The FreeBSD Project: A Replication Case Study of Open Source Development

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

***Abstract*— Case studies can help to validate claims that open source software development produces higher quality software at lower cost than traditional commercial development. One problem inherent in case studies is external validity — we do not know whether or not results from one case study apply to another development project. We gain or lose confidence in case study results when similar case studies are conducted on other projects. This case study of the FreeBSD project, a long-lived open source project, provides further understanding of open source development. The paper details a method for mining repositories and querying project participants to retrieve key process information. 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 (1) a smaller set of core developers — developers who control the code base — that implement a smaller percentage of the system, (2) a larger set of top developers to implement 80% of the system, and (3) a more well-defined testing process. FreeBSD and Apache have a similar ratio of core developers to people involved in adapting and debugging the system, and people who report problems. Both systems have similar defect densities, and the developers are also users in both systems.**

I. Introduction

Both the trade press and researchers have examined open source software (OSS) development [3, 4, 12, 13]. The key attribute of OSS development is unique in that these systems are developed by a large number of volunteers. However, some OSS projects are supported by companies with paid participants, in addition to many volunteers. Unlike most commercial development, project 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 of OSS development claim that the quality of OSS development is equivalent or even superior to traditional commercial development and "many companies are drawn by the low cost and high quality of open source software" [21]. Claimed advantages of OSS development tend to revolve around the notion of "freedom" — anybody can have a copy of the program and can contribute to the improvement of the system [9,14,25,26], so that OSS development "directly leads to more robust software and more diverse business models" [29]. Since everybody can access and review anybody else's work, developers can learn from each other and improve their overall software development skill [14]. Also, OSS developers can work without interference and in their own time, resulting in great creativity [21]. We find many claims that OSS is developed faster, cheaper, and the resulting systems are more reliable [6, 9, 14, 17, 19, 24, 26, 29].

Others challenge the value of OSS, and question its long term success. Possible weaknesses of OSS development include a lack of a formal process [27], poor design and architecture [1, 21], and development tools (such as CVS) that are not comparable to those used in commercial development [27]. Messerchmitt [15] argues that OSS development will not be effective for software systems with a majority of the users that are not programmers. He explains that, since the developers of such systems are not users, the developers will not understand the users' needs, and OSS development mechanisms lose their advantages.

Although OSS development has been investigated, only a few studies are accompanied by empirical evidence. In one empirical study, Schach et al. examine 365 versions of the Linux kernel and report that the kernel size, in lines of code, has increased linearly with the version number, while the number of common couplings has increased exponentially [25]. 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 versions of the kernel [9]. They found that although Linux is very large (over two millions lines of code), it has continued to grow at a "super-linear" rate for several years. Given that the growth of large commercial systems tends to slow down when 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 in seven hypotheses [17]. These hypotheses were first developed through an empirical study of the Apache project, which is describe by Mockus et al. as a *pure* OSS project — a project without major commercial support. After conducting the second study on a larger project, Mozilla, which is supported by a company, Netscape, the authors refined the hypotheses.

One or two case studies cannot conclusively determine the nature of OSS development. There are just too many differences between application domains, project participants,

project support, and project lifespan. Understanding the nature of a software process such as OSS development will require many case studies. Our objective is to obtain further evidence to help determine whether or not the hypotheses represent general rules by examining other open source systems.

Our goal is to identify (1) the common characteristics in the development processes of successful OSS projects and (2) the quality of software that was produced using these processes. Thus, we repeated the Mockus et al. study on a different OSS development project [2], the FreeBSD project, an open-source version of the Unix operating system. We selected the Mockus et al. study for replication because it addresses both the requirements of success in an OSS project and the quality of an OSS software product. Furthermore, Mockus et al. measure product defect density, an external software quality attribute. In contrast, other empirical studies of OSS development measure the rates of code growth [9] and the degree of common coupling [25], which are internal quality attributes. Ultimately, external quality attributes are most important as these are what users actually observe.

FreeBSD was selected for this study because it is a "successful" OSS project. FreeBSD is well-known — in July 2003 FreeBSD was used in almost 2 millions web sites with nearly 4 million host names [18]. The project website shows a list of about 100 software vendors who offer commercial products and/or services for FreeBSD. The FreeBSD development process is well defined, well documented and easy to access. Information concerning the FreeBSD development process is readily available through an email archive, a bug database, and a CVS repository. Moreover, FreeBSD is similar to Apache and Mozilla in technical complexity, size of user community, and continuing success. Thus, one would expect the hypotheses developed in the Mockus et al. study to apply to the FreeBSD project as well, if they are, in fact, accurate.

FreeBSD is mature and has been active since 1993. It has a longer history than either Apache or Mozilla. The long history allowed us to examine project activities over a nine year period, compared to the three year period of the Apache project studied by Mockus et al.

We were able to assess five of the seven hypotheses posed by Mockus et al. Our results support three of the hypotheses and suggest revisions to the others. Our data also supports stronger conclusions about the reliability of OSS. Mockus et al. suggest that the defect densities in OSS releases are lower than that of feature-tested commercial code. However, they did not find such an improvement in post-release defect densities. Our study finds that both feature-tested and post-release defect densities of FreeBSD are equivalent to that of the commercial software systems.

In this paper, we extend the results of our prior case study [2]. An analysis of top developers who contribute more than 80% of the code base reveals differences between the FreeBSD project and Apache. In particular, we compare each three year period of the FreeBSD project to the three years of Apache project data reported by Mockus et al. and find notable differences. A reexamination of the FreeBSD data allowed us to recompute data on defects to improve accuracy; we now include only confirmed code defects in the analysis. The updated analysis allowed us to proposed additional revisions to the hypotheses. We also provide a detailed description of the research tools that we used to extract data from the open source archives.

II. 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 that of commercial development [17]. 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."

III. Study Method

A. Research Questions

Mockus et al. [17] 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?"

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 the FreeBSD project website [23]. To help answer our questions, one member of the Core Team (this term is used in FreeBSD to refer to the group of developers that control the code base) 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 guidelines and requirements for the FreeBSD *committers*. In the FreeBSD project, *committers* play a role that is similar to *developers* in the Mozilla and Apache projects. 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.

After developing a draft description of the FreeBSD development process, we sent the description to the Core Team members and GNATS Administrator to verify the accuracy of our account. We received suggestions for minor revisions, which we used to improve the accuracy of the description.

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?

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

B. Data Sources

In order to answer the quantitative research questions at the beginning of Section III-A (questions one through four), we obtained the necessary data from the project CVS repository, the bug report database, and 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 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. [17], 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.

1) 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 [7].

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.

2) Developer Email Archive.: The FreeBSD project maintains many different email lists for various purposes. We studied all of the email messages sent to freebsd-bugs@FreeBSD.ORG to report problems. Out of 51,156 PRs recorded in the bug report database, 16,115 were also recorded in the freebsd-bugs email list. We used this list to extract names of those who reported problems and the number of problems reported by each person.

3) 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 [22]. Each report contains a description of the problem, the name of the reporter, the reported date and other information. The FreeBSD official website provides a GNATS

web based interface that allows one to query a set of bugs based on matching PR field values, including priority, state, severity, and class. The result of the query is a list of PRs with the following information: the status, the reported date, the PR tracker identification (ID), the person who takes the responsibility to fix the problem, and a short description of the problem. To find more information about a PR (i.e, the name of the PR submitter), one must follow the PR link to open a new page with the full description of the PR.

For our research, we used this web-based interface to search for the total numbers of the code-related PRs in the Stable branch and in the Current branch. We created queries concerning code-related PRs in the two branches, and counted the total number of PRs in the result. To query the code-related PRs only, we selected the PR class to be $sw-bug$. The other PR classes include $doc-bug$, $support$, $change-request$, $mistaken$, $duplicate$, $wish$, $update$, and $maintainerupdate$. A description of the problem classes, given to us by one of the FreeBSD GNATS administrators, is shown in Table I. To find the PRs in Stable branch only, every Stable release has the keyword "-STABLE" in its release name. Hence, to find the PRs in the Stable branch, we made a query to search for all PRs that contain this keyword. Note that every PR includes information about the name of the release that contains the problem.

TABLE I
THE DESCRIPTION OF PR CLASSES

PR class	Description
sw-bug	Problems that require a software (code) correction
doc-bug	Problems that require a documentation correction
support	Support problems or questions
change-request	Suggested changes in functionality
mistaken	Bad PR submissions, not a problem
duplicate	Duplicate submissions of another existing PR
wish	A wish list request
update	Non-maintainer requests to update/change software
maintainer-update	Maintainer requests to update/change software

C. Data Extraction Tools

Retrieving information manually from the FreeBSD CVS repository and email archive is not feasible due to the large size of the data sets. For example, the CVS repository records about 527,930 changes to the src directory. We developed a set of Java-based tools to extract useful data from the CVS and the email archive. The extracted data includes the number of committers and the number of deltas committed by each person. The following discussion provides an overview of the tools that we developed; an appendix contains further details.

1) Retrieving information from the CVS repository: One approach to retrieving information from the FreeBSD CVS repository is to use CVS commands through the internet. This approach can lead to consistency problems, since it requires repeated access to the repository several times to extract the desired data. During the process of accessing the repository, the repository can change as committers continuously modify source code and other documents.

The approach used in this project is to download the entire CVS repository and store a copy in our computer system, which represents a snapshot of the FreeBSD project history. We retrieved this CVS copy in early April, 2003, and used the CVS command "log" to retrieve information about all deltas committed to the code base (the "src" directory) from the start of the repository until the day we downloaded it.

Each delta includes the time of the delta, 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. We developed a set of tools to scan the log to record the number of people that contributed code, the number of changes committed by each committer, the total number of changes committed by all committers and the total number of lines of code added to the code base. We assume that each developer 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 descriptions of deltas to fix bugs contain the keyword "PR". Deltas without this keyword are assumed to be deltas that add new features. We also distinguish between deltas that contribute to code (files with ".h" and ".c" extension), and deltas to non-code files (such as readme files and script files).

2) Retrieving information from the email archive: We downloaded the FreeBSD email archive and stored a copy in our computer system. The email archive is structured into several directories, each has messages that are exchanged in an email list. The directory `pr` stores all messages that relate to bug reports. We assume that each PR in the email archive contains the name of the originator in a line started with the key word "Originator". We extracted the number of people who reported the bugs, and the number of bugs reported by each person. We retrieved this information using the same technique used to obtain the number of code contributors and the number of deltas committed by each contributor.

D. Data For Commercial Projects

For this paper, we reused the data describing commercial projects that was provided in the Mockus et al. study [17]. 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 network" 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 were very well-defined.

IV. RESULTS

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

A. Answers to the Research Questions

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 [23], FreeBSD can run on x86 compatible, DEC Alpha,

IA-64, PC-98 and UltraSPARC architectures. As described by Godfrey and Tu [9], 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 BSD 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.

a) Roles and Responsibilities: Contributors to the code base play one of three main roles: *Core Team member*, *committer*, and *contributor*. In an OSS project that is not commercially supported, 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 membership in 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.

b) 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 rarely happens 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 [23], 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 desirable, 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). The FreeBSD webpage also provides a guideline for problem reporters, to help them to make the description as informative as possible.

c) 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 have solutions suggested by the person reporting the defect. Contributors can scan the PR database and propose 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 into the Current code base. Sometimes a committer will insert a solution into 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 with the Stable version.

d) 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 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 defects are 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.

e) 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 approve new code before it is added to the code base.

A developer can determine the active maintainers of a code location by using the CVS command log, which will indicate who is currently changing the code. Committers may assigned themselves to be 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 code is expected to ask some other committers to review the changed code. Committers in the Release Engineer Team review code 30 days before a release date.

f) 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 updates, 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 widespread testing, until the final release is ready. The only changes allowed during the code freeze period are serious bug fixes and security repairs.

g) 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 not commercially supported 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.

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 code in 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 the Mockus et al. case study [23], we counted the number of distinct people who contributed 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 51,156 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).

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?":** A comparison of the IDs of the developers who fixed bugs to the IDs of those who added new features can answer the first part of the question. The analysis shows that 220 out of 354 committers added code to do both tasks. A comparison of the committers' names with bug reporters' names provides further insight. The comparison used committer names from the FreeBSD website, and the bug reporter names from the email archive. At least 183 committers also report bugs. We cannot determine if there are additional developers who report bugs, because we do not have the full list of bug reporters. Nevertheless, the data indicates that FreeBSD contributors do not primarily assume a single role. A FreeBSD committer can contribute code both to fix bugs and to add new features, as well as report errors.

The results from Mockus et al. [17], indicate that a small group of less than 15 committers committed more than 80% of the new source code (code for new features). However, our results, shown in Figure 2, indicate that the top 15 committers contribute only 56% of the deltas adding 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.

The FreeBSD project is of much longer duration than the

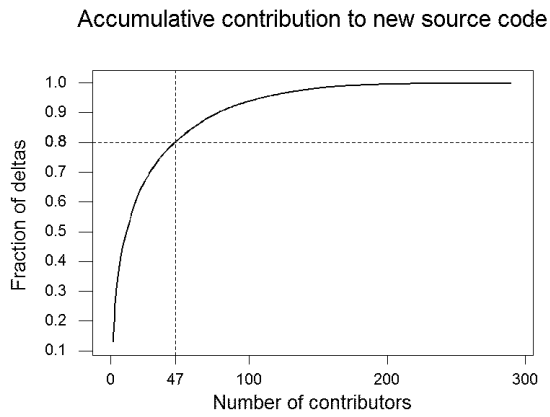

Fig. 2. Distribution of developer (committer) contributions of source code deltas adding new features.

Apache project. FreeBSD had been in operation for more than ten years, while Apache had been in operation for three years when the case studies were conducted. In order to verify the possibility that the length of the operation time dictates the number of the top developers, we analyzed the accumulative distribution of the FreeBSD source code changes to add new features in each three year period. We use three year periods to make it easier to compare FreeBSD to the Apache project, which had been in operation for three years when the Mockus et al. [17] study was conducted.

Figure 3 shows the accumulative distributions of the FreeBSD deltas in the 1994-1996, 1997-1999, and 2000-2002 periods:

- **1994-1996:** 209 committers added new source code in the 1994-1996 period. The top 15 committers added 69% of the new source code and the top 28 contributed 80%.
- **1997-1999** The number of committers who contributed new source code decreased to 161 during the 1997-1999 period. However, the top 15 committers still contributed about 69% of the new source code. The top 27 committers contributed 80% of the new source code.
- **2000-2002:** The number of new source code contributors increased to 263. However, the top 15 committers only contributed 59% of the new source code. It took the top 42 committers to contribute 80% of the new code.

These results suggest that the number of top developers — those who contribute 80% or more of the source code — in the FreeBSD project was always larger than that of the Apache project. The number of the top developers in the FreeBSD project is actually comparable to the number of the top developers in the Mozilla project (Table II). These results suggest that the number of top developers does not depends on the length of the time a project has been in operation.

Figure 4 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 [17]. A small number of committers added most of the new features, but the

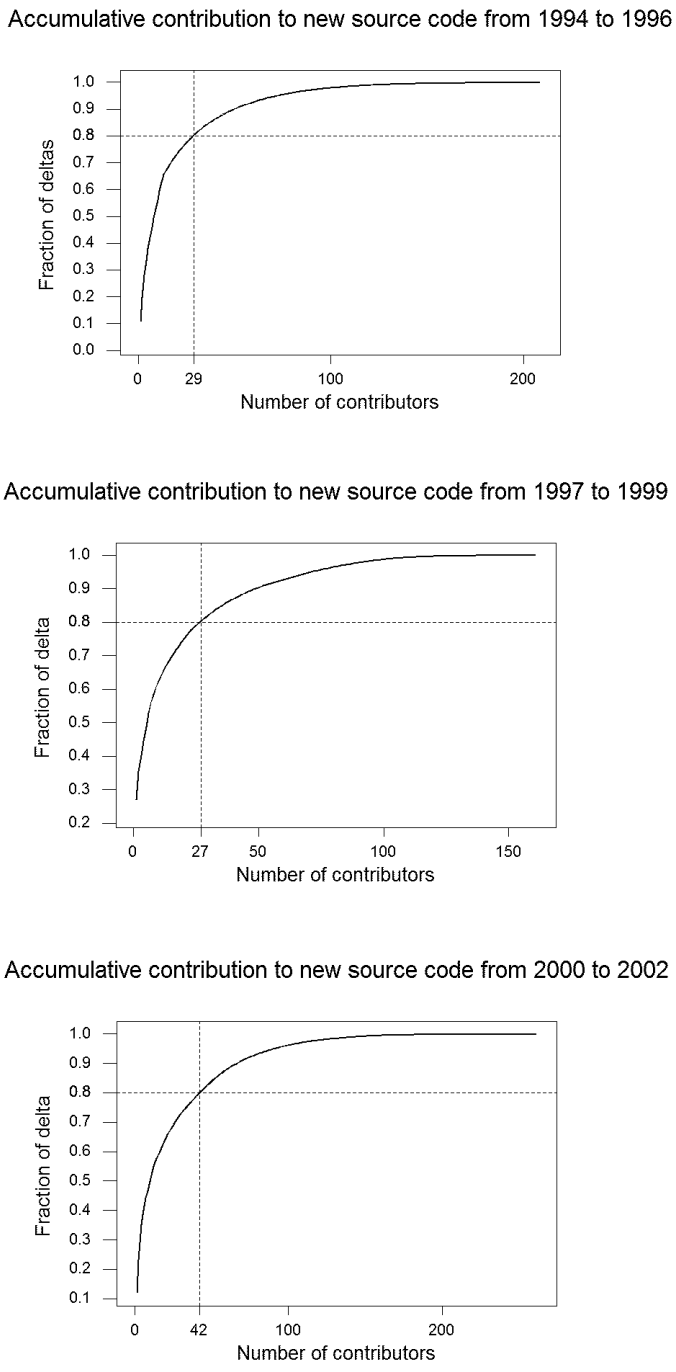

Fig. 3. Distribution of developer contributions of new source code during each three-year period.

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) **Q4: "Where did the code contributors work in the code? Was strict code ownership enforced on a file or**

TABLE II

THE NUMBER OF DEVELOPERS (DEV), TOP DEVELOPERS (TOPDEV — DEVELOPERS CONTRIBUTING 80% OR MORE OF THE SOURCE CODE, AND % OF DEVELOPERS WHO ARE TOP DEVELOPERS (% TOPDEV)

Project	Dev	Kdeltas	TopDev	% TopDev
FreeBSD (94-96)	212	180	28	13.2
FreeBSD (97-99)	161	172	27	16.7
FreeBSD (00-02)	265	174	42	15.8
Apache	388	15	18	4.6
Mozilla (layout)	174	42	35	20.1
Mozilla (js)	127	14	24	18.9
Mozilla (rdf)	123	12	26	21.1
Mozilla (network)	106	10	24	22.6
Mozilla (editor)	118	8	25	21.2
Mozilla (intl)	87	5	22	25.2
Mozilla (xpinstall)	102	5	22	21.5

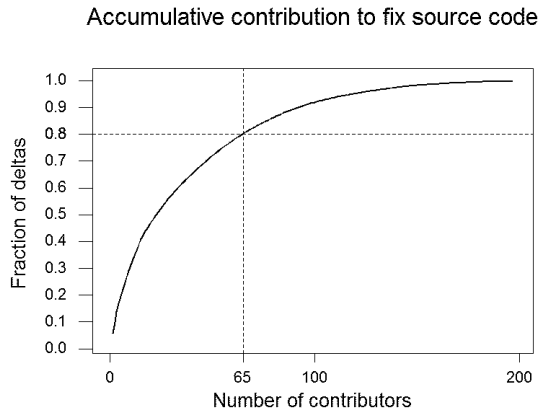

Fig. 4. Distribution of developer (committer) contributions of source code deltas to fix errors.

module level?" : The study of the Apache project [17] 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.

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 a code review before committing code to files that are actively maintained by other committers.

5) **Q5: What is the defect density of FreeBSD code?:** Following the approach used by Mockus et al. [17], we measure the number of defects per thousand lines of code added and per thousand deltas. Among the PRs recorded in the GNATS data base, we only counted the problems that require a correction to software. These PRs are categorized into the class "sw-bug". The "mistaken" and "duplicated" PRs are obviously not real problem reports. The "change-request", "update", and "maintainer update" PRs are also not reports about defects. The "doc-bug" PRs are the reports about defects in the documents, hence they are outside the scope of interest. We also counted the deltas and lines of code added to the src directory only, since this directory contains the Free-BSD code.

To determine the defect density after feature test, we counted the "sw-bug" occurrences in all FreeBSD branches. The defect density after release is determined by counting only PRs of the Stable branch. The result is shown in Table III. We compare the result with the defect density in the 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. There is no data for the post-feature defects in Project A.

The results indicate that after feature tests, the defect density of FreeBSD is similar to Apache and is lower than that of the commercial systems. The results also indicate that the defect density of FreeBSD after release is at least equivalent to or better than that of the commercial telecommunications software systems used in our comparison. The testing strategy used in the FreeBSD project appears to be as effective as those used in commercial practice, at least in the systems used in the study. The case study provides empirical support for claims that OSS systems are not less reliable than commercial software systems [19, 6].

TABLE III

DEFECT DENSITIES IN FREEBSD, APACHE, AND FOUR COMMERCIAL SYSTEMS.

Measure	FreeBSD	Apache	A	C	D	E
Post-release defects/KLOC	0.55	2.64	0.11	0.1	0.7	0.1
Post-release defects/Kdelta	11.17	40.8	4.3	14	28	10
Post-feature defects/KLOC	1.89	2.64	*	5.7	6.0	6.9
Post-feature defects/Kdelta	38.58	40.8	*	164	196	256

B. 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, the code base was controlled by the Core Team, an elected core of developers. A total of 36 people were members of the Core Team at some time over the period studied. These Core Team members contributed 47% 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, perhaps because the larger sized group was unwieldy.

Thus, the size of the Core Team was smaller than that given in H1, and the Core Team members contributed much less of the functionality. A possible explanation for why the FreeBSD Core Team members contributed a smaller portion of the functionality than the corresponding group in the Apache project is that FreeBSD is a larger project with more than ten times as many deltas as

shown in Table II. In addition, each FreeBSD Core Team member also had to spend time for other responsibilities such as coordinating the CVS mirror sites, managing the Release Engineering Team, or managing the mailing lists. We also examined a larger group of influential developers, the "top developers" — developers who contribute 80% or more of the code base, to match the 80% threshold used in the Mockus et al. study. As is shown in Table II, there were between 28 and 42 top developers over the ten year period studied. Thus, the number developers who contribute 80% of the code base is larger than that given in H1.

We suggest that H1 is overly proscriptive. A more realistic hypothesis will separate the core developers from the top developers as follows:

> H1': A core of fifteen or fewer core developers will control the code base and contribute most of the new functionality. A group of fifty or fewer top developers at any one time will contribute 80% of the new functionality. The group will represent less than 25% of the set of all developers.

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 more than 15 top developers throughout the project. In fact, in every three-year-period that we studied, the number of FreeBSD top developers was always more than 15. Compared to Apache, FreeBSD uses additional mechanisms to coordinate code contribution. These mechanisms include the use of release engineering teams, a system test procedure, a set of rules to assign committer and core team privileges. FreeBSD also has guidelines for "assigning" development tasks, unit testing, and inspection. However, most of these mechanisms are still informal. The system test procedure is informal; for example, it does not have a well-defined criterion to determine when the testing should stop. Also, FreeBSD lacks a mechanism to verify that developers follow the development guidelines. Therefore, our results do not fully support H2, and suggest a revised hypothesis:

> H2': As the number of developers needed to contribute 80% of OSS code increases, a more well-defined mechanism must be used to coordinate project work.

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 H5. Based on our results from the FreeBSD project, which separates unstable releases from final releases, we revise H5 as follows:

> H5': Defect density in OSS releases will be lower than commercial code that has only been feature-tested. If an OSS has a mechanism to separate unstable code from stable code or "official" releases, then the defect density of the stable code releases will be equivalent to that of commercial code after release.

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.

V. Threats to Validity

Like most case studies there are threats to validity. We assess four types of threats: construct validity, content validity, internal validity and external validity. Construct validity refers to the meaningfulness of measurements [10, 20] — 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 [5, 11, 16]. The variables in this study, which 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" [10]. 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

TABLE IV
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

not really lend itself to a statistical analysis of correlations 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 a 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 [28]. 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. [17]. 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 quality of OSS development, we compare the defect density of FreeBSD with commercial products A, C, D and E, which were provided in the Apache and Mozilla case studies. [17]. These commercial projects were chosen so that they are comparable to Apache, which may not be completely comparable to FreeBSD as shown in Table IV.

Another threat is that we studied only 16,115 out of a total of 51,156 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 affect our conclusion at all.

Finally, to avoid the replication of experimental errors caused by the specific research tools used by Mockus et al. [17], we independently developed our own data extraction and analysis tools.

Further research can reduce the threats to validity. Studies that include additional product quality indicators (other than defects), such as adaptability, will reduce threats to content validity. In addition, studies of the distribution of deltas, and a wider sampling of developers and PRs can reduce content validity threats. Clearly, additional case studies of both OSS and commercial development will reduce threats to external validity.

VI. 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 replicated in a study of another system.

This study repeated the work of Mockus et al. [17], 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, H2, 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) needs revision. FreeBSD uses a smaller group of core developers to control the code base. However, a larger group of top developers contribute 80% of the code base. Our results do not support the hypothesis about the need for a formal arrangement to coordinate the work (H2) — in the FreeBSD project more than 15 developers contribute 80% of the code, yet the guidelines for assigning task, testing and inspection are informal. We also extend the hypothesis concerning the defect density of OSS (H5). Our data suggests that the defect density after release of OSS is equivalent to that of the commercial software systems. We cannot test hypotheses 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.

Additional studies of existing, on-going open source and commercial software projects are clearly needed to gain further insights into the nature of software development. The following is a sample of open questions that can be answered only through further research:

1) What are the factors that distinguish between successful and unsuccessful OSS projects? Answering this question, relevant to evaluating Hypothesis 4, requires the development of criteria to identify unsuccessful OSS projects, and in depth studies of them.
2) Do the hypotheses hold with successful, but smaller OSS projects? In particular, do small OSS projects require large groups of code contributors and bug reporters?
3) What testing techniques are used in the OSS projects that exhibit higher reliability than equivalent commercial software? Answering this question requires developing an assessment mechanism that can objectively compare the reliability between OSS and commercial software, and an examination of the testing process, tools, and criteria used in these projects.

Results from efforts to answer these and other related questions can surely lead to improvements in development meth-

ods. The advantage of studying OSS projects is that process and product data is readily available.

ACKNOWLEDGEMENTS

We thank many FreeBSD project participants for help on this research. FreeBSD Core Team members W. Peters, J. Baldwin, M. Losh, and M. Murray provided detailed project information. Core Team member R. Watson and FreeBSD developer J. Gibbs helped us to mirror the FreeBSD CVS repository. GNATS Administrator C. Davies provided information about PR classes and the bug report process.

We also thank Audris Mockus for his encouragement and for his very helpful comments and suggestions for revising and extending an earlier version of this paper. Suggestions by the reviewers helped to improve the presentation of our results.

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] T. T. Dinh-Trong and J. Bieman. Open source software development: A case study of FreeBSD. *Proc. Tenth Int. Software Metrics Symposium (Metrics 2004)*, pages 96–105, 2004.

[3] 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.

[4] 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.

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

[6] B. Fitzgerald and T. Kenny. Developing an information systems infrastructure with open source software. *IEEE Software*, 21(1):50–55, Jan-Feb 2004.

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

[8] E. Gamma, R Helm, Johnson R., and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading MA, 1995.

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

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

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

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

[13] 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.

[14] S. Lussier. New tricks: How open souce changed the way my team works. *IEEE Software*, 21(1):68–72, Jan-Feb 2004.

[15] D. G. Messerschmitt. Back to the user. *IEEE Software*, 21(1):89–90, Jan-Feb 2004.

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

[17] 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.

[18] Netcraft. Nearly 2 million active sites running FreeBSD. *Netcraft*, July 2003. Posted July 12, 2003 to http://news.netcraft.com/archives/2003/07/12/nearly_2_million_active_sites_running_freeb

[19] J.S. Norris. Mission-critical development with open sourse software: Lessons learned. *IEEE Software*, 21(1):42–49, Jan-Feb 2004.

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

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

[22] GNATS GNU Project. Gnats (version 4.0). `http://www.gnu.org/software/gnats/`.

[23] The Free BSD Project. FreeBSD (version 5.0), [computer software]. `http://www.freebsd.org/`, 2003.

[24] E.S. Raymond. Up from alchemy. *IEEE Software*, 21(1):88–90, Jan-Feb 2004.

[25] 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.

[26] N. Serrano, S. Calzada, J. M. Sarriegui, and I. Ciordia. From proprietary to open source tools in information systems development. *IEEE Software*, 21(1):56–58, Jan-Feb 2004.

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

[28] 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.

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