaluate the efficiency of OSS development, we compare the defect density of FreeBSD project with commercial projects A, C, D and E, which were provided in the Apache and Mozilla case studies. [12]. These commercial projects were chosen so that they are comparable to Apache, which may not be completely comparable to FreeBSD as shown in Table 2. However, our results show that the defect density of FreeBSD is higher than that of Apache, so we suggest that commercial projects that are "comparable" to FreeBSD should also have higher defect densities than those of projects A, C, D and E. Still we see that the defect densities of A, C, E and D are much higher than the defect density of FreeBSD. Because of that, we conclude that our results support hypothesis H5.

Another threat is that we studied only 16,115 out of a total of 36,118 PRs to extract the names of the problem reporters. The key result is that the number of problem reporters in FreeBSD is larger than the number of developers (committers) by an order of magnitude (this result supports hypothesis H3). Obviously, if we examined all PRs, the number of problem reporters would have been even larger, and it will not effect our conclusion at all.

Table 2. Comparisons between FreeBSD, Apache, and four commercial systems.

Project	K Deltas	Years	Developers
FreeBSD	528	10	354
Apache	18	3	388
A	129	3	101
C	2.8	1.3	17
D	0.7	1.7	8
E	2.4	1.5	16

6 Conclusions

The goal of this study was to better understand the nature of Open Source software development, and to see if prior case study results can be obtained in a study of another system.

This study repeated the work of Mockus et al. [12], a study of Apache and Mozilla, on FreeBSD. We conclude that the FreeBSD process is fairly well-defined and organized; project members understand how decisions are made, and it appears fairly effective.

We examined whether the FreeBSD project supported six hypotheses proposed by Mockus et al. We gathered enough data to evaluate hypotheses H1, H3, H5 and H6. Our data supports hypotheses about the relationship between the number of core developers, developers and contributors (H3), the defect density of OSS (H5), and that OSS developers are also users (H6). Our results show that the hypothesis about core developers (H1) is too restricted — our data would support a relaxed version of H1. We cannot test hypotheses H2 and H4 due to the nature of FreeBSD. Hypothesis H7 concerning the time to respond to customer problems was not tested due to a lack of data.

Finally, we feel that, by studying existing, on-going software projects, researchers will gain insights into the nature of software development that can lead to improvements in development methods.

Acknowledgements

We would like to thank FreeBSD Core Team members W. Peters, J. Baldwin, M. Losh, and M. Murray for providing project information. We would also like to thank Core

Team member R. Watson and FreeBSD developer J. Gibbs for helping us to mirror the FreeBSD CVS repository.

This material is based in part on work supported by the U.S. National Science Foundation under grant CCR-0098202. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] D. Cooke, J. Urban, and S. Hamilton. Unix and beyond: An interview with Ken Thompson. *IEEE Computer*, 32(5):58–62, 1999.

[2] J. Feller. Meeting challenges and surviving success: The 2nd workshop on open source software engineering. *Proc. 24th Int Conf. Software Engineering (ICSE-24)*, pages 669–670, 2002.

[3] J. Feller, B. Fitzgerald, and A. Hoek. Making sense of the bazaar: 1st workshop on open source software engineering. *ACM SIGSOFT Software Engineering Notes*, 26(6):51–52, 2001.

[4] N. Fenton and S. Pfleeger. *Software Metrics - A Rigorous and Practical Approach Second Edition*. Int. Thompson Computer Press, London, 1997.

[5] K. Fogel. *Open Source Development with CVS (1st Ed.)*. Coriolis Open Press, `http://cvsbook.red-bean.com/`, 1999.

[6] M. Godfrey and Q. Tu. Evolution in open source software: A case study. *Proc. Int. Conf. Software Maintenance (ICSM)*, pages 131–142, 2000.

[7] F. Kerlinger. *Foundations of Behavioral Research, Third Edition*. Harcourt Brace Jovaonvich College Publishers, Orlando, Florida, 1986.

[8] D. Krantz, R. Luce, P. Suppes, and A. Tversky. *Foundations of Measurement*, volume I Additive and Polynomial Representations. Academic Press, New York, 1971.

[9] T. Lawrie and C. Gacek. Issues of dependability in open source software development. *ACM SIGSOFT Software Engineering Notes*, 27(3):34–37, 2002.

[10] A. Lonconsole, D. Rodriguez, J. Borstler, and R. Harrison. Report on Metrics 2001: The science & practice of software metrics conference. *ACM SIGSOFT Software Engineering Notes*, 26(6):52–57, 2001.

[11] J. Michell. *An Introduction to the Logic of Psychological Measurement*. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey, 1990.

[12] A. Mockus, T. Fielding, and D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Software Engineering and Methodology*, 11(3):309–346, July 2002.

[13] J. Nunnally. *Psychometric Theory, Second Edition*. McGraw-Hill, New York, 1978.

[14] G. Perkins. Cultural clash and the road to world domination. *IEEE Software*, 16(1):23–25, 1999.

[15] G. G. Project. Gnats (version 4.0). `http://www.gnu.org/software/gnats/`.

[16] T. F. B. Project. FreeBSD (version 5.0), [computer software]. `http://www.freebsd.org/`, 2003.

[17] S. Schach, B. Jin, D. Wright, G. Heller, and J. Offutt. Maintainability of the Linux kernel. *IEE Proceedings — Software*, 149(1):18–23, February 2002.

[18] G. Wilson. Is the open source community setting a bad example? *IEEE Software*, 16(1):23–25, 1999.

[19] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

[20] M. Wu and Y. Lin. Open source software development: An overview. *IEEE Computer*, 46(6):33–38, 2001.