Perspectives on Open Source Software

Scott Hissam
Charles B. Weinstock
Daniel Plakosh
Jayatirtha Asundi

November 2001

TECHNICAL REPORT
CMU/SEI-2001-TR-019
ESC-TR-2001-019

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Perspectives on Open Source Software

CMU/SEI-2001-TR-019
ESC-TR-2001-019

Scott A. Hissam
Charles B. Weinstock
Daniel Plakosh
Jayatirtha Asundi

November 2001

Internal Research and Development

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Norton L. Compton

Norton L. Compton, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Abstract

Open source software (OSS) is emerging as the software community's next "silver bullet" and appears to be playing a significant role in the acquisition and development plans of the Department of Defense (DoD) and industry. Yet, as with all previous silver bullets, there are problems with blindly embracing the OSS paradigm.

To become familiar with the benefits and pitfalls of using OSS, the Software Engineering Institute (SEI) undertook an internally funded study looking at it from various perspectives:

- the user of OSS
- the developer of OSS
- the organizations looking to deploy software systems comprised (partially or completely) of OSS components

During the period of this study, members of the SEI technical staff hosted meetings, conducted interviews, participated in open source development activities, workshops, and conferences, and studied available literature on the subject. Through these activities, the authors have been able to support and sometimes refute common perceptions about OSS. This report is the result of their study.

1 Introduction

As the modern software industry has evolved, there have been a myriad of innovations, some successful and some not so successful. Early successes such as structured analysis and design have been wonderful stepping-stones to more modern, spiral-development approaches. In the past, failures appeared as exciting prospects for software and software engineering as a whole but only ended up as a short-lived “blip” on the radar screen. The Ada language and CASE (computer-aided software engineering) tools are two that immediately come to mind. Brooks, acknowledging Ada as something less than a revolutionary advance, states that the philosophy behind Ada (e.g., modularization, abstract data types, and hierarchical structuring) is more an advance than the language itself [Brooks 87]. Be it an early success that laid the groundwork for future innovation, or a spent “silver bullet” in the chamber of software engineering, each contributed to a greater understanding of the software community.

Now there is a would-be silver bullet with the words *open source software* (OSS) emblazoned on the side of it. This bullet may indeed be the next innovation or great leap forward in the way the software community develops software; however, it may also be a dud—and without a crystal ball it is hard to tell at this time. So until it becomes possible to see into the future, we (i.e., the software community) can only analyze the current situation to gain an understanding of what OSS is, how it is developed, and how it is contributing to the way we develop software, and to learn where we can apply OSS in the overall science of software engineering.

1.1 OSS at the Software Engineering Institute

In 1999, the Software Engineering Institute (SEI) initiated a study to look at OSS. The purpose of the study was to get a broad understanding of not only OSS itself, but moreover the development methods involved (which included those from the developer community) as well as the users of OSS. The study was geared towards an eye on a practical perspective of what open source really is, with the goal of differentiating between hype and reality.

The SEI partnered with Carnegie Mellon University’s Software Industry Center (SWIC) to aid in the study. The SWIC is the 16th industry center established by the Alfred P. Sloan Foundation to identify and sift through industry trends, separating the ephemeral from the lasting focus on innovation and competition, software-development practice, talent, workforce, and human capital.

To that end and over the course of the study, the members of the SEI technical staff and the SWIC hosted meetings, conducted interviews, participated in open source development activities, workshops, and conferences, and studied available literature on the subject. Through these activities we have been able to support and sometimes refute common perceptions about OSS. This report is the result of our study.

1.2 Organization of This Report

This report is organized as follows:

- In Section 2, we introduce OSS in its many forms and talk briefly about its history.
- In Section 3, we discuss the various case studies conducted by the SEI and SWIC and highlight important findings from those studies.
- In Section 4, we expand on those case studies, glean observations on the state of practice of OSS, and make recommendations that will be helpful to those wanting to get involved in the OSS community.
- In Section 5, we summarize this report.

2 What Is Open Source Software (OSS)

The term *open source software* at the most basic level simply means software for which the source code is open and available. Open and available is meant to convey two concepts:

- *open*—The source code for the software can be read (seen) and written (modified). Further, this term is meant to promote the creation and distribution of derivative works of the software.
- *available*—The source code can be acquired either free of charge or for a nominal fee (e.g., media and shipping charges or online connection charges).

Today, making source code *available* can be as simple as posting the code on the World Wide Web (WWW) or posting it in an online newsgroup. Making the software *open* is also simple—place no restrictions on how the software is actually used or by whom.

2.1 OSS—A Formal Definition

Others have gone to great lengths to define OSS. In fact an entire group, the Open Source Initiative (OSI), formed and established the Open Source Definition (OSD). The OSD is a formalization of what it means to distribute software that is open source, namely

1. Free Distribution (i.e., *license cannot restrict selling or giving away*)
2. Source Code (included) (i.e., *software includes unobfuscated source code*)
3. Derived Works (i.e., *software can be modified and distributed by others*)
4. Integrity of the Author’s Source Code (i.e., *know who gets credit for the source code*)
5. No Discrimination Against Persons or Groups (e.g., *ethnic groups, religious groups*)
6. No Discrimination Against Fields of Endeavor (e.g., *genetic research*)
7. Distribution of License (i.e., *forbidding the addition of further restrictive licensing)*
8. License Must Not Be Specific to a Product (i.e., *rights cannot depend on a particular distribution*)
9. License Must Not Contaminate Other Software (e.g., *both licensed software and OSS can coexist in the same distribution*) [OSI 01a]

The complete text and rationale of the OSD are included in Appendix B.

Under OSI (strictly speaking) a software product is in fact open source if and only if it conforms to the OSD [OSI 01b]. Upon reviewing the complete text of the OSD, it is interesting to point out that the definition does not pertain specifically to the source code itself, but rather to the license under which the source code is distributed. Therefore, in strict conformance to the OSD written by the OSI, a software product that conforms to only eight of the nine criteria is not OSS. As a means to differentiate between OSS and non-OSS, the OSI has established a legal certification mark called the OSI certification mark. Under the OSI Certification Mark Program, software that is distributed under an OSI-approved license can be labelled as *OSI Certified*. The complete notice is

> *"This software is OSI Certified Open Source Software.*
> *OSI Certified is a certification mark of the Open Source Initiative"* [OSI 01b].

As of October 2001, there are 23 OSI-approved licenses.[1] A software distribution can adopt one of these licenses, unchanged, and be OSI certified immediately. Alternatively, a company can apply to have its license included on the OSI-approved list of licenses by submitting the license for review to the OSI (at which point it has to be accepted).

It is vital to point out that the OSI certification mark does not talk to, espouse, or certify anything about the actual software itself. That includes anything about the quality of the software (robustness, security, integrity, maintainability, usability, etc.). Nor does it include anything about the methods, process, or techniques used to develop the software (testing, analysis or design, review, configuration management, documentation, etc.). *It would be a mistake to assume that any OSS distribution, touting an OSI certification mark, is either of high quality and developed under rigorous methods or of low quality and developed under ad hoc methods.*

In Section 2.4, we discuss more about the motivation behind using and the perceptions of OSS.

2.2 What's Not OSS

In stark contrast to OSS is commercial off-the-shelf software (COTS) or better, closed-source software (CSS). In this report we prefer to refer to software in which the source code is not open and available as CSS. COTS is typically a binary distribution of software which, through its licensing agreement (one that you agreed to by purchasing and installing the software), legally bars the purchaser of the software from disassembling and/or reverse engineering the software, and furthermore from taking any such (illegally) derived works and modifying them for any purpose whatsoever. In that vein COTS is typically CSS. Other forms of CSS include shareware and royalty-free libraries (e.g., the runtime library for a compiler). These are virtually never distributed with source code, often have restrictive licenses, and usually require the payment of a fee.

[1] This is according to the Open Source Initiative's Web site located at <http://www.opensource.org/licenses/index.html>.

It would be wrong not to recognize that there are real COTS products that do actually come with source code both in read-only and read-write form. But such products are still considered CSS because they do not promote (and their licenses explicitly forbid) the creation and distribution of any derived works.

CSS often violates many tenets of the OSD (discussed above). For example once you acquire a CSS product, you are often forbidden to redistribute that software in nearly any form (regardless of whether you actually paid for the product). Often software licenses forbid the use of the software in safety-critical and even in military environments (criteria #6 of the OSD). An interesting case in point is the license for the Java[™] Development Kit (JDK) version 1.3 from Sun Microsystems, which explicitly states (in paragraph 2 of the `LICENSE` file) that the development kit

> *"… is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility."*

Such a statement discriminates emphatically against a field of endeavor regardless of the rationale that leads Sun Microsystems to include such a clause in its license agreement.

In Section 4.4, we take a closer look at OSS and its ties to CSS.

2.3 The History of OSS

Software source code that is open and available has been around since the earliest days of modern computing [Arief et al. 01]. Feller and Fitzgerald provide a good historical account of open and available source code from the 1940s to the contemporary advocacy campaign of the OSI [Feller et al. 02]. In their book, Feller and Fitzgerald acknowledge some of the earliest code-sharing activities between scientists working on some of the earliest computers, such as the Electronic Numerical Integrator and Calculator (ENIAC); formalized groups that share software, such as the Project for Advancement of Coding Techniques (PACT); and published articles including source code, such as "Algorithms" in the *Communications of the ACM* [Feller et al. 02, Leonard 00].

[™] Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Many trace the beginnings of the modern open source movement back to the University of California at Berkeley's software distribution of Unix (BSD Unix) and later to the formation of the GNU ("GNU's Not Unix") Project and the establishment of the Free Software Foundation (FSF) by Richard Stallman. The purpose of the GNU Project was to create a free version of Unix and Unix tools not impaired by the restrictions of licensing and distribution. Stallman explains that the term *free software* meant *free* as in freedom and not price (as in *free beer*). Further, Stallman clarifies that a program is *free* software if

> *"You have the freedom to run the program, for any purpose.*
>
> *You have the freedom to modify the program to suit your needs. (To make this freedom effective in practice, you must have access to the source code, since making changes in a program without having the source code is exceedingly difficult.)*
>
> *You have the freedom to redistribute copies, either gratis or for a fee.*
>
> *You have the freedom to distribute modified versions of the program, so that the community can benefit from your improvements"* [Dibona et al. 99].

Although GNU did not actually produce a free Unix kernel under the FSF, it did produce (adapting from Unix implementations) a host of free utilities that essentially make a Unix system what it is, everything from Internet utilities, servers, and clients, to compilers and editors. Today the FSF boasts the collaboration of the Linux kernel and the GNU tools suite as having met the goal of achieving a free version of Unix [Stallman 98].

Perhaps most importantly, the GNU Project produced the General Public License agreement that is commonly referred to as the GPL or Copyleft License. That license, perhaps more than anything else, was the instrument that facilitated the development of all the tools that are now commonly used in Linux as well as many commercial variants of Unix and further non-Unix operating systems. The GPL permitted the redistribution of source code for unfettered use and access so long as those reusing (and redistributing) the source code continue to do the same and so long as any changes are also available in source form and subject to the same license.

The ideas of Richard Stallman and their formalization embedded in the GPL are a keystone in the movement to keep software open and available to facilitate innovation and the advancement of computer science.

Before the term *open source* was used as readily as it is today, software that was open and available existed. Constructing an exhaustive list of software that could be categorized as open and available would be difficult. Some of the predominant names that would appear in that list (with apologies to those not listed) are shown in Table 1.

Table 1: Open and Available Software Predating OSS

Operating Systems Linux FreeBSD NetBSD OpenBSD	GUI X-Windows	Languages Perl tcl/tk Python TeX	Compilers GCC
Internet FTP Sendmail BIND	Editors Emacs Vi	Web Server/Browser Apache Mosaic Lynx	

The formalization and definition of what OSS is would come later.

2.3.1 Birth of OSS

As presented earlier, software for which the source code is available predates the contemporary campaign of OSS. A number of events are attributed to the eventual birth of the modern OSS movement. The highlights of those events include

- Shortly after AT&T developed and released the first Unix, the university and user community began to create software (Unix tools and utilities) that ultimately became "standard" in almost every Unix distribution.
- In 1984, Richard Stallman founded the GNU Project and shortly thereafter founded the FSF (discussed above).
- In 1991, Linus Torvalds released Linux version 0.01 as an existence proof of a rudimentary Unix kernel free from any intellectual property. Linus was motivated to demonstrate that a single monolithic kernel was just as portable as a microkernel architecture [Dibona et al. 99].
- In 1992, USL (Unix Systems Lab, a subsidiary of AT&T) sued BSDI (a commercial start-up) and the Regents of the University of California, claiming that portions of the Net/2 release of BSD-Unix contained AT&T-copyrighted code and intellectual property [Salus 94].
- In 1993, FreeBSD, a patchkit for the 386BSD from Bill Jolitz, was released, but this version was still based on the Net/2 BSD code [Hubbard 95].
- In 1994, Linus Torvalds officially released Linux version 1.0.
- In 1994, FreeBSD version 2.0 was released free from any BSD code [Hubbard 95].

- In 1995, Apache HTTP Web Server version 1.0 (based on the NCSA httpd version 1.3 Web Server) was released [Apache 99].
- In 1996 and 1997, Microsoft Corporation's Internet Explorer version 3.0 (1996) was released, followed by version 4.0 (1997). Meanwhile Netscape® Communicator (Netscape Communications Corporation's Web browser) continued to lose market share in the Internet Web browser market [Barksdale 99, Schnoll 01].
- In 1997, Eric Raymond wrote *Cathedral and the Bazaar*, documenting his successful open source project, `fetchmail`, and explaining the open source movement and the motivations for getting involved in it [Raymond 99].

Through this brief history, an interesting pattern is visible—first with Unix (a product of AT&T) and then later (see below) with the Netscape Communicator—the growth and development of an open, available, and free (as in *free beer*) software product was instigated by the *wranglings* of a large commercial company. In the USL v. BSDI suit, AT&T claimed that its intellectual property was contained in the Net/2 BSD[2] release of Unix (developed by the Regents of the University of California). This lawsuit was ultimately settled when it was revealed that the AT&T distribution of Unix incorporated a serious amount of code developed at Berkeley. Later Microsoft Corporation dominated the personal computer software market while exercising its ability to offer a free Internet Web browser at the expense of another company (Netscape Communications Corporation in this case).

The precipitating event in February 1998 that heralded the birth of OSS was the unprecedented release of the commercial source code for Netscape Communicator version 5.0, for free licensing on the Internet (i.e., Project Mozilla). In that same move, Netscape also made the current Netscape Communicator version 4.0 standard edition—in its binary form, not OSS—free to all users (for which it had been charging roughly $30) [Netscape 98]. Netscape's approach and subsequent decision to release the source code for its next-generation browser to the Internet was inspired by the 1997 writings of Eric Raymond. This was an attempt by Netscape to engage a community of developers (those same developers who contributed to the success of Linux, Apache, and a host of other open and available software products) to build a better Internet Web browser—and to compete against the market infiltration of Microsoft Corporation's IE (which threatened the existence of Netscape).

Soon after Netscape Communication Corporation's announcement, Tim O'Reilly and other leaders from other free software projects participated in the first Free Software Summit in March of 1998 (now called the Open Source Summit) and adopted the term *open source* [Raymond 99]. Key in their strategy was to coordinate a complete media blitz and tout the successes of the (now) open source poster children, including Linux and Apache, with Netscape's announcement as the most recent battle cry to encourage other corporations to take the same road. Further, the Open Source Web site opened up for business, complete with the first version of the OSD (discussed in its present form in Section 2.1).

® Netscape is a registered trademark of Netscape Communications Corporation.

[2] For the definition of this and other acronyms in this document, see Appendix C.

The open source movement and the OSD were now alive and well.

2.4 Perceptions of OSS

How OSS came to be labelled as a silver bullet is a matter of opinion and speculation. Much of any opinion (including ours) would be based on the attention that OSS has received in the mainstream press. Growth in the software industry, fueled by the explosion of e-commerce and dot-com companies—all in response to the overwhelming acceptance and interest of the WWW—through the 1990s, warranted such coverage. The list of companies that grew from literally nothing to companies whose stock was worth millions of dollars on Wall Street gave credence to what then appeared as a phenomenon whose potential was limited only by those investors who were willing to dump money into dot-com companies. The premium example of this explosion came with the initial public stock offering for Red Hat Software.

Red Hat Software, founded by Robert Young and Marc Ewing in January 1995, was (and still is) the leading seller of Linux-based operating systems. Their basic business model was to give the Linux software away for free (as permitted by the OSD) but to sell support services and provide superior packaging and distribution, making it easy for users to acquire, install, and use something as complex as a Unix-based workstation/desktop environment. Red Hat Software's initial public offering occurred in August 1999. Its stock opened at $14 per share and closed the day at $52 1/16, stunning the technology markets[3]. Other companies emerged to make a business from OSS. Following on the success of the Red Hat Software IPO, VA Linux, which sold products and services around the open source Linux operating system but included hardware as part of its product suites, had an initial public offering in December 1999 that soared close to 700% in one day, settling close to $242 per share.[4]

These tremendous starts, which garnered national and worldwide attention, brought OSS and the companies that intended on making a business from OSS into the mainstream.

[3] Shares of Red Hat Software stock peaked just above a closing of $136 per share and through the softening technology markets of 2000 and 2001 fell to under $4 per share.

[4] As of this writing, VA Linux no longer sells hardware and its stock is hovering just above $1 per share.

2.4.1 OSS Is the Best Thing Since…

It is not surprising, given the attention that OSS has received, that myths about OSS—positive and negative—have come out.

The first myths we'll discuss are about the software itself. The first three items in the list below stem from the idea that OSS, being under constant peer review by developers around the world and around the clock, must therefore be of higher quality, that is, it must be

- (more) reliable
- (more) robust
- (more) secure (or no security through obscurity)

Raymond makes two basic arguments that support this notion [Raymond 99]. The first is that *hackers* (OSS developers), knowing in advance that others will see the code that they write, will be more likely to write the best code that they can possibly write—out of fear of embarrassment from community shame for writing anything less.

The second argument is asserted as *Linus's Law*: "Given enough eyeballs, all bugs are shallow" [Raymond 99]. Again the notion is that because there are thousands of developers reviewing OSS code, 24x7x365, a flaw in the code will be obvious (hence shallow) to someone (who will either report or fix it). Based on those two premises, the common community belief is that OSS is of higher quality, in the sense that it applies to all of the '*ilities*.[5]

In fact there is open and available software, which is good software and, by many measures, high-quality software. Much of the software listed specifically in Table 1 on page 7 would fit into that category. The question posed in this report and addressed in the case studies in Section 3 is whether all OSS should share the same status as high-quality software.

The next myth about OSS that predominates the community is the sense of control you have when you have the source code for an OSS product. Such control means the ability to read and modify the source code for your own purposes. Such an edge is viewed as the main advantage over CSS. In CSS, if the vendor for a CSS product goes out of business, what would happen to the software already purchased (with respect to support, future versions, and bug fixes)? The response to such a scenario in the OSS realm is that there is no vendor to go out of business. And even if there were such a vendor, you have the source code. What more could you want? In the OSS realm, you have omnipotent control. In Section 4.4, we explore this myth further.

[5] The term 'ilities is often used to refer to various properties of systems and components in general, such as scalability, reliability, security, and adaptability.

Closely related to this notion is the myth that OSS (without a traditional vendor as in the CSS realm) has poor documentation and little support. The idea of poor documentation comes from the assumption that hackers are off coding wildly and have neither the time nor the motivation to document what they produce—either the code-level documentation or the end-user documentation. The concern that there is little support for OSS comes from the sense that there is no one to phone when there is a problem (since there is no traditional vendor in the contact). O’Reilly discusses this myth briefly [O’Reilly 99]. The premise of O’Reilly’s response to such a myth is based on software that has been around for 10 or 20 years. One positive note is that there is a trend that any quality gaps in support and/or documentation are being filled by support companies like Red Hat and VA Linux. But again, should this always apply to all OSS?

Other myths about OSS are targeted towards the hackers themselves rather than the software: for instance, the myth that there is a world’s worth of hackers sitting around waiting and eager to work on an OSS project free of charge (*free* as in no cost). Such an image gives the false impression that anyone can get an army of programmers to work on a problem free of charge and forego the traditional development costs associated with traditional software-development activities. Unfortunately, there is a disturbing trend that some companies are trying to benefit from the generosity and culture of the OSS community, which we report in Section 3. But also, on a positive note, there are trends that are working towards linking up eager hackers, for cost, with companies that want to obtain freelance programming talent (discussed in Section 4.1). The old adage *“You can lead a horse to water, but you can’t make him drink”* best describes the OSS community—that is, *“You can put the code out in the community, but you can’t make a hacker code.”* The likelihood that an OSS product will be successful (or that the hackers will help you) is based on the characteristics discussed in Section 5.1 (which are not guaranteed).

The last myth that we cover in this report is that those hackers out in the OSS community are a group of mavericks working in an unorganized, haphazard, ad hoc fashion. Given the global reach of the Internet and therefore distributed nature of hacker-based development, this might be a foregone conclusion. For some, this is the allure of the OSS development process—that is that there is no “process-monger” or program manager hanging over the progress of the development effort (hence the process is unpredictable and progress is immeasurable). As shown in the Apache[6] case study in Section 3.2, not only is the OSS development highly organized and controlled, but many characteristics of the design and architecting process mimic that which is typically seen in commercial organizations.

[6] The Apache HTTP Web server is considered to be one of the top three successful OSS products to date.

3 Case Studies

One of the ways in which we attempted to understand the OSS phenomenon was to actually get involved in or research several efforts/events. There were five such studies:

- AllCommerce—an e-commerce storefront solution
- Apache—an open source Web server
- Enhydra™—a Java-based application server
- NAIS—a NASA-operated Web site that switched from Oracle to MySQL
- Teardrop—a successful Internet attack affecting OSS and CSS

The purpose in selecting these specific OSS projects was to take varying perspectives of OSS, in terms of software development, the products themselves, and users.

The AllCommerce case study focused on software development in the OSS paradigm. A member of the SEI technical staff got involved in the process of hacking the product to discover bugs and add new features to the product. The express purpose of this case study was to obtain firsthand experience in working on an OSS product from the *inside*, that is, to learn the process by which changes are actually proposed, tracked, selected/voted on, and accepted.

The Apache case study takes an academic, research perspective (actually the result of a doctoral thesis) of the OSS-development process. This case study looked at the individual contributions made to the Apache Web server over the past five years and whether that contributor was from core or non-core Apache developers.

From a purely product-centric perspective, the Enhydra case study focused on the qualitative aspects of an OSS product and looked at coding problems found in the product by conducting a critical code review.

The NAIS case study, which focused on the end user, looked at a real application developer who switched from a commercially acquired software product to an OSS product. Specifically, this case study examined how and why that particular OSS product was selected, the degree to which the application developer was engaged with the OSS development community, and the level of satisfaction that the NAIS had with the selected OSS product.

™ Enhydra is a trademark of Lutris Technologies, Inc.

Finally, the Teardrop case study looked into one of the predominant axioms of OSS: that OSS is more secure than software developed under more traditional means. This case study takes apart one of the most successful distributed denial-of-service (DDoS) attacks, and looks at the role that OSS played in the propagation of that attack on CSS and the response by the OSS community.

Each of these case studies is discussed in the remainder of this section.

3.1 AllCommerce Case Study

AllCommerce is an open source package that implements a storefront on the Web. The complete package, written in Perl and dependent upon MySQL (or some other relational database system), is released under the GPL and is free to anyone to use and/or modify. The complete package promises to provide a storefront complete with purchasing, inventory, shipping, billing, and sales capabilities.

AllCommerce was developed originally with support from a company called OpenSales, which later became Zelerate. This case study is based upon our observations of this product and its development over an approximately 10-month period. As we had no visibility into the internal operations of OpenSales/Zelerate, some of what follows is based upon conjecture, which we have been unable to have officially confirmed or denied.

3.1.1 Objective of the Study

With AllCommerce we wanted to look at OSS from the inside. The objective of this was to see what it took to be accepted by the community, to learn how changes really get handled for those who aren’t core developers, and to look for the benefits and deficiencies of the OSS model.

Zelerate provided support in the form of employees and resources for developing AllCommerce. Apparently Zelerate employees did the initial design. The Web site that hosted AllCommerce was provided by Zelerate, which also provided ongoing development in the form of employee time. Zelerate apparently intended to make money through consulting services related to AllCommerce, including customizations, installation, maintenance, and so forth. This appears to be a common way in which companies attempt to profit from OSS. It is also increasingly apparent that this is a difficult business model to execute properly. Zelerate folded in the middle of this study but, due to the open source nature of its product, AllCommerce lives on.

It is not entirely clear what motivated Zelerate to make AllCommerce open source in the first place. Possible reasons include

- to obtain community help in developing a very complex product
- to create an installed base, which may lead to customers for its commercial services
- to capitalize on the hype around OSS

3.1.2 Conducting the Study

We became aware of AllCommerce shortly before beginning our effort. We began by obtaining a copy of the software and attempting to install it on a local Solaris server. Our intention was to evaluate the software and see where we might be in a position to contribute. Installation went fairly smoothly, but in our initial tests of the product, we discovered a bug in how credit cards were dealt with—something that we fixed and submitted back to the maintainers. Our fix was included in the next release of the product.

We then turned to evaluating the features of the product. It soon became apparent that this was a *work in progress,* not ready for prime time—although there were many sites purporting to use it as an e-commerce solution, and it even garnered an extremely positive review from ZDNet.[7] In fact, we never were able to fully install this system in a useable way on our e-commerce Web site. While we had no products to sell, we found that setting up our dummy store took much too much effort and required us to adapt to a business model that was not relevant to our products (SEI-logo merchandise).

After considering many of the things that needed to be done to make it useable for our store, we decided to focus on shipping models. As installed on our systems, AllCommerce had the concept of warehouses, but had implemented only the ability to specify a single warehouse. Its shipping model was to charge the actual cost based upon the weight of the object being shipped and the distance it had to travel. To accomplish this, tables were built from information supplied by shipping companies (e.g., USPS, UPS, Federal Express). Shipping to other countries was not implemented.

Because this model does not work well for all types of businesses, we sat down and specified what we believed was a complete shipping model for AllCommerce. This included the ability to drop-ship from multiple warehouses, charge several different ways (e.g., by the piece, a flat rate, or weight and distance), and ship to foreign countries. The resulting one-to-two-page document was submitted to Zelerate and quickly placed on the Web site as a project. A mailing list for the project was created and we were added to it. Unfortunately no one else seemed interested in pursuing this aspect of AllCommerce. A short while later Zelerate folded and

[7] This is according to the article located on ZDNet’s Web site at <www.zdnet.com/products/stories /reviews/0,4161,2629084,00.html>.

AllCommerce development was transferred to SourceForge.net’s Web site. Along the way, the shipping model document disappeared.

Although AllCommerce lives on at SourceForge.net and the mailing lists associated with it are still somewhat active, it appears that little or no development is taking place. The messages on the mailing lists mostly have to do with installation and configuration problems by people attempting to use AllCommerce. The last stable release as shown on SourceForge.net is from March 2001. The listed alpha release predates that and has the same version number. Attempts to get a status from the listed project administrators have been met with silence. The conclusion is that AllCommerce is an open source solution that has failed.

3.1.3 Summary

While it was relatively easy to get involved in this effort while there was active development, it seems apparent that there was no real developer community (as opposed to a user community) outside of Zelerate. Once Zelerate was unable to provide support for development, what activity there was stopped. This fits with a conjecture of ours that open source efforts succeed when the developers are also the users of the product and/or when there are substantial resources being allocated to the project. With the loss of Zelerate’s support, it seems that neither of these conditions is true any longer.

3.2 Apache Case Study

A popular public domain Web server was developed at the National Center for Supercomputing Applications (NCSA), located at the University of Illinois, Urbana-Champaign, during the early 1990s. When most of the developers of the NCSA Web server left to join Netscape, the support group for the NCSA Web server was depleted. This resulted in the creation of the Apache (read *a-patchy*) Web server project by a number of independent noncommercial developers who were interested in maintaining and enhancing the NCSA Web server. An early task of the founding group was to test and integrate the various patches written by Webmasters everywhere. It has been six years since the conception of the Apache project, and because so many architectural changes have been made to the original Web server, it hardly resembles the old NCSA Web server.

Apache is managed jointly by a group of volunteers who are known as the Apache Group. This project is now a project of the Apache Software Foundation, which provides support to a number of open source software projects. The foundation accepts contributions in the form of development support as well as money.[8]

[8] See the Apache Software Foundation at <http://www.apache.org>.

3.2.1 Objective of the Study

The research was carried out to examine the question, Are open source projects any different from commercial projects? The issue is analyzed from the perspective of the development process, the contributors, and the type and distribution of their contributions. Reports in the popular press portray OSS projects as consisting of a maverick group of hackers, organized haphazardly and adding code to the project in an ad hoc manner ("babbling bazaar of differing agendas and approaches" [Raymond 99]). Other notions include the evolution of architectural designs through public interaction and a large number of people looking through the code base ("Given enough eyeballs, all bugs are shallow" [Raymond 99]). We structure our argument around addressing all of these popular notions, draw comparisons between commercial software and OSS development, and show that they are in fact not true.

3.2.2 Conducting the Study

Important artifacts of any OSS project are the public email distribution lists. The distribution lists are for purposes of announcement and discussion. The discussion of improvements, bug fixes, and the future direction of the software are carried out on the list called *new-httpd*. The *apache-cvs* list sends email to all subscribers whenever source code is committed to the concurrent version system (CVS) source tree. The *apache-bugdb* distribution list announces to subscribers the various bug reports submitted by users. This list is a good indicator of all the available beta testers for Apache and provides a forum for bug reporting and problem dissemination. All emails have been archived and provide a wealth of information on the evolution of the software. We have used the information contained in these archives for our analysis. There are also other distribution lists such as: *apache-docs*, *announce*, *current-testers*, *gui-dev*, *mirrors*, *modproxy-dev,* and *stable-testers*. These lists are discussions about topics related to Apache or discussions about related modules that are not explicitly part of the core server. For this reason, we do not use these archives for our analysis.

The popular notion assumes that there is no organization or clearly defined hierarchy in OSS projects. On the contrary, the structure of the Apache server project as described by Fielding [Fielding 99] is quite similar to that of successful commercial organizations [Cusumano et al. 95]. Our observed equivalence of development roles between Apache and commercial organizations is shown in Table 2, but with a difference: the roles that Apache developers assume are volunteered,[9] not appointed, and all participants in the Apache server project also contribute as testers of the product.

[9] In contrast to commercial development, Apache developers volunteer or appoint themselves into one or more various roles, which are ultimately accepted by the community for a project.

Table 2: Equivalence of Roles in Software Projects

Commercial Projects	**Apache Server Project**
Project manager	Release manager
System architects	Senior members of the core group
Project members	Core group
Testers	All developers of the project

A central CVS repository is used to manage the source code and an active core-group member carries out the day-to-day maintenance. Mockus and associates outline the peer review process of the Apache project by which problems are discovered, assigned, fixed, and incorporated [Mockus et al. 00]. The process describes how any change must be voted in by at least three core-group members so that it can be committed to a source tree. It is assumed that every new change is tested for compilation problems and works according to the specification. The approach to development is incremental and very similar to the *synchronize and stabilize* concept (described by Cusumano and Selby [Cusumano et al. 95]) that was adopted at Microsoft Corporation. This incremental approach is similar to those adopted at many other commercial organizations like Hewlett-Packard, EDS, TRW, and Motorola.

It is a common perception that the larger the number of people working on an OSS project, the more likely the detection of bugs in the source code will be. OSS is therefore perceived to be relatively more bug free than commercial software that would not see so many testers before commercial release. A detailed analysis suggests a different interpretation. We believe that there are three types of errors in any software: coding errors, logical errors, and architectural flaws. Coding and logical errors are usually business-context independent and are easy to spot and eliminate. An analysis of the number of people reporting bugs and those contributing patches shows that against a total of 5,116 testers reporting bugs, there are only 200 contributing patches. Testers typically report problems they observe while running the software, but rarely look into the source code to identify the source of the problem. On the other hand, patch writers may scrutinize source code and make changes to fix errors. Hence, the free availability of source code does not automatically imply the scrutiny of the source code by all testers, but rather only by a subset of them. The number of patch submitters could be considered a proxy for the number of source-code scrutinizers. The ratio of source-code scrutinizers (~200) to the number of active project participants (~50 core-group members) is still much higher (~4:1) than that observed in commercial organizations (at the maximum 1:1). Thus we see that Apache differs from commercial development in the area of testing, as it has a larger number of users willing to report bugs in a structured manner. The ability to spot and fix bugs does not automatically lead to a more secure software product. McGraw argues that because a software product is a continually evolving and dynamic system, security bugs are difficult to eliminate because they are generated continuously [McGraw 00]. Schneider, on the other hand, suggests that since security holes are due to design and architectural flaws, the availability of source code alone is inadequate to eliminate these security holes [Schneider 00]. Architectural flaws are business-context dependent and require extensive knowledge of the software system. A more detailed understanding of the architecture and design would be required for an individual to be able to find and fix these flaws. As discussed later, this competence is not widely available among OSS developers.

Commercial software-development organizations face a major challenge in the distributed development of software. Herbsleb and Grinter studied the development of a commercial product at geographically distributed locations [Herbsleb et al. 99]. Their results suggested that distributed development leads to time delays and frequent miscommunication amongst developers due to cultural and time differences. Mockus and associates showed that 85% of the modification requests (MRs) came from the top 15 developers whom they considered to be members of the core group. We performed this analysis counting the number of contributions from core-group members. Our analysis (shown in Figure 1) indicates that 51 core-group members accounted for ~93% of the MRs. Analysis by various file types (shown in Figure 2) shows that ~94% of the code and header files and ~98% of the engineering environment (ENGENV) files (like build files and configuration files) were contributed by the core group. Even though 500 individuals contributed to the project, the core group dominates in MRs, thus guiding the direction of development.

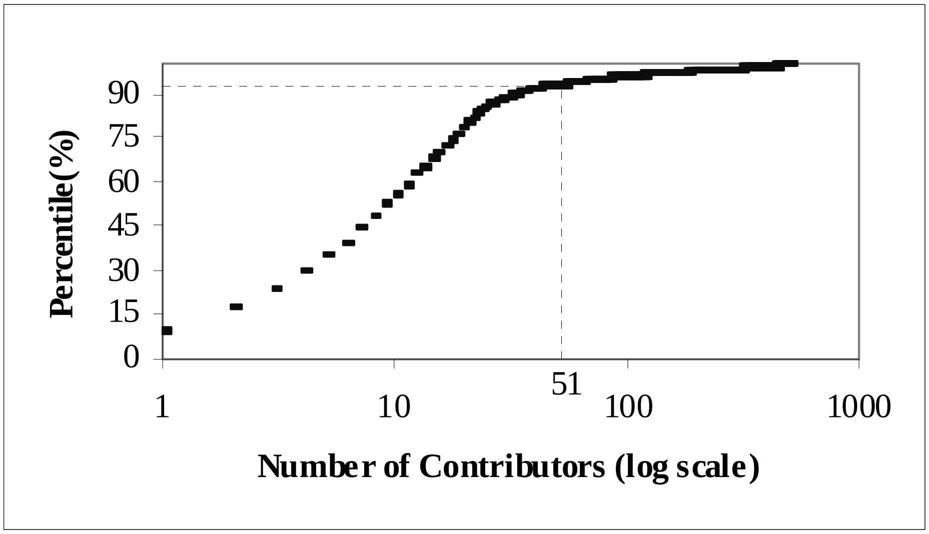

Figure 1: Cumulative Distribution of Modification Requests

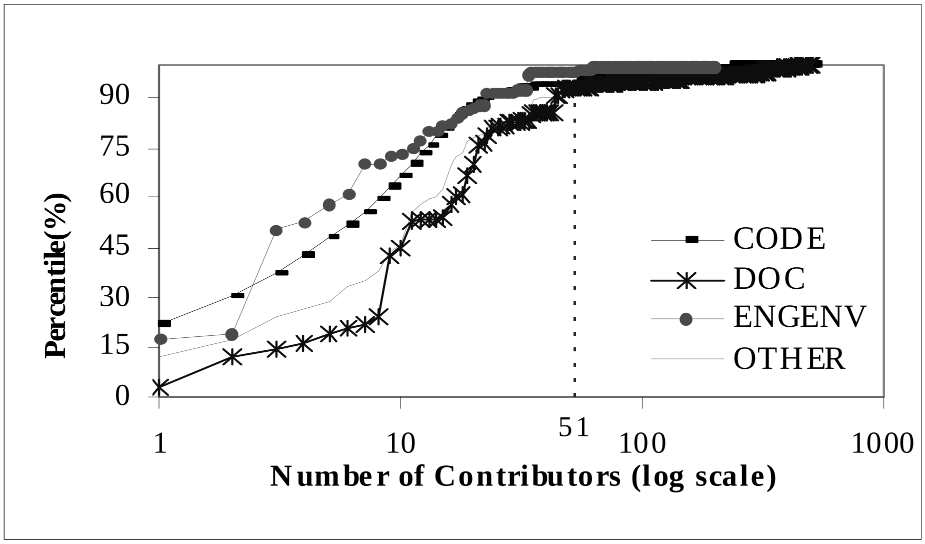

Figure 2: Cumulative Distribution of File Types

Software architecture forms the heart of any software system. Traditionally very few, mostly collocated individuals work on architecture design, and this effort is not shared widely. We looked for anecdotal evidence of major architectural changes to the system and tried to map the source of these changes to developers. To preclude the possibility that while selecting anecdotal evidence we could have selectively sampled only those design changes that are made by a few people, we randomly chose a sample set of changes to the system and divided them into architectural and nonarchitectural changes on the basis of expert voting. To differentiate the contributors to architectural and nonarchitectural changes, we compared the number of contributors for the change as well as the number of similar changes documented in the *apache-cvs* distribution list. In the Apache project, developers readily acknowledged help obtained from other developers, and this helped us to identify the people involved in effecting the change.

Anecdotal evidence on the following products shows that a few individuals perform architecture design independently:

- Shambhala or Apache version 0.8
- Netscape Portable Runtime (NSPR)
- Multi-Process/Multi-Threading Model (MPM)

The community is notified and invited to contribute once the broad structure is laid out. The implementation details are then fine-tuned, and the server is tested extensively. This suggests that, for the Apache server project, a coherent architecture is maintained by a small select group of individuals and the development is not chaotic. Similarly, in commercial development projects, we observe only a small number of collocated individuals working on the architecture design.

The 30 sampled changes (six changes from every year) were categorized by experts and compared using the following measures obtained from the *apache-cvs* distribution list archive:

1. the number of times a change similar in context to the sampled change is committed. A similar-context change is defined as a change that performs an operation on objects or components that are altered in the sampled change. This roughly measures the amount of collaboration between various developers either over time or over geographical locations. This number would be high if the developers share their changes or design documents or in an *open* development process where the changes or designs are open for public scrutiny and alteration.
2. the number of unique persons who have worked or are working on a change, similar in context to the sampled change, from dispersed locations.[10] This counts all the contributors for the sampled or similar-context changes that are not collocated.

Table 3: Comparison of Architectural and Nonarchitectural Changes

Type of Change	**N**	**Similar Change**		**Number of Contributors**	
		Mean (Std Dev)	**Median**	**Mean (Std Dev)**	**Median**
Architectural	4	1.25 (0.43)	1	1.25 (0.43)	1
Nonarchitectural	26	4.96 (5.95)	3	2.85 (2.38)	2
T-Stat (d.o.f)		3.13[11] (18.79)		3.10[11] (100.6)	

The results obtained are shown in Table 3. We can reject the null hypothesis that architectural and nonarchitectural changes involve the same number of developers and have the same number of similar-context changes. The results obtained above show us that architectural changes are carried out by fewer, possibly collocated, developers. The smaller number of contributors could indicate that fewer developers possess the ability to perform architecture design.

3.2.3 Summary

Apache as an example of OSS is developed, not by a random set of hackers angry at what they see as commercial exploitation of information and knowledge, but by professionals working in different sectors, from government organizations and universities to commercial enterprises. Care seems to have been exercised in the formation of a core group and the proc-

[10] We count individuals from all past similar-context changes instead of just counting the individuals in that particular change itself to protect ourselves from instances of selective attribution. This would happen if individuals do not acknowledge similar contributions more than a few weeks prior to their own changes.

[11] $p < 0.10$

ess for accepting or rejecting changes to the software. The core group appears to be concerned with issues relating to the schedule and size of the software. The responsibility for architectural changes is typically in the hands of a few individuals. In other words, Apache seems to have been organized and developed as though by a commercial software vendor.

3.3 Enhydra Case Study

Enhydra is an open source e-business application server that was designed for the development and deployment of Java-based Extensible Markup Language (XML) applications using servlets. The term *application server* can mean many different things depending on the context in which it is used. In this case it can be thought of as an application that provides services for other applications. Applications are typically built on top of application servers.

Enhydra was not always open source; initially it was developed by Lutris Technologies, Inc. to assist Lutris Consulting in building Web applications. Lutris decided to build its own application server because when it needed this type of technology, the commercially available products were very immature.

On January 15, 1999, after four years of development, Lutris open-sourced Enhydra, assumed the role of the open source maintainer/product champion, and also began the role of being a support provider for its open source product. Lutris sells it own productized version of Enhydra that includes professional documentation, platform certification for popular hardware configurations, installation support, and maintenance updates. Enhydra customers can also purchase additional technical support and training.

When a commercial business is trying to profit from an OSS product, the business will often make outlandish claims with respect to the quality of the product. The OSS movement itself also fuels these claims (see Section 2.4). For example, the text below comes directly from a Lutris white paper titled “The Enhydra™ Competitive White Paper:”

> *“Benefits of the Open Source Process*
> *The open source process means quality implementations driven by real-world needs. Open source efforts such as Enhydra.org ensure that the technology will support meaningful features as driven by a plugged-in community of consultants and end users.* ***The same community ensures that the code base evolves with the highest possible quality. With thousands of eyes having access to source code, no algorithm is safe from scrutiny. Enhydra undergoes 24x7 worldwide code review.*** *To paraphrase Sun’s Bill Joy, most of the smart people in the world don’t work for you or Sun or Microsoft. With open source, you are using code that has been scrutinized by the best and brightest”* [Young 01].

The general premise that open source efforts support meaningful features driven by those in the community is fair. However, a claim that the Enhydra code base is of the highest quality since it is under the scrutiny of thousands of eyes that never sleep is outlandish. A Lutris Web page makes the following claim:

> *"Industry leaders recognize that open source methodology results in highly reliable, stable code. A thriving worldwide community reviews source code, contributes fixes, and reviews designs. The result is reliability that surpasses that of any single, private development and quality assurance team*" [Lutris 01a].

Again, feeding the assumptions that all open source products are highly reliable, stable, and of the highest quality, and conversely that all high-quality software must be open source, will most likely place you in a precarious position.

And finally a Lutris sales brochure states the following:[12]

> *"Open Source Software:*
> ***Lower Risk:*** *Complete access to the code means you don't have to take anyone else's word that there are no back doors, viruses, bugs, or other time bombs waiting to explode. Only Lutris Enhydra can promise the unparalleled security of Lutris product support, training and consulting services right alongside an Open Source community of more then 10,000 Enhydra developers who are always ready to help.*
>
> ***Puts you in control:*** *Open Source software gives you complete control over the success of your Internet applications. Instead of spending most of your budget on software licenses, Lutris Enhydra lets you redirect dollars where they really count: development and support of your applications.*
>
> ***Ensures the highest quality standards:*** *Only the Open Source peer review process can promise that quality code is always maintained and that decisions are made based on quality standards, not market forces or other pressures."*

These claims that OSS is of the highest quality are quite astonishing. Naturally, they are intended to convince prospective users of OSS that it is better than conventional (and yes, more expensive) commercial products. With more than 10,000 Enhydra developers as the Lutris sales brochure claims, or for that matter even 1,000 developers (an order of magnitude less than Lutris claims), you would believe that a cursory review of the source and documentation would not uncover any obvious coding errors, poor programming practices, or documentation deficiencies.

[12] This was an unsolicited direct-mail marketing piece received by our office during February 2001. Copies are available by request.

3.3.1 Objective of the Study

Since it was difficult to believe that the people who were writing OSS for free would change their coding habits between home and work, we conducted a cursory review of the Enhydra source code and documentation to determine if the claim that Enhydra, being OSS, is of the highest quality has merit. It is also important to note that claims of OSS being of the highest quality have not only been made by Lutris but also by the OSS community itself [Netscape 97, O'Reilly 98, OSI 01c].

3.3.2 Conducting the Study

The Enhydra source code that we decided to use for this study was version 3.1.1b1. During this study, we reviewed Java and C source-code files, as well as some of the provided documentation. Our intent in this study was not to do a formal and exhausting code and documentation review but rather to try to determine whether claims that have been made with respect to software quality and Enhydra were true.

3.3.2.1 Packaging

After unzipping the software, we examined the structure of the source-code directory and the provided documentation. The code, documentation, and install scripts were laid out in a very nice and convenient directory-tree structure. This made it very easy to find the relevant material for the review.

3.3.2.2 Documentation Review

Next, we reviewed the documentation that was provided in the initial download. Naturally, we started with the `README` file to determine how to install the software. To our surprise, this file directed us to the Lutris Web site for installation information.

On their Web site, additional online documentation is available from Lutris free of charge, as long as you register by providing your name, address, company, telephone number, and email address. We suspect that Lutris requires registration to view any documentation so that it can pursue prospective customers. Lutris also has other documentation available for downloading, but only for customers who purchased the commercially packaged version of Enhydra.

We first decided to review the Enhydra documentation that is available through the Internet to noncustomers (people who didn’t buy the productized version of Enhydra). It appeared to be well written and comparable with other commercial products that we have reviewed in the past.

Returning to the documentation packaged with the downloaded software, we continued the OSS documentation review. This documentation was marginal at best and very limited. It would be very difficult and time consuming to use this OSS product without the additional documentation that is available from the Lutris Web site. While it may be possible to use the downloadable version of the OSS product with the online documentation that is available for nonpaying customers, it would most likely be best to purchase the commercially packaged version in order to save a considerable amount of time becoming familiar with the product. So in this case, it is our determination that the OSS-supplied documentation is inadequate and far below accepted, commercial standards.

Poor documentation being provided with the OSS version of Enhydra is no accident; Lutris is well aware of the disparity between the OSS and commercial offerings [Lutris 01b]. This conclusion is based on the comparison that Lutris provides on its Web site of its commercially packaged and OSS versions of Enhydra. Parts of this comparison are shown in Table 4.

Table 4: Parts of the Lutris Comparison of Commercially Packaged and OSS Versions of Enhydra [Lutris 01b]

Lutris[®] Enhydra	Enhydra.org
Documentation	
• Detailed installation instructions • Lutris Enhydra Getting Started Guide • Lutris Enhydra Developer's Guide • Lutris Enhydra Wireless Application • Online VoiceXML documentation • Lutris Knowledge Base	• Detailed installation instructions
Samples	
• Getting started • Using multiple clients from a single application • Using the included XSLT parser • Using HTML, Flash, WML, XHTML, cHTML, J2ME, and VoiceXML with Lutris Enhydra	• Getting started • Using HTML with Enhydra
Wireless Development Suite	
• Support for WML, J2ME, cHTML, XHTML, VoiceXML • Comprehensive documentation • Samples showing the use of HTML, Flash, WML, cHTML, XHTML, VoiceXML • Comprehensive Web Mail demo application, EnhydraMail, with HTML and WML clients	• Support for WML and J2ME • No documentation • No bundled tools • No samples

This comparison points out all of the additional documentation and examples included in the commercially packaged version of Enhydra. Recall that Lutris is both the caretaker and champion of this OSS product. Also note that the online installation instructions seem to be adequate, but appear to be used by Lutris as bait to get potential customers to its commercial Web site from its OSS Web site.[13]

[®] Lutris is a registered trademark of Lutris Technologies, Inc.

[13] The Web page located at <http://enhydra.enhydra.org/software/downloads/enhydra311 /index.html#instinstr> says (in part), "Installation instructions are available from Lutris Technologies as part of their commercial version of Enhydra."

3.3.2.3 Code Review

One of the most common C-programming errors is the failure to check the return values that indicate success or failure for system function calls [Miller et al. 00]. This error is quite common when performing memory allocations and should not be found in code that claims to be high quality or world class. So, given the claims by Lutris and the OSS community with respect to the quality of OSS, we did not expect to find this type of programming error in the Enhydra C source code. Additionally, if this common C-programming error is present, it is usually a good indication that the software does not have an adequate error-handing model.

We searched the C source for `MALLOC` function calls to verify that memory-allocation errors were being handled correctly. We found that a `NULL` return value from a `MALLOC` function call was not handled correctly in the C modules shown in Table 5. This mistake occurred in the C source over 80 times.

We also noticed the following code commenting/documentation problems while performing this review:

1. The preamble information in the beginning of the source code (such as the name of the file) did not always match the filename.
2. There were variations in the format of function descriptions.
3. The quality of function descriptions ranged from acceptable to nonexistent.
4. Overall the code was poorly or sparsely commented.

Table 5: C Modules Failing to Check `MALLOC` Return Values in C Source Code

C Source File	Number of Occurrences Where a `NULL` Return Value from `MALLOC` Is Not Checked
edir_conf_xml.c	2
edir_exception.c	1
edir_ipc.c	2
edir_kvtable.c	6
edir_load_choose.c	1
edir_load_multiproc.c	1
edir_load_scoreboard.c	1
edir_load_status.c	1
edir_mutex_filesys.c	5
edir_mutex_sysv.c	4
edir_refcount_simple.c	2
edir_refcount_sysv.c	7
edir_shmem_mmap.c	6
edir_shmem_sysv.c	3
edir_state.c	1
edir_string.c	9
edir_table.c	4
edir_xml.c	4
edir_isapi_filter.c	3
edir_isapi_handler.c	2
HTMLParser.c	2
parser.c	4
xpath.c	1
edir_nsapi_handle.c	4
edir_nsapi_init.c	3
autochange.c	5
jserv_wrapper_unix.c	3
Total	**87**

Next we reviewed the Java source code, examining exception handling as an indication of good error handling or as a good model for error handling. Often only high-quality software has really good or exceptional error handling. During our casual review of the exception handling in Java, we found a number of cases where exceptions were caught and subsequently ignored. Examples are shown in Table 6. The error handling in the Java code does not appear to conform to any particular model and for the most part appears to be ad hoc. The documentation within Java source code was similar to our review of the C code. We noted the following code commenting/documentation problems in the Java code:

1. There were variations in the format of function descriptions.
2. The quality of function descriptions ranged from acceptable to nonexistent.
3. Overall the code was poorly or sparsely commented.

Table 6: Examples of Ignored Exceptions in Java Source Code

Java Source File	**Code Fragment**

`CloneableDO.java`
lines 88-91
	`try {`
	`obj = super.clone();`
	`} catch (CloneNotSupportedException ex) {`
	`// Should not happen`
	`}`

`ContentMD5Module.java`
lines 151-152
	`Catch (IOException ioe)`
	`{ return; }    // shouldn’t happen`

`DefaultAuthHandler.java`
lines 705-706
	`Catch (ParseException pe)`
	`{ }`

`DefaultAuthHandler.java`
lines 1049-1050
	`Catch (Throwable t)`
	`{ }`

`DefaultAuthHandler.java`
lines 1075-1076
	`Catch (ArrayIndexOutOfBoundsException aioobe)`
	`{ }`

`AuthorizationModule.java`
lines 213-214
	`Catch (AuthSchemeNotImplException asnie)`
	`{ /* shouldn’t happen */ }`

`CharIndexedInputStream.java` line 84	`Catch (IOException e) { }`

We also discovered a large number of `FIXME` statements and unimplemented routines. This type of code is not usually found in high-quality software. Some examples of unimplemented routines are shown in Table 7.

Table 7: Examples of Unimplemented Functionality in Java Source Code

Java Source File and Problem	**Code Fragment**

`RemoteZipResource.java`
lines 165-168
Function is not implemented.
	`public long getCurrentLastModifiedTime() {`
	` return -1; //FIXME: implement`
	`}`

`RemoteDirResource.java`
lines 117-119
Function is not implemented.
	`public long getCurrentLastModifiedTime() {`
	`        return -1; //FIXME: IMPLEMENT`
	`    }`

`HttpsRequest.java`
lines 393-395
Function is not implemented.
	`public String getCharacterEncoding() {`
	`        return null; //FIXME`
	`}`

As part of this review, we submitted the coding errors that we found to the Enhydra bug-reporting email address: bugs@enhydra.org. We found it astonishing that such a large software-development community (Lutris claims to have thousands of developers) uses only email for bug reporting without the support of collaborative bug tracking and updating tools.

3.3.3 Summary

Based on our review of the source code, we concluded that the Enhydra source code is no better than commercial source code we have reviewed in the past. The code as a whole is not outstanding, but it is not terrible either; it is simply average. Like any software-development effort, there are routines that are well written and commented as well as others that are not. It appears in this case that the *many eyes* code-review assertion has not been completely effective, given that our review was casual and tended to look for common coding errors and poor programming practices.

The documentation provided with the Enhydra source code was far below any commercial standard. As we stated earlier, we believe that the poor documentation issue is the result of a business trying to profit from OSS. Other OSS projects that we have looked at in the past have had much better documentation than that provided with the Enhydra OSS.

3.4 The NAIS Case Study

The NASA Acquisition Internet Service (NAIS) is an Internet-accessible information system permitting anonymous access to information relating to competitive solicitations and other procurement-related documentation [NAIS 01]. The NAIS is run by the Marshall Space Flight Center and has been operating since 1994.

Like many Web sites, the NAIS evolved from a traditional Web site with static pages that had to be updated manually to one that is updated dynamically. To that end, the NAIS uses Web servers and Common Gateway Interface (CGI) scripts to broker requests issued from Web browsers to back-end databases to construct Web pages that are specific to a visitor’s request. The back-end database of the NAIS was Oracle from Oracle Corporation. In an article to Government Computer News, John Sudderth wrote that the small operating and maintenance budget of the NAIS was being threatened by price restructuring for the Oracle database server [Sudderth 00]. As such, the NAIS began the transition to MySQL[14] and completed that switchover in November 2000 [Trimble 00].

3.4.1 Objective of the Study

One of the predominant axioms of the OSS paradigm is the (potential) involvement of the end user in the development, maintenance, and debugging of the OSS product. The term *user* denotes the person who is downloading or acquiring the OSS product for the purpose of actual integration and/or use in an operational or business setting. As a user, it is believed that this person will participate actively in the forward development of the OSS product. That development may be as small as reporting bugs or as large as actively developing new features for the OSS product.

It is this axiom, or perception, that was being evaluated in this case study. And unlike the other case studies where we focused on a product or some aspect of the open source development process, this case study looked to an adopter of an OSS product: in this case, the NAIS.

3.4.2 Conducting the Study

The switch by the NAIS to an open source database server was precipitated by an increase in fees for the licensing of and support for a commercial product. However, there was little information about how the selection of an alternative product was conducted (commercial or otherwise), what role the database server plays in the overall system, and how experienced and involved the NAIS is in the open source community.

To obtain that information, we asked the NASA project leader for the NAIS to follow up on the interviews conducted by the media with the NAIS, so that we could learn more about its experience using MySQL and the role of that program within its system. This request was granted, so we sent a list of 27 questions to the NAIS. These questions were designed to help us understand

- the size of the NAIS itself (in terms of number of users, frequency of Web hits, and database size)

[14] MySQL is an open source, relational, database-management system. More information on it is available at <http://www.mysql.com>.

- the experience that the NAIS gained using MySQL (how it was selected, deployed, and maintained)
- the explicit involvement of the NAIS in the MySQL open source community

The complete list of questions and the NAIS responses to them are included in Appendix A.

3.4.2.1 The NAIS and the Database Server

The database server plays a significant role in the NAIS system. Predominantly, the data contained in the database is procurement-related data stored in over 1,000 tables. Responses to queries posed by visitors to the NAIS Web site will involve correlating data across those database tables (averaging two to four tables per query). Further, the NAIS reports that it experiences on the order of 1,900 Web hits per day, with each one likely resulting in access to the back-end database server. Therefore in a given 24-hour period, you should expect the NAIS to operate at an average rate of about 75-80 database queries per hour. Obviously, the NAIS serves an important role at NASA, but overall, as an operational system, the NAIS and its reported database-server access is fairly small.

3.4.2.2 The Selection of MySQL

As mentioned in the media articles above and confirmed in the questionnaire to the NAIS, the new pricing structure for the existing commercial product instigated the search and subsequent evaluation of a new database server. What is interesting though is that the new search appears to have been limited to only open source products. During the search for a new database server, the NAIS team reported looking at four other open source products (other than MySQL), namely GNU SQL Server, mSQL, Postgresql, and SQLight. Although it is reasonable to ask whether other commercial alternatives were considered, the questionnaire responses indicate that only open source products were. In a question asking how MySQL compares to commercial database products, NAIS team members responded that they did not compare test results from MySQL to a range of commercial products (but pointed us to the MySQL open source site for such comparisons).

Best cost appears to be the overall determining factor for selecting the database server. Paramount is the fact that the new database search was started based on cost restructuring imposed on the NAIS by the commercial database vendor. To eliminate and/or to reduce that cost, only open source alternatives were considered (all of which can be downloaded at no cost). However, the NAIS team did consider support costs in its evaluation, which is a strong indication that being *free* was not the overriding concern. The NAIS team did respond that the availability of support was a criterion for evaluation (along with cost).

After considering cost (essentially normalizing the field of candidates on this point), other criteria were also considered, including speed, industry acceptance, standard adherence, maintainability, and ease of use. Although no specific units of measure were provided in the responses to the questionnaire, it is clear that other factors were addressed in the evaluation, but there is no indication of the expressed usefulness of the measures that were used. When asked about the specific advantages of the chosen measures over the others that were evaluated, the NAIS listed speed, robustness, security, and ease of use—overall characterizing those measures as the best match for the needs of the NAIS (in light of deficiencies specific to MySQL).

3.4.2.3 The NAIS and the OSS Development Community

There is little doubt that the NAIS team is a skilled development and maintenance team. It has developed the back-end database tables, CGI scripts, and a periphery of Web artifacts (HTML pages, images, etc.) to support the NAIS mission. However, in the NAIS team's use of MySQL, it was essentially treating the program as a black-box component, not unlike the manner in which it would be *forced* to treat a commercial product (since such a product does not come with source code). For instance, source-code inspection was one avenue of evaluation that was not conducted. This is not surprising based on the presumed criticality of the NAIS system: it is not mission-safety critical. The only dichotomy to this position taken from the NAIS response is that security played a role in the evaluation of open source database servers. That is, if security was a criterion in the evaluation, and given the functional specifications of the product's security features, the product's claims could be supported only by a review of the source-code implementation of those features.[15]

There are other aspects of the use of MySQL by the NAIS that closely mirror commercial offerings, such as bug reports, documentation, support, and product modification. Specifically the NAIS, as a user of MySQL, has reported flaws in the software (i.e., bugs) to MySQL via the support channel along with other users of MySQL. Next the NAIS relies on product documentation for day-to-day use and reports having to contact the open source developers to clarify certain points. Further, the NAIS acquires releases of MySQL through a Web site provided to all users who have support contracts. Finally, the NAIS has not modified MySQL for its specific use. This is all typical of commercial software.

Training options for MySQL are also typical of commercial offerings of database servers with course offerings available from the vendor as well as third parties. Training ranges from free (online newsgroups) to authorized training partners. In this area the NAIS training for developers and administrators has been via in-house knowledge (of the subject area) and available documentation and books on MySQL.

However, there is one aspect that is markedly different in the use of MySQL by the NAIS with respect to commercial software: it builds the binaries for MySQL from the source code

[15] Often OSS supporters claim that security through obscurity is not security at all and want source code to verify security algorithms and architecture.

rather than downloading the binaries from the MySQL Web site (which is available through the support contract).

3.4.2.4 The Experience of the NAIS with MySQL

The selection of MySQL by the NAIS certainly addressed the core risk of being “expensed” out of a commercial product. Based on the responses to the questions asked, the NAIS appears to be fully satisfied with its selection and subsequent use of MySQL. The NAIS commented numerous times on the cost and speed advantage that it has over the existing commercial offering. Given the frequency of use and dependency on the database server, the high licensing and support costs for Oracle appear to be overkill for the business application environment needed by the NAIS. Further, the advantages afforded by a “slim and trim” database server having a smaller footprint with respect to memory and CPU resources and lacking the overhead burden of unused features (such as stored procedures, database triggers, transaction rollbacks, and row-level locking) underscore a seemingly wise choice for MySQL over Oracle.

3.4.3 Summary

Although troubling but not surprising, the NAIS chose not to evaluate other commercial alternatives to Oracle in its search to replace the database server. Other commercial alternatives exist which do not necessarily have the same capabilities as Oracle, but are on the same scale as MySQL, such as Solid’s database Server and Microsoft Corporation’s Access 2000.[16] Could other commercial alternatives have supported the needs of the NAIS, perhaps even better? And given those choices, what would the short-term and long-term costs be, comparatively, between MySQL and those commercial databases?

Obviously, the entry costs for OSS, namely MySQL, are far less than just about any commercial alternative (i.e., free), but the entire spectrum of costs, which include development, integration, maintenance, and sustainment costs, should be considered.

There is no doubt that OSS, as a category, is a viable source for software components from which to build systems. Large commercial firms such as IBM, Amazon, and Lycos are users of the Apache open source Web server. NASA and the NAIS project approached their dilemma (licensing costs) by turning to the open source marketplace, applying traditional evaluation techniques in selecting the right product for their application and mission. And in this case, there seems to be an excellent match between the NAIS and MySQL.

[16] The MySQL Web site, located at <http://www.mysql.com/information/benchmarks.html>, lists a number of open source and commercial closed-source relational databases. The fact that they are listed there should not be construed as an endorsement.

Cost should not be the only factor to consider when selecting open source products. Conversely, software that has no acquisition cost should not be eliminated as a choice. A proper evaluation of viable alternatives, commercial or otherwise, should be conducted to ensure that the capabilities of the software match the overall mission needs of the system and its users.

3.5 Teardrop Case Study

Incidents of security-related attacks have been increasing steadily since the mid-1980s [CERT 01a]. Although there is no single event that can be identified as the sole contributor to this increase, it is likely that the ever increasing dominance of the WWW and the estimated hundreds of millions of people who are now interconnected play a role [Telcordia].

This vast population is made up of a variety of users. Some of those are simply the curious; others are looking for the next bargain in e-commerce. Many are the architects, designers, implementers, and the workforce of the digital age—building the systems that are the underpinnings of our cities, economies, governments, and national defense. And there are a few, the cyber terrorists, who seek to wreak havoc on the rest.

The cyber terrorists use information about the software in our systems as ammunition against us. Just about any piece of information can help a cyber terrorist. This information may include specific versions of software, for instance a specific version of an operating system, database, or Web server. Specific knowledge about the behavior of the software when used in a particular context can be exploited. For example, a specific Web server with this database on this specific operating system is known to fail in a very specific manner [Hissam 97]. What the cyber terrorist is looking for is information that is key to orchestrating an attack to achieve some particular purpose, like a service interruption or economic gain, or to make a socioeconomic/political statement. What the cyber terrorist is looking for is vulnerability.

3.5.1 Security Through Obscurity

Vulnerabilities can come in various forms. Most common are bugs in some specific implementation of an algorithm. But these are not the only vulnerabilities. Some vulnerabilities can stem from how a piece of software or an entire system is architected or designed. Vulnerabilities may also come in the form of poor requirements, flawed specifications, or even how the system is operated [Neumann 95, Ellison et al. 97]. Regardless of the source, the key is being able to identify and then exploit a vulnerability.

To exploit a vulnerability, the cyber terrorist has to find the information, understand the potential for damage, and then design and perfect an attack. Information about software components can come from a variety of sources such as vendor Web sites, bug lists, FAQs, and security reports. Information can also be obtained through black-box testing and visibility techniques. What makes this difficult is correlating all the sources of information and raw data to discover and understand a vulnerability in order to create an attack.

However, there is another source of information about a software component: the source code itself. If a cyber terrorist has the source code to the software components in a system, the job of correlating all the sources of information becomes much easier—questions about the implementation and the design can be answered by inspecting the source code itself. When dealing with software components from commercial vendors, access to the source code is rarely possible. However, systems are coming to rely more upon OSS as a source of components from which to achieve functionality. This means that cyber terrorists have greater access to the source.

This does not necessarily mean that systems are more vulnerable because they are comprised in whole or in part of OSS. What it does mean is that cyber terrorists who want to attack a system have access to more information about those constituent components than they would in systems comprised of software where the source code is not available (e.g., CSS). And with more information, there is more opportunity for discovery.

There is a continuing debate over whether there is security through obscurity. Put another way, are systems comprised of CSS more secure because the source code is not available for inspections (hence obscured)? The positions are strong on both sides of the argument [Whitlock 01, Lipner 00, McGraw 00, Neumann 00]. But regardless of the side you take, one thing is clear: with the source code, insight into how an OSS component works and its implementation is there for all to see, scrutinize, and—if you are a cyber terrorist—exploit.

3.5.2 Linus's Law: A Premise of OSS

Recall Linus's Law: "Given enough eyeballs, all bugs are shallow." Essentially, this is stating that given enough developers looking at any particular piece of source code, any flaw in that source code will be blatantly obvious (hence shallow) to at least one of them. Although this is quite a compelling observation, a deeper appreciation of the developer leads to questions about the scrutiny being used by any one of those pairs of developers' eyeballs.

Viega points out many of the motivations that drive developers to look at any one piece of software [Viega 01]. Such reasons include everything from altruism to personal gain; perhaps the developer has found something of particular use. However, it may be the case that those who set out to look at the software are often discouraged for a variety of reasons (time, large amounts of software, lack of necessary knowledge), thereby reducing the number of qualified eyeballs.

Any latent security vulnerability in a piece of software is, for all intents and purposes, a flaw (bug) in the code. It is a flaw in that the software behaves in a manner in which it was not intended to behave. If Linus's Law holds then, naturally some developer will discover the shallow bug, proving that "Given enough eyeballs, all security flaws are shallow." And like before, at least one developer should be able to inspect, detect, and repair the flaw—hence the security flaw is obvious to someone.

If this were indeed true, the OSS community should, over time, be producing invulnerable software as all security flaws are detected and repaired. But for this to be true, you must assume that OSS is in a steady state in that only bugs are being removed and no new (potentially flawed) functionality is being introduced. But this is not the case. While all kinds of bugs are being squashed, others are being introduced with each new piece of functionality and new versions of software. So before software becomes invulnerable, latent flaws will continue to persist.

Lastly, we have to consider Viega's observations regarding the motivations behind why developers look at the source code. It may well be the case that the flaw is blatantly obvious, not to the altruistic developer, but to the cyber terrorist. If the shallow vulnerability is obvious to the altruistic developer, the vulnerability will be detected, repaired, and returned to the OSS code base for all to partake. However, if the shallow vulnerability is first obvious to cyber terrorists, they have the opportunity to wreak havoc on the user community and perhaps even make a name for themselves.

At first brush, Linus's Law appears to paint a euphoric notion of only the goodness that comes from OSS development methods by ferreting out software bugs. However, the law also explains other behaviors and nefarious uses of the source code that come along with OSS. The law indirectly points out an inherent race condition between the cyber terrorist and the OSS development community: just exactly who will find the shallow security vulnerability first? Will it be the cyber terrorist or the altruistic developer?

3.5.3 From the Cyber Terrorist's Workbench

Like the OSS developers, the cyber terrorist is a participant in the overall OSS development activity (whether or not the cyber terrorist chose such a role). Time and time again, attacks perpetrated by the cyber terrorist are followed by alerts (such as those generated by the CERT Coordination Center [®]) and repairs. Interestingly, some view the role of the cyber terrorist as something close to a liberated quality-control inspector [Thomas et al. 00]!

Aside from such a positive spin, there is no question that the OSS movement has made the life of the cyber terrorist somewhat easier. Source code gives cyber terrorists additional tools for their workbench, tools that they can use to carry out attacks that without the source code would be much more difficult to orchestrate. Although there is some rationale for believing

[®] CERT and CERT Coordination Center are registered in the U.S. Patent and Trademark Office.

that cyber terrorists are actually helping to make OSS less vulnerable, one has to question the cost that is being incurred as a result of their QA inspections.

For a moment, we step outside ourselves to take a look at OSS not from the perspective of the altruistic developer but from that of the cyber terrorist.

3.5.3.1 Cyber Terrorist's View #1: "What I Like About OSS"

Predominantly the source code is the greatest asset to me. The source code is the blueprint and the key. I have insight into what the software is supposed to do and what it is not supposed to do. I do not have to rely on documentation to tell me what the software can and cannot do; I can see that for myself. There is no such thing as undocumented functionality. The documentation is there, one line at a time. The code also tells me precisely how the software's functionality is carried out—step-by-step and flaw-by-flaw.

Such insight into the software permits me to get as close to the source code as I want. With such intimacy, I can learn very sophisticated and direct attacks with such pinpoint accuracy it is like I wrote the software myself. And since I have the source code and the means to build it myself, I can work diligently in the privacy of my lair fine-tuning my attack without alerting anyone to my intentions. Once I have verified and tested my attack, I can unleash my lethal attack on the unsuspecting world.

One other aspect of OSS that is really appealing to me is the voracity with which OSS is released onto the net. Sometimes it appears as if there is a race going on between many of the OSS projects today: for instance, who can get out what feature first and beat their competitors. This often means that releases occur often but not necessarily right, resulting in software containing many bugs.

3.5.3.2 Cyber Terrorist's View #2: "What I Don't Like About OSS"

Some of the properties of the OSS community that work to my advantage also work to my disadvantage, particularly speed and source code. Once I have released my attack on the user community, fixes or countermeasures are devised quickly. Rather than having to deal with one individual or one small group of developers, I have to deal with an entire community of developers acting as cyber sleuths.

Since the source code is available to me, it is also available to my competitors—other cyber terrorists who are motivated to unleash their attacks just as fiercely as I am. It is possible that, while I spend my energy fine-tuning my attack, cyber sleuths may repair the flaw I plan to exploit before I get the opportunity to release the attack. It is also possible that other cyber terrorists have identified the vulnerability and perfected their attack prior to my release. Unfortunately, those are the risks that I take in my business.

The ultimate attack is one in which there is no countermeasure. Unfortunately, such a cyber terrorist's nirvana is unlikely to appear for quite a long time, as each attack and responding countermeasure only serves to strengthen and improve the software over time.

3.5.3.3 Cyber Terrorist's View #3: "Even More Things I Like About OSS"

But even the things that I dislike about OSS also work to my advantage. When a patch is released for a piece of OSS, whether it is a response to an attack (i.e., countermeasure) or the repair of a flaw, there is much that can be learned from the patch itself. First, the patch can sometimes alert me to a vulnerability that was previously unknown to myself and other cyber terrorists. Furthermore, the patch and the underlying flaw can point me to other similar software systems that may exhibit the same flaw. Such a situation can occur in software that extends from the same root-code base. For instance, a bug discovered and repaired in an OSS variant of FTP may point to a similar flaw in another OSS variant of FTP. One example of this is bind's "nxt record bug" [CERT 01b].

Perhaps less known is the insight that OSS source code can give me into CSS systems such as COTS software. As in the OSS community, CSS can sometime emanate from the same root-code base. Knowledge that some particular piece of CSS is shared or perhaps descendant from a similar piece of OSS gives me clues as to what attacks could be possible against a CSS system. Again returning to the FTP example, if a commercial offering of an operating system includes a variant of FTP, possibly a descendant of `wu-ftpd`,[17] it may be susceptible to flaws already patched in the root-code base for `wu-ftpd`. I can turn my attention to the `wu-ftpd` patches and use what I learn from them to attack the unpatched commercial offering of FTP.

Finally, given the nature of OSS, the source code to the countermeasure is also available. With the source, counterattacks to the countermeasure can be devised or abandoned more quickly. If a review of the source code to a countermeasure shows additional vulnerability, the design and implementation of a counterattack is much easier to create. If however, the countermeasure instituted by the OSS community is sound, I can determine, again through inspection, that my time is better spent working on the next attack for another flaw. I do not have to spend my time trying to get around the countermeasure in the dark; the countermeasure's implementation is there in the open for me to see.

We now suspend our "out-of-body" experience in the mind of the cyber terrorist and examine a real attack that occurred a few years ago, to illustrate some of the advantages afforded the cyber terrorist.

[17] `Wu-ftpd` is the affectionately known name for the OSS FTP daemon, `wuarchive-ftpd`, that was developed at Washington University by Bryan P. O'Connor. For more information, go to <http://www.wu-ftpd.org>.

3.5.4 Teardrop: One Perspective of the Attack

Earlier we surmised the advantages and disadvantages of OSS from the perspective of the cyber terrorists. Perhaps the most interesting, if not most chilling, issue is the insight into CSS that can be achieved through OSS. This insight is not limited to situations where both an OSS and CSS program share the same root-code base, but can occur when both OSS and CSS share the same architecture, design, or specification. One case in point is the specification for the Internet Protocol (IP) [DARPA 81].

Going back to 1997, sparse reports were appearing in the Internet news groups about a denial-of-service (DoS) attack against the Linux operating system. This attack, called Teardrop, was a full frontal assault on the operating system's IP stack that is implemented in kernel space. The essence of the attack was to construct two or more abnormal IP packets and transmit those packets to an unsuspecting Linux host with the flawed IP stack. Upon receipt of those packets, the Linux kernel would crash, denying the use of that host until the system was restarted.

What made this attack attract our attention was the level of intimacy needed with Linux's implementation of the IP stack to pull it off. What also made this attack interesting was that it was a heterogeneous DoS attack in that it not only affected Linux kernels prior to version 2.0.32, but also affected Windows 95 and Windows NT versions 3.5 and 4.0 (prior to service pack 3). The fact that the perpetrator had to have such intimate knowledge of the implementation details and its effect on a CSS product truly warranted investigation.

Teardrop worked because the IP packets that it constructed would not be generated normally by properly functioning IP stacks. What Teardrop did to crash the Linux kernel was to produce IP packets that were marked as fragmented packets (as per the specification) and introduce incorrect offsets and payload sizes into the IP header. When interpreted by the Linux kernel, the combination of fragmented IP packets and malformed offsets and sizes caused the kernel code to generate a negative length (signed integer value) that was actually treated as a large positive number by the kernel (unsigned integer value). Upon reassembly of the fragmented packets, the kernel was then tricked into moving too much data, thereby overflowing allocated kernel buffers and, in most cases, causing the operating system to crash. Reviewing the Linux kernel from the period (specifically version 2.0.30), we were able to trace the code and reproduce the IP fragmentation flaw.

The flaw stems from an ambiguity in the IP specification about how to handle IP fragments that overlap with prior fragments where the payload is nonexistent or very small (smaller than the fragment claims to be). The inspection of this anomalous condition is shown in Figure 3.

```
 1 /* Copy the data portions of
   all fragments into the new buffer. */
 2 fp = qp->fragments;
 3 while(fp != NULL)
 4 {
 5  if(count+fp->len > skb->len)
 6  {
 7   NETDEBUG(printk(
      "Invalid fragment list: Fragment over size.\n"));
 8   ip_free(qp);
 9   kfree_skb(skb,FREE_WRITE);
10   ip_statistics.IpReasmFails++;
11   return NULL;
12  }
13  memcpy((ptr + fp->offset), fp->ptr, fp->len);
14  count += fp->len;
15  fp = fp->next;
16 }
```

Figure 3: Fragmentation Flaw

The programming error that was exploited by the authors of the Teardrop DoS attack is located at line number 5 in Figure 3. This code is part of the IP packet assembly logic that takes a set of fragmented IP packets and combines them into one complete IP packet by copying each fragment into a contiguous data area. Line 5 of the code verifies that there is enough space left in the contiguous data area before a fragment is copied to that area. This code expects that the value of `fp->len` is greater than or equal to zero and will work fine as long as that assumption is true. However, if `fp->len` is less than zero, the code allows the memory copy in line 13 to occur with the negative value of `fp->len` being treated as a very large positive number (used to indicate number of bytes to copy) by the `memcpy()` function. This memory copy with erroneous data causes unintended data to be overwritten, thus crashing the operating system. The programming was corrected by changing line 5 to read

```
      if (fp->len < 0 || count+fp->len > skb->len)
```

The corrected code now verifies that the length of the IP fragment is valid (greater than or equal to zero) before the data copy is allowed to occur; otherwise it returns an error.

What is curiously interesting about this attack is that another operating system, Windows 95/NT, was also its victim, making it a heterogeneous attack upon a very specific condition. It is possible that both IP stack implementations suffered from the ambiguity in the IP specification and were engineered forward subsequently with the same defect. However it is also plausible that both IP stack implementations shared a common root ancestry and were distant cousins, siblings, or twins! But in any case, the perpetrator of Teardrop, knowing precisely how to crash the Linux kernel IP stack with only a pair of IP fragmented packets, was also able to initiate an identical DoS attack (only with additional IP fragmented packets) against a CSS operating system.

This may have just been dumb luck. But something even more interesting occurred after the OSS community released the patch for Linux. In the patch to `ip_fragment.c`, there was a repair of another anomalous condition in the state transition of IP fragment reassembly that had not been exploited by the cyber terrorists. The OSS community, with forethought of purpose, closed the door on another DoS attack. This is a great example of OSS development at work and a credit to the patch's authors.

Initially unofficial patches were released to the Internet newsgroups. Eventually, the official patch made its way into Linux kernel version 2.0.32. Around that same time the CERT Coordination Center issued an advisory coordinating the announcement with both commercial vendors and the OSS community whose operation systems may have also been affected [CERT 01c].

But the OSS patch for the Linux kernel had an unintended side effect. The patch advertised to the world (or anyone who was reading the newsgroups or looking at the patch source code) that there was a second flaw in the IP stack that could be vulnerable to variations on the Teardrop version of malformed, IP packet fragments. Soon after the release of the CERT advisory, other variants of Teardrop (counterattacks) began to appear (e.g., Bonk, Boink, NewTear). Since Linux introduced the patch to this untapped vulnerability, it was immune to these attacks; however Microsoft Corporation (the developer of Windows) was not. The Teardrop variants quickly ripped through the Microsoft IP stack. To Microsoft Corporation's credit, it quickly released a second patch on the heels of the Teardrop variants, repairing that vulnerability.

This study reveals that it is possible for cyber terrorists to learn about one or more vulnerabilities in a CSS product from OSS source code and patches.

3.5.5 Summary

OSS brings an entirely new parameter to the equation for system integrators. OSS is not only a viable source of components from which to build systems, but the source code enables the integrator to discover other properties of the component that are not typically available when using CSS components. Unfortunately there is a cost to this benefit, as cyber terrorists also gain additional information about those components and discover vulnerabilities at a rate which is comparable to those looking to squash bugs.

This is not to say that security through obscurity is the answer. There is no doubt that sunshine kills bacteria. That is, the openness of OSS development can lead to better designs, better implementations, and eventually better software over time. However, until a steady state in any software release can be achieved, the influx of changes, rapid release of software (perhaps before its time), and introduction of new features and, invariably, flaws will continue to feed the vicious cyclic nature of attack and countermeasure.

4 Lessons Learned/Observations

4.1 Trends Towards OSS Use and Development

There is no doubt that industry has accepted and embraced OSS. This is because some OSS products have proven to be just as reliable and secure as similar commercial products that are on the market today. Both Linux and Apache have played a big role in gaining industry confidence in OSS.

According to the Netcraft Survey, a survey of Web server software usage on Internet-connected computers, the Apache Web OSS server has been the predominant Internet Web server since 1996, with a current market share of about 60% [Netcraft 01]. The survey also shows the Linux operating system with a current market share of about 30%. These statistics are a good indicator of industry's acceptance of and trust in OSS products.

Industries, specifically those in the software-development business, are beginning to support the idea of development in the style of OSS. For example, Borland, a company that produces software-development tools such as integrated-development environments, compilers, and databases, has open-sourced some of its products. To date, Borland has open-sourced its component library for Kylix, a rapid application development (RAD) environment for the Linux platform, and a database called Interbase. The idea of commercial companies making their products open source, as a business practice, has not become mainstream, and it is difficult to determine whether this will become the trend.

When OSS became popular, companies were created to profit from the OSS movement. The focus of these companies or their Web sites was usually to profit or promote their products by providing

1. Web-based collaborative environments for OSS development
2. OSS incubators—Web sites that host OSS projects and provide a Web-based collaborative software-development environment
3. OSS project databases and repositories that catalog information about different OSS projects
4. third-party support or value-added packaging for a suite of OSS products

CollabNet is an example of a company that sells a product for Web-based, collaborative, OSS development called SourceCast. CollabNet also offers consulting services to companies that want to set up collaborative software-development environments and practices based on open source principles. Initially CollabNet was focused on helping companies set up OSS development projects but over time has expanded its scope to include collaborative software development within an enterprise.

The development of software within an enterprise is sometimes called *corporate-source software* (in the past, prior to the contemporary OSS advocacy campaign, this was referred to as distributed software development). Corporate-source software is basically proprietary software that is developed usually across globally disparate communities with select customers, corporate groups, partners, and businesses. The idea of distributed software development has been around for over 10 years. However, the availability of Web-based COTS tools to facilitate distributed software development is a fairly new concept.

Companies that run OSS incubator Web sites, such as SourceForge.net (which is run by VA Linux), use this Web site to promote OSS and market their OSS collaboration software. The expense to maintain this free Web site is augmented by additional revenue from advertisements that are embedded within the site's Web pages. VA Linux also markets its OSS, Web-based, collaborative software-development environment to industry for use in the development of corporate-source software.

The Web sites for OSS project databases and repositories often have advertisement-based revenue schemes as well. Others run this type of Web site to promote their OSS collaboration software, such as Freshmeat, which is also funded by VA Linux. It is important to note that companies such as VA Linux can make the OSS movement look bigger through the creation and funding of various OSS Web sites.[18]

The third-party OSS vendor tends to try to profit from OSS by providing professional documentation, packaging, and support. These types of companies often provide true value to the OSS product and are often well worth the additional expense. It is also important to note that some companies occasionally open-source a product that was once being marketed as commercial (CSS) software, only to turn around and then become a third-party vendor to the now OSS product. Often in this case, the company is just looking for another way to profit from a product that for some reason was unable to compete with other commercial products.

[18] VA Linux, Inc. owns the trademarks to the Open Source Development Network (OSDN), Freshmeat, Geocrawler, Linux.com, LinuxGram, NewsForge, Slashcode, Slashdot, SourceForge.net, and themes.org—all of which are portals to the OSS movement.

Many of the companies that were created as a result of the OSS movement have run into financial trouble and are now looking into new ways of marketing their products. At this stage, it is impossible to determine the fate of OSS and whether it will be the trend of the future, but at least for the moment, OSS has received a lot of attention from industry and the software community.

4.2 What It Takes for a Successful OSS Project

In the summer of 2000, we conducted a birds-of-a-feather (Bof) session at the O’Reilly Open Source Conference in Monterey, California. Approximately 30 people attended the session and a lively discussion ensued. The session’s purpose was to attempt to learn what it takes to make an OSS project successful (and by extension, what it means for such a project to be successful.) The participants consisted mostly of OSS developers and users. The success of an open source project is determined by several things, which can be placed loosely into two groups: people and software.

As one participant of the workshop who was paraphrasing Raymond put it, “The best OSS projects are those that scratch the itch of those who know how to code” [Raymond 99]. This is saying that a large potential user community is not enough to make an open source project successful. It also requires a large, or at least dedicated, developer community. Such communities are difficult to come by, and the successful project is likely to be one that meets some need of the developer community.

For instance, the development of an OSS accounting system is less likely to be successful than that of a graphics system. The potential developer pool for the former is much smaller than that for the latter—just because of interest.

The success stories in OSS all seem to “scratch an itch.” Linux, for instance, attracts legions of developers who have a direct interest in improving an operating system for their own use. However, it scratches another important “itch” for some of these folks: it is creating a viable alternative to Microsoft Corporation’s products. Throughout our discussions with groups and individuals, this anti-Microsoft-Corporation sentiment was a recurring theme.

Another successful open source project is the Apache Web server. Although a core group is responsible for most of its development, it is the Webmaster community that actually contributes to its development.

On the other hand, as already detailed in Section 3.1, we found that Zelerate AllCommerce, without serious corporate sponsorship, was unable to sustain itself as a viable open source project. Without being paid, there weren’t enough developers who cared deeply enough to sustain it.

While people issues play a large part in the success of an open source project, there are software issues as well that can be divided into two groups: design and tools.

The poorly thought out initial design of an open source project is a difficult impediment to overcome. For instance, huge, monolithic software does not lend itself very well to the open source model. Such software requires too much up-front intellectual investment to learn the software's architecture, which can be daunting to many potential contributors. A well-modularized system, on the other hand, allows contributors to carve off chunks on which they can work.

An example of an open source project that appears to not be working well because of the structure of the software is Mozilla (the open source Web browser). In order to release Mozilla, Netscape apparently ripped apart Netscape Communicator, and the result, according to Bof participants, was a "tangled mess." Perhaps it is not coincidental that Mozilla is having trouble releasing a product that people actually use (though we understand that people do use pieces of it in other projects).

To its credit, Netscape realized that there was a problem with Mozilla and, in an attempt to help the situation, created a world-class set of open source tools. These tools, such as Bonsai, Bugzilla, and Tinderbox, support distributed development and management and help developers gain insight into Mozilla. While perhaps not true several years ago, the adoption of a reasonable tool base is required for an open source project to have a significant chance of success (if only to aid in the distributed-development paradigm and information dissemination). Tools such as revision-control software and bug-reporting databases are keys to success. Fortunately for the community, organizations like SourceForge.net (www.SourceForge.net) are making such tool sets easily available; this goes a long way towards solving that aspect of the problem.

A final factor in the success of an open source project is time. Corporate software development can be hampered by unrealistically short time horizons. OSS development can be as well. However, in the former case, projects are all too often cancelled before they have a chance to mature, while in the latter case an effort can continue on (perhaps with reduced numbers of people involved). The result may be that an apparent failed open source project becomes a success. Because of this, it is difficult to say that a particular project has failed. As one of the attendees said regarding the GIMP product (a Photoshop-like piece of OSS), "It hasn't failed, it just hasn't succeeded—yet." In fact in the ensuing year, it appeared that the GIMP product was becoming more successful.

4.3 The OSS Development Model

It may not be surprising that the development process for OSS differs from traditional software development. What may be surprising to some is how ultimately similar they are.

Traditional software development starts with a detailed requirements document that is used by the system architect to specify the system. Next comes detailed system design, implementation, validation, verification and, ultimately, maintenance/upgrade. Iteration is possible at any of these steps. Successful OSS projects, while not conducted as traditional (e.g., commercial) developments, go through all of these steps as well.

But the OSS development model differs from its traditional cousin. For instance, requirements analysis may be very ad hoc. Successful projects seem to start with a vision and often an artifact (e.g., prototype) that embodies that vision—at least in spirit. This seems to be the preferred way of communicating top-level requirements to the community for an OSS project. As the community grows, the list of possible requirements will grow as well. Additional requirements or new features for an OSS project can come from anyone with a good (or bad) idea. Furthermore these new requirements actually may be presented to the community as a full-fledged implementation. That is, someone has what he thinks is a good idea, goes off and implements it, and then presents it to the community. Usually this is not the case in a traditional project.

In a traditional project, the system architect will weigh conflicting requirements and decide which ones to incorporate and which to ignore or postpone. This is not done as easily in an OSS development effort where the developer community can vote with its feet. However, successful projects seem to rely on a core group of respected developers to make these choices. The Apache Web server (see Section 3.2) is one example of such a project. This core group is taking on the role of a system architect. If the core group is strong and respected by the community, the group can have the same effect (virtually identical) as determining requirements for a traditional development effort.

Implementation and testing happens in OSS development efforts much as it does for traditional software-development efforts. The main difference is that these activities are often going on in parallel with the actual system specification. Individual developers (core or otherwise) carve out little niches for themselves and are free to design, implement, and test as they see fit. Often there will be competing designs and implementations, at most one of which will be selected for inclusion in the OSS system. It is the core group (for systems so organized) that makes the selections and keeps this whole process from getting out of control.

Finally, to conduct maintenance activities, upgrade, re-release, or port to new platforms, the open source community relies on sophisticated tools for activities such as version control, bug tracking, documentation maintenance, and distributed development. The OSS project that does not have or use a robust tool set (usually open source itself) either has too small a community to bother with such baggage or is doomed to failure. This is also the case for traditional development.

4.4 The Relationship of OSS to CSS

Judging from the press it receives, OSS is something new in the world of software development. To the limited extent that the press itself is sensitive to the term, there is truth to that statement. It would be fair to acknowledge that more people (and not just software engineers) are now sensitive to the term *open source* than ever before—for which we can also thank the press. But what makes OSS *new* to the general, software-systems engineering community is that we are faced with more choices for viable software components than ever before. But you may ask yourself, before what?

4.4.1 The World Before OSS

Before OSS became a popular term, software engineers had three generalized choices for software components. They could be

- built from the ground up
- acquired from another software project or initiative
- purchased from the commercial marketplace

If the component was to be built from the ground up, there were basically two approaches: to actually undertake the development of the component from within the development organization (i.e., in-house), or to negotiate a contract to develop the component via an external software-development organization. Essentially the component was custom built. As such, the software sources were available for the component acquired in this fashion.

One other popular approach was to locate components of similar functionality from other (potentially similar) software projects. The term often used in this context was *reuse* or *domain-specific reuse*. If a component could be located, it could then be adapted for the specific needs of the *using* software-development activity. In U.S. government vernacular, this was also referred to as government off-the-shelf (GOTS) software. Typically reuse libraries and GOTS software would come in binary and source-code form.

Finally, software engineers had the option of looking to the commercial marketplace for software components. Software-development organizations would undergo market surveys trying to locate the components that best fit their needs. Evaluations would commence to determine which of the commercial offerings most closely matched, and a selection would be made. In many instances, the source code was not delivered as part of the component's packaging. In some cases, the source code may have been available for an additional cost (if at all). And in the event that the source code could be bought, there were (and still are) very restrictive limitations placed on what could and could not be done to those sources.

4.4.2 The World After OSS

With the advent of OSS, the community has an additional source of components, which is actually a combination of all three of the above choices. OSS and reusable components are very much alike in that they are both developed by others and often come in binary and source-code form. But like reusable components, it can be challenging to understand what the OSS component does [Shaw 96].

Because it comes with the source, OSS is similar to custom-built software. However, it lacks the design, architectural, and behavioral knowledge inherent to custom-built software. This is also a problem with commercially purchased software. This lack of knowledge allows us to draw a strong analogy between OSS and COTS software in spite of the source code being available for the former and not for the latter.

The SEI has been studying COTS-based systems for a number of years and has learned some important lessons about them, many of which apply directly to OSS.[19]

Organizations adopting an OSS component have access to the source, but are not required to do anything with it. If they choose not to look at the source, they are treating it as a black box. Otherwise they are treating it as a white box. We discuss both of these perspectives below.

4.4.3 OSS as a Black Box

Treating OSS as a black box is essentially treating it as a COTS component; the same benefits and problems will apply. For instance, an organization adopting COTS products should know something about the vendor (e.g., its stability and responsiveness to problems), and an organization adopting OSS should know something about its community.

If the community is large and active, the organization can expect that the software will be updated frequently, that there will be reasonable quality assurance, that problems are likely to be fixed, and that there will be people to turn to for help. If the community is small and stagnant, it is less likely that the software will evolve, that it will be well tested, or that there will be available support.

Organizations that adopt COTS solutions are often too small to have much influence over the direction in which the vendor evolves the product [Hissam et al. 98]. Black-box OSS is

[19] See the COTS-Based Systems (CBS) Initiative Web site at <http://www.sei.cmu.edu/cbs>.

probably worse in this regard. A COTS component will change due to market pressure, time-to-market considerations, the need for upgrade revenue, and so forth. OSS components can change for similar market reasons, but can also change for political or social reasons (factions within the community), or because someone has a good idea—though not necessarily one that heads in a direction suitable to the organization.

Organizations that adopt COTS products can suffer from the vendor-driven upgrade problem: the vendor dictates the rate of change in the component, and the organization must either upgrade or find that the version it is using is no longer supported. This same problem exists with OSS. The software will change, and eventually the organization will be forced to upgrade or be unable to benefit from bug fixes and enhancements. The rate of change for an eagerly supported OSS component can be staggering.

Organizations that adopt COTS solutions often find that they have to either adapt to the business model assumed by the component or pay to have the component changed to fit their business model [Oberndorf et al. 99]. We have found that adapting the business model usually works out better than changing the component [Brownsword et al. 00]. Once you change a component, you *own the solution*. If the vendor does not accept your changes, you'll be faced with making them to all future versions of the software yourself.

For black-box OSS, it may be easier for a change to make its way back into the standard distribution. However, the decision is still out of the organization's control. If the community does not accept the change, the only recourse is to reincorporate the change into all future versions of the component.

Because of a lack of design and architectural specifications, undocumented functionality, unknown pre- or post-conditions, deviations from supported protocols, and environmental differences, it is difficult to know how a COTS component is constructed without access to the source code. As a consequence, it can be difficult to integrate the component. With OSS, the source is available, but consulting it means that the component is no longer being treated as a black box.

4.4.4 OSS as a White Box

Because the source is available, it is possible to treat OSS as a white box. It therefore becomes possible to discover platform-specific differences, uncover pre- and post-conditions, and expose hidden features and undocumented functionality. With this visibility comes the ability to change the components as necessary to integrate them into the system.

However, sometimes the source is the only documentation that is provided. Some consider this to be enough. Linus Torvalds, the creator of Linux, has said, "Show me the source." Yet if this were the case, there would be no need for Unified Modeling Language (UML), use cases, sequence diagrams, and other sorts of design documentation. Gaining competency in the OSS component without these additional aids can be difficult.

An organization that treats OSS as a white box has a few key advantages over one that treats it as a black box. One advantage is the ability to test the system knowing exactly what goes on inside the software. Another advantage is the ability to fix bugs without waiting for the community to catch up. A seeming advantage is the ability to adapt the system to the organization's needs. But as we've already discussed, the rejection of your change by the community means that you own the change and have given up many of the benefits of OSS.

4.4.5 Summary

Just as for COTS components, OSS components require substantial skills to understand, evaluate, use, and integrate. OSS improves upon COTS products because access to the source lets an organization gain insight into the component and, if warranted, repair it by making adaptation directly to the source code.

4.5 Acquisition Issues

According to the President's Information Technology Advisory Committee (PITAC)

> *"Existing federal procurement rules do not explicitly authorize competition between open source alternatives and proprietary software. This ambiguity often leads to a de facto prohibition of open source alternatives within agencies"* [PITAC 00].

The PITAC recommends that the federal government allow open source development efforts to "compete on a level playing field with proprietary solutions in government procurement of high-end computing software." We wholeheartedly endorse that recommendation.

In the presence of such a level playing field, acquiring OSS would not be fundamentally very different from acquiring COTS software. The benefits and risks would be similar and both must be judged on their merits.

We've already discussed issues such as security in the open source context, so we won't consider them here. Those sorts of issues aside, there are two risks that an organization acquiring OSS faces:

- that the software won't exactly fit the needs of the organization
- that ultimately there will be no real support for the software

We'll address each of these in turn.

A key benefit of OSS is that the sources are available, allowing them to be modified as necessary to meet the needs of the acquiring organization. While this is, indeed, a benefit, it also introduces several significant risks. Once the OSS is modified, many open source licenses require the organization to *give* the changes back to the community. For some systems this may not be a problem, but for others there may be proprietary or sensitive information involved. Thus it is very important to understand the open source license being used.

As discussed in the preceding section on CSS, the fact that a modification is given back to the community does not mean that the community will embrace it. If the community doesn't embrace it, the organization faces a serious choice. It can either stay with the current version of the software (incorporating the modifications) or move on to updated versions—in which case, the modifications will have to be made all over again. Staying with the current version is the easy thing to do, but in doing so you give up some of the advantages of OSS.

With COTS software there is always the risk of the vendor going out of business, leaving the organization with software but no support. This can be mitigated, somewhat, by contract clauses that require the escrowing of the source code as a contingency. No such escrow is needed for OSS. However, in both cases, unless the organization has personnel capable of understanding and working with the software's source code, the advantage of having it available is not clear. Certainly there would be tremendous overhead should there be a need to actually use the source code; essentially by taking it over, you are now in the business of producing that product.

Most government software is acquired through contracts with contractors. A contractor proposing an open source solution in a government contract needs to present risk-mitigation plans for supporting the software just as it would have to do if it were proposing a COTS product. In the case of a COTS product, this might include statements regarding the stability of the vendors involved. No such statement is valid regarding OSS. The community surrounding an open source product is not guaranteed to be there when needed, nor is it guaranteed to care about the support needs of the government. Furthermore if the proposing contractor is relying on the OSS community to either add or enhance a product feature or accept a contractor-provided enhancement in the performance of the government-funded, software-development contract, the government should expect a mitigation if the OSS community does not provide such an enhancement or rejects the enhancement outright. Thus the ultimate support of the software will fall on the proposing contractor.

There are, of course, unique benefits of OSS—many of which have been discussed elsewhere in this report. From an acquisition point of view, the initial cost of OSS is low. Also, at least for significant open source products, it is likely (but by no means guaranteed) that the quality of the software will be on par with many COTS solutions. Finally, when modifications are needed, it is guaranteed that they can be made in OSS. For COTS software there is always the possibility that the vendor will refuse. (But, as we've seen, the ability to modify is also a source of risk.)

4.6 Security Issues

Trust in the software components that are in use in our systems is vital, regardless of whether the software comes from the bazaar or the cathedral. As integrators, we need to know that software emanating from either realm has been reviewed and tested and does what it claims to do. This means that we need *eyes* that look beyond the source code and to the bigger picture. That is the holistic and system view of the software—the architecture and the design.

Others are beginning to look at the overall OSS development process [Feller et al. 01, Nakakoji et al. 01]. More specifically from the Apache case study (above), we observed what type of contributions have been made to the Apache system and whether those who made them were core or non-core Apache developers. We learned that a majority (90%) of changes to the system (implementation, patches, feature enhancements, and documentation) were carried out by the core-group developers, while many of the difficult and critical architectural and design modifications came from even fewer core developers. Non-core developers contributed to a small fraction of the changes. What is interesting is that the Apache core developers are a relatively small group compared to the non-core developers—in fact the size of the core developers is on par with the typical size of development teams found in CSS products.

This is not intended to imply that OSS lacks architectural and design expertise. Actually, the Apache modular architecture is likely central to its success. However, even with the existence of a large community of developers participating actively in an OSS project, it is questionable to the extent that many *eyes* are really critiquing the holistic view of the system's architecture and design, looking for vulnerabilities.

This is not just an issue for OSS: it is a problem for CSS as well. That is, in CSS we have to trust and believe that the vendor has conducted such a holistic review of its commercial software offerings. We have to trust the vendor because there is little likelihood that any third party can attest to a vendor's approach to ridding its software of vulnerabilities. This specific point has been a thunderous charge of the OSS community, and we do not contest that assertion. But we caution that just because the software is open to review, it should not automatically follow that such a review has actually been performed (but of course you are more than welcome to conduct that review yourself—*welcome to the bazaar*).

5 Conclusions

5.1 Making Lightning Strike Twice

Instances of successful OSS products such as Linux, Apache, Perl, sendmail and much of the software that makes up the backbone of today's Web are clear indications that successful OSS activities can strike often. But like lightning we can ask, "Is it possible to predict where the next strike will be?" or, "Is it possible to make the next strike happen?"

Continuing with this analogy, we can answer these questions to the extent that science will permit. Like lightning, meteorologists can predict the likelihood of severe weather in a metro region given atmospheric conditions. For OSS it may be harder to predict the likelihood of success for an OSS product or activity, but certain conditions appear to be key, specifically

- **It is a working product**. Looking back at many of the products, especially Apache and Linux, none started in the community as a blank slate. Apache's genesis began with the end of the National Center for Supercomputing Applications (NCSA) Web server. Linus Torvalds released Linux version 0.01 to the community in September 1991. Just a product concept and design in the open source community has a far less likely chance of success. A prototype, early conceptual product, or even a *toy* is needed to bootstrap the community's imagination and fervor.
- **It has committed leaders**. Equally as important is a visionary or champion of the product to chart the direction of the development in a (relatively) forward direction. Although innovation and product evolution are apt to come from any one of the hackers in the development community, at least one person is needed to be the *arbiter of good taste* with respect to the product's progress. This is seen easily in the Apache project (regarding the Apache Foundation).
- **It provides a general community service**. This is perhaps the closest condition to the business model for commercial software. It is unlikely that a commercial firm will bring a product to market if there is no one in the marketplace who will want to purchase that product. In the open source community, the same is also true. Raymond points out a few valuable lessons:
  - "*Every good work of software starts by scratching a developer's personal itch.*"
  - *"Release early, release often. And listen to your customers."*
  - *"To solve an interesting problem, start by finding a problem that is interesting to you"* [Raymond 99].

  From these lessons, there is a theme that talks to the needs of the developers themselves (e.g., personal itch) and a community need (e.g., customers or consumers who need the product or service).

- **It is technically *cool***. You are more likely to find an OSS device driver for a graphics card than an accounting package. Feller and Fitzgerald categorized many of the open source projects in operation, noting that a high percentage of those were Internet applications (browsers, clients, servers, etc.), system and system-development applications (device drivers, code generators, compilers, and operating systems/kernels), and game and entertainment applications [Feller et al. 02].
- **Its developers are also its users**. Perhaps the characteristic that is most indicative of a successful OSS project is the developers themselves also being the users. Typically this is a large difference between OSS and commercial software. In commercial software, users tend to convey their needs (i.e., requirements) to engineers who address those needs in the code and then send the software back to the users to use. A cycle ensues with users conveying problems and the engineers fixing and returning the code. However in OSS, it is more typical that a skilled engineer would rather repair the problem in the software and report the problem *along with the repair* back to the community. The fact that OSS products are technically *cool* explains why many of the most popular ones are used typically by the developer community on a day-to-day basis. (Not many software developers we know use accounting packages!)

This is not to say that any product or activity exhibiting these conditions will, in fact, be successful. But those products that are considered to be successful meet all of them.

This leads us to the next question: “Is it possible to make the next strike happen?” In lightning research, scientists use a technique called *rocket-and-wire technique* to coax lightning from the skies to the ground for research purposes. In that technique and under the optimum atmospheric conditions, a small rocket is launched trailing a ground wire to trigger lightning discharges [UFECE]. For OSS, a comparable technique may involve creating conditions that are favorable to OSS development but may fail to instigate a discharge from the OSS community.

At this point, we abandon our lightning analogy and observe (and dare predict) that there will be other successful OSS products and activities in the coming years. Furthermore, we surmise that such products will exhibit the conditions discussed above. Whether they happen by chance or by design is difficult to tell.

5.2 In Closing…

The SEI views OSS as a viable source of components from which to build systems. However, we are not saying that OSS should be chosen over other sources simply because the software is open source. Rather, like COTS and CSS, OSS should be selected and evaluated on its merits. To that end, the SEI supports the recommendations of the PITAC subpanel on OSS to remove barriers and educate program managers and acquisition executives and allow OSS to compete on a level playing field with proprietary solutions (such as COTS or CSS) in government systems.

Adopters of OSS should not enter into the open source realm blindingly and should know the real benefits and pitfalls that come with OSS. The fact that OSS is open means that everyone can know the business logic encoded in the software that runs those systems. That means that anyone is free to point out and potentially exploit the vulnerabilities with that logic—anyone could be the altruistic OSS developer or the cyber terrorist. Furthermore, having the source code is not necessarily the solution to all problems: without the wherewithal to analyze or perhaps even to modify the software, it makes no difference having it in the first place.

It should not follow that OSS is high-quality software. Just as in the commercial marketplace, the bazaar contains very good software and very poor software. In this report, we have noted at least one commercial software vendor that has used its role in the OSS community as a marketing leverage point touting the "highest quality software" when, in fact, it is no better (or worse) than commercial-grade counterparts. *Caveat emptor* (let the buyer beware); the product should be chosen based on the mission needs of the system and the needs of the users who will be the ultimate recipients.

References/Bibliography

[Apache 99] The Apache Foundation. “About the Apache HTTP Server Project” [online]. <http://httpd.apache.org/ABOUT_APACHE.html> (October 2001).

[Arief et al. 01] Arief, B.; Gacek, C.; & Lawrie, T. *The Many Meanings of Open Source* (CS-TR-737). Newcastle, England: Department of Computer Science, University of Newcastle upon Tyne, August, 2001.

[Asundi 01] Asundi, J. “Issues in Software Development: Outsourcing, Design and Organization.” PhD diss., Carnegie Mellon University, 2001.

[Barksdale 99] Barksdale, J. Written Testimony from Netscape Communications Corporation’s CEO James Barksdale, United States v. Microsoft Corporation, Civ. No. 98-1232, Paragraphs 47-68.

[Bass et al. 98] Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice* (ISBN: 0201199300). Reading, MA: Addison-Wesley, 1998.

[Brooks 87] Brooks, Frederick P. “No Silver Bullet: Essence and Accidents of Software Engineering.” *Computer 20*, 4 (April 1987): 10-19.

[Brownsword et al. 00] Brownsword, L. & Place, P. *Lessons Learned Applying Commercial Off-the-Shelf Products: Manufacturing Resource Planning II Program* (CMU/SEI-99-TN-015, ADA228027). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, June 1999. <http://www.sei.cmu.edu/publications/documents/99.reports /99tn015/99tn015abstract.html>.

[CERT 01a] CERT Coordination Center. “Number of Incidents Reported” [online]. <http://www.cert.org/stats/ cert_stats.html#incidents> (July 2001).

[CERT 01b] CERT Coordination Center. “CERT® Advisory CA-1999-14 Multiple Vulnerabilities in BIND” [online]. <http://www.cert.org/advisories/CA-1999-14.html> (2001).

[CERT 01c] CERT Coordination Center. “CERT® Advisory CA-1997-28 IP Denial-of-Service Attacks” [online]. <http://www.cert.org/advisories/CA-1997-28.html> (2001).

[Clements et al. 01] Clements, P.; Kazman, R.; & Klein, M. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2001.

[Cohen 60] Cohen, J. “A Coefficient of Agreement for Nominal Scales.” *Educational and Psychological Measurement 20*, 1 (Spring 1960): 37-46.

[Cusumano et al. 95] Cusumano, M. A. & Selby, R. W. *Microsoft Secrets*. New York, NY: The Free Press, 1995.

[DARPA 81] DARPA. “Internet Protocol, DARPA Internet Program Protocol Specification” (RFC-791). Marina Del Ray, CA: Information Sciences Institute, University of Southern California, September 1981. Available FTP: <ftp://ftp.isi.edu/in-notes/rfc791.txt>.

[Dibona et al. 99] Dibona, C.; Stone, M.; Ockman, S.; et al. *Open Sources: Voices from the Open Source Revolution* (ISBN: 1565925823). Sebastopol, CA: O’Reilly & Associates, January 1999.

[Ellison et al. 97] Ellison, R.; Fisher, D.; Linger, R.; Lipson, H.; Longstaff, T.; & Mead, N. *Survivable Network Systems: An Emerging Discipline* (CMU/SEI-97-TR-013, ADA341963). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, November 1997. <http://www.sei.cmu.edu/publications/documents/97.reports/97tr013/97tr013abstract.html>.

[Feller et al. 01] Feller, J. & Fitzgerald, B. “A Framework Analysis of the Open Source Software Development Paradigm,” 58-69. *Proceedings of the International Conference on Information System.* Brisbane, Australia, December 10-13, 2000. Atlanta, GA: International Conference on Information Systems, 2001.

[Feller et al. 02] Feller, J. & Fitzgerald, B. *Understanding Open Source Software Development* (ISBN: 0201734966). Boston, MA: Addison-Wesley Professional, UK, March 2002.

[Fielding 99] Fielding, R. T. “Shared Leadership in the Apache Project.” *Communications of the ACM 42,* 4 (April 1999): 42-43.

[Fleiss 81] Fleiss, J. L. *Statistical Methods for Rates and Proportions*. New York, NY: John Wiley & Sons, 1981.

[Hassan et al. 00] Hassan, A. E. & Holt, R. C. “A Reference Architecture for Web Servers.” *Proceedings of the Working Conference on Reverse Engineering, WCRE 2000.* Brisbane, Australia, November 23-25, 2000. Los Alamitos, CA: IEEE Computer Society, 2000.

[Hecker 99] Hecker, F. “Setting Up Shop: The Business of Open Source Software.” *IEEE Software 16*, 1 (January/February 1999): 45-51.

[Herbsleb et al. 99] Herbsleb, J. H. & Grinter, R. E. “Splitting the Organization and Integrating the Code: Conway’s Law Revisited.” *Proceedings of the 21st International Conference on Software Engineering.* Los Angeles, CA, May 16-22, 1999. New York, NY: Association for Computing Machinery, 1999.

[Hissam 97] Hissam, S. “Case Study: Correcting System Failure in a COTS Information System.” *SEI Monographs on the Use of Commercial Software in Government Systems* (monograph). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, October 1997. <http://www.sei.cmu.edu/cbs/papers /monographs /case-study-correcting/ case.study.correcting.htm>.

[Hissam et al. 98] Hissam, S.; Carney, D.; & Plakosh, D. *SEI Monograph Series: DoD Security Needs and COTS-Based Systems* (monograph). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 1998.

[Hubbard 95] Hubbard, J. Part of Ch. 1, “A Brief History of FreeBSD.” *FreeBSD Handbook* [online]. <http://www3.au.freebsd.org/handbook/history.html> (October 2001).

[ISS 01]] Internet Security Systems, Inc. “Teardrop IP Fragmentation” [online]. <http://xforce.iss.net/static/338.php> (2001).

[Leonard 00] Leonard, A. Salon Free Software Project: Chapter 1, “Boot Time” [online]. <http://www.salon.com/tech/fsp/2000/03/06/chapter_one_part_2/index.html> (October 2001).

[Lipner 00] Lipner, S. B. “Security and Source Code Access: Issues and Realities,” 124-125. *Proceedings of the IEEE Symposium on Security and Privacy*. Berkeley, California, May 14-17, 2000. Los Alamitos, CA: IEEE Computer Society, 2000.

[Lutris 01a] Lutris Technologies, Inc. “*Deploying Lutris Enhydra Applications*” [online]. <http://www.lutris.com/products/enhydra3_5/inDepth/deployment.html> (September 28, 2001).

[Lutris 01b] Lutris Technologies, Inc. “*Why Buy Lutris Enhydra*” [online]. <http://www.lutris.com/products/enhydra3_5/whybuy.html> (October 2001).

[McGraw 00] McGraw, G. “Will Openish Source Really Improve Security?” 128-129. *Proceedings of the IEEE Symposium on Security and Privacy.* Berkeley, CA, May 14-17, 2000. Los Alamitos, CA: IEEE Computer Society, 2000.

[Miller et al. 00] Miller, B.; Koski, D.; Lee, C.; Maganty, V.; Murthy, R.; Natarajan, A.; & Steidl, J. “Fuzz Revisited: A Re-Examination of the Reliability of UNIX Utilities and Services.” Madison, WI: Computer Sciences Department, University of Wisconsin, February 2000.

[Mockus et al. 00] Mockus, A.; Fielding, R. T.; & Herbsleb, J. “A Case Study of Open Source Software Development: The Apache Server,” 263-272**.** *Proceedings of the 22nd International Conference on Software Engineering.* Limerick, Ireland, June 4-11, 2000. New York, NY: Association for Computing Machinery, 2000.

[NAIS 01] NASA Acquisition Internet Service. “Frequently Asked Questions” [online]. <http://nais.msfc.nasa.gov/cgi-bin/NAIS/faq.cgi> (September 2001).

[Nakakoji et al. 01] Nakakoji, K. & Yamamoto, Y. “Taxonomy of Open Source Software Development,” 41-42. *Making Sense of the Bazaar: Proceedings of the 1st Workshop on Open Source Software Engineering* held during the 23rd International Conference on Software Engineering. Toronto, Canada, May 12-19, 2001. Los Alamitos, CA: IEEE Computer Society, 2001.

[Netcraft 01] Netcraft. “Netcraft Web Server Survey” [online]. <http://www.netcraft.com/survey> (October 2001).

[Netscape 97] Netscape Communications Corporation. “Netscape Communicator Open Source Code White Paper” [online]. <http://home.netscape.com/browsers/future/whitepaper.html> (1997).

[Netscape 98] Netscape Communications Corporation. “Netscape Announces Plans to Make Next-Generation Communicator Source Code Available Free on the Net” [online]. <http://www.netscape.com /newsref/pr/newsrelease558.html> (1998).

[Neumann 95] Neumann, P. *Computer-Related Risks* (ISBN: 020155805X). Reading, MA: Addison-Wesley, October 1995.

[Neumann 00] Neumann, P. “Robust Nonproprietary Software,”122-123. *Proceedings of the IEEE Symposium on Security and Privacy*. Berkeley, California, May 14-17, 2000. Los Alamitos, CA: IEEE Computer Society, 2000.

[Oberndorf et al. 99] Oberndorf, P. & Foreman, J. “Lessons Learned from Adventures in COTS-Land,” track 8 on CD-ROM. *Proceedings of the 11th Annual Software Technology Conference.* Utah State University, Salt Lake City, Utah, May 2-6, 1999. Hill AFB, UT: Utah State University-Extension in cooperation with the Software Technology Support Center, 1999.

[O’Reilly 98] O’Reilly & Associates. “Open Source Pioneers Meet In Historic Summit” [online]. <http://press.oreilly.com/opensource.html> (April 1998).

[O’Reilly 99] O’Reilly, T. “Ten Myths about Open Source Software” [online]. <http://opensource.oreilly.com/news/ myths_1199.html> (1999).

[OSI 01a] Open Source Initiative. “The Open Source Definition Version 1.8” [online]. <http://www.opensource.org/docs/definition.html> (2001).

[OSI 01b] Open Source Initiative. “OSI Certification Mark and Program” [online]. <http://www.opensource.org/docs /certification_mark.html> (2001).

[OSI 01c] Open Source Initiative. “Advocacy: The Open Source Case for Business” [online]. <http://www.opensource.org/advocacy /case_for_business.html> (2001).

[Parnas 72] Parnas, D. L. “On the Criteria to Be Used in Decomposing Systems into Modules.” *Communications of the ACM 15*, 12 (December 1972): 1053-58.

[PITAC 99] President’s Information Technology Advisory Committee (PITAC): co-chairs Joy, B. & Kennedy, K. “Report to the President, Information Technology Research: Investing in our Future” [online]. <http://www.ccic.gov/ac/report/pitac_report.pdf> (February 1999).

[PITAC 00] President's Information Technology Advisory Committee (PITAC), Panel on Open Source Software for High End Computing: co-chairs Smarr, L. & Graham, S. "Developing Open Source Software to Advance High End Computing" [online]. <http://www.ccic.gov/pubs /pitac/pres-oss-11sep00.pdf> (September 11, 2000).

[Raymond 99] Raymond, E. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary* (ISBN: 1565927249). Cambridge, MA: O'Reilly & Associates, October 1999.

[Salus 94] Salus, P. *A Quarter Century of UNIX* (ISBN: 0201547775). Reading, MA: Addison-Wesley Publishing Co., June 1994.

[Schneider 00] Schneider, F. B. "Open Source in Security: Visiting the Bizarre," 126-127. *Proceedings of the IEEE Symposium on Security and Privacy*. Berkeley, California, May 14-17, 2000. Los Alamitos, CA: IEEE Computer Society, 2000.

[Schnoll 01] Schnoll, S. "The History of Internet Explorer" [online]. <http://www.nwnetworks.com/iehistory.htm> (2001).

[Shaw 96] Shaw, M. "Truth vs. Knowledge: the Difference Between What a Component Does and What We Know It Does," 181-185. *Proceedings of the 8th International Workshop on Software Specification and Design*. Schloss Velen, Germany, March 22-23, 1996. Los Alamitos, CA: IEEE Computer Society, 1996.

[Shaw et al. 96] Shaw, M. & Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. UpperSaddle River, NJ: Prentice-Hall, 1996.

[Stallman 98] Stallman, Richard. "Linux and the GNU Project" [online]. <http://www.fsf.org/gnu/linux-and-gnu.html> (1998).

[Succi et al. 01] Succi, G.; Paulson, J.; & Eberlein, A. “Preliminary Results from an Empirical Study on the Growth of Open Source and Commercial Software Products,” [online]. *Proceedings of the Third Workshop in Economics-Driven Software Engineering Research (EDSER-3).* Toronto, Canada, May 12-19, 2001. Los Alamitos, CA: IEEE Computer Society, 2001. <http://www.cs.virginia.edu/~sullivan/edser3/> (2001).

[Sudderth 00] Sudderth, J. “NAIS Is Sold on Open Source.” *Government Computer News 19*, 33 (November 2000): 29. <http://www.gcn.com/vol19_no33/enterprise/3275-1.html>.

[Telcordia] Telcordia[™] Technologies, Inc. Telcordia Internet Sizer. <http://www.netsizer.com>.

[Thomas et al. 00] Thomas, B., ed. “A Discussion of Open Source Software.” *SEI Interactive* [online]. <http://interactive.sei.cmu.edu/ Features/2000/March/Roundtable/Roundtable.mar00.pdf> (2000).

[Trimble 00] Trimble, P. “Open Minds on Open Source” [online]. *Federal Computer Week* (December 04, 2000). <http://www.fcw.com/fcw/articles/2000/1204/pol-nasa-12-04-00.asp>.

[UFECE] University of Florida, Electrical and Computer Engineering Department. <http://www.ece.ufl.edu/labs/melemag.html>.

[Viega 01] Viega, J. “The Myth of Open Source Security” [online]. <http://webdeveloper.earthweb.com/websecu/article/0,,12013_621851,00.html> (2001).

[Whitlock 01] Whitlock, N. “Does Open Source Mean an Open Door?” [online]. <http://www-106.ibm.com/developerworks/linux/library/l-oss.html> (2001).

[™] Telcordia is a trademark of Telcordia Technologies, Inc.

[Wilson 99] Wilson, G. “Is the Open Source Community Setting a Bad Example?” *IEEE Software 16*, 1 (January/February 1999): 23-25.

[Young 01] Young, D. “The Enhydra™ Competitive White Paper.” Santa Cruz, CA: Lutris Technologies, Inc, February 2001. <http://www.lutris.com/documentation/whitePapers /competitive_wp_v53.pdf>.

Appendix A The NAIS Questionnaire

Questionnaire & Response

1 NAIS Questions

1. **On the whole, what is the operational profile of the NAIS system? That is, estimated number of daily users? daily queries?**

   Current average daily hits: ~1900
   80-90% of this hits the database in one way or another.
   Internal Users: ~ 800
   External: ~ 5000 (This number reflects users signed up for email notification. The actual user count is higher.)

2. **How large is the NAIS? Could you please elaborate in terms of estimated number of static HTML pages, CGI scripts, Web server instances (threads)?**

   The file count for code behind the NAIS (scripts, images, HTML, libraries, etc.) is 800+. If you include content files that are not part of the code behind the NAIS, the number jumps easily into the two-thousand range.

3. **How large is the NAIS relational database? That is, number of tables? Number of entities? Complexity of queries (average number of tables involved in common NAIS queries)?**

   Table count is 1000+; new data warehousing efforts could drive it to around 10,000 in the near future.

   Query count is high, as the applications do a great deal of verification to the database during processes.

   Average table count on queries is 2 - 4.

4. **What information is maintained in the NAIS database?**

   The NAIS stores numerical, character, and binary data. Sizes range from one character to Blob fields. Data content is procurement related.

2 NAIS and MySQL

5. **Was MySQL compared/evaluated against other databases?**
   Yes

   If so then:

   5.1. What were the comparison/evaluation criteria?
   Speed
   Industry acceptance / Standards adherence
   Installation / Configuration ease and maintainability
   Data recovery ease
   Available support
   Cost

   5.2. What were the other products that were evaluated?

   MySQL was compared to other competitive open source products available at the time of our investigation including Postgresql, mSQL(mini SQL), GNU SQL Server, and SQLight.

   5.3. What were the advantages and disadvantages of MySQL in relation to the other products?

   MySQL was the best match for our needs overall. It lacked some features such as triggers and stored procedures, but was very fast and robust in other areas, and provided excellent security features. MySQL had a very low learning curve, and we were able to become comfortable in a very short period of time.

6. **How was MySQL selected for use in the NAIS? Could you please elaborate on any testing, evaluations, reviews, inspections, that were conducted?**

   A test instance was installed, and we ported our most database-intensive application to the product, analyzing the process and looking for problems we could expect to encounter upon use of the product. Upon completion, the software was load-tested and benchmarked against the existing application. The results of this activity, coupled with technical opinions from each developer, formulated our end recommendation.

7. **Has the NAIS development team participated or contributed to MySQL development? If so, could you please elaborate on the level of participation/contribution (bug reports, documentation fixes, code fixes, feature enhancements, etc.) and whether this involvement is ongoing?**

   The NAIS has reported bugs, though the number has been small. Most were corrected immediately.

8. **Was the MySQL modified in any way for use in the NAIS? If so, please explain.**

   No, the source code was not modified for our use.

9. **How are any bug and product issues with MySQL resolved?**

   We handle all issues through the standard reporting methods available to all MySQL users.

10. **How do you acquire MySQL releases?**

    We acquire releases through the MySQL Web site for all users who have support contracts.

11. **How do you determine when to incorporate a release/upgrade of MySQL into the NAIS?**

    We evaluate our software on a regular basis to keep abreast with new functionality and bug fixes. Normally upgrades are performed only if needed, for example, if a bug has been fixed or a major improvement has been added that would help our service. This policy guarantees the developers and end users a stable environment to work in, with scheduled outages.

12. **Do you build the product yourself or do you use a precompiled binary release?**

    We build all of our binaries from source code and distribute a binary internally.

13. **Do you have a third-party support contract for MySQL? If so, with whom?**

    We have a support agreement with MySQL, but no third parties at this time.

14. **Have members of the NAIS development team reviewed any of the MySQL source code? If so, what where the results of the review? How would they rate the quality of the code?**

    No review of the source code has been performed at this time. We evaluated the product from a functional standpoint and did not address the style and so forth of the code itself.

15. **How would you rate the quality of the MySQL documentation?**

    Overall it is acceptable for day-to-day use. We have needed to contact the developers on a small number of occasions to clarify certain points.

16. **Can you quantify the cost of ownership of MySQL as compared to similar commercial products (i.e., Oracle).**

    Cost of ownership has declined, as the product is provided at no cost, and the support contract cost is minimal. The database is a technically simpler product, with fewer features to master, thus lowering the initial learning curve and annually required training to continue efficient use. Our prior product cost was in the $5,000 annual range. The initial estimated cost for the new pricing structure of our previous product was $750,000 and was later renegotiated. This estimate instigated the new DBMS search. The NAIS was not an issue by the time the contract was completed; therefore the new estimated cost is not available.

3 MySQL as an Open Source Alternative

17. How does MySQL compare to commercial database products? Could you please elaborate on quality, performance, features, support, scalability, etc. Which of these are perceived and which are measured?

Our tests did not compare MySQL to a range of commercial products, however MySQL has performed tests such as this and posts the results for public viewing.

18. What are the major advantages and disadvantages of using MySQL in a development effort?

The major advantages include the cost, availability of support through the many users of the product who participate in online forums, its ease of setup, adherence to standards, low machine resource requirements, speed, and the availability to connect to the database through many technologies including C++, ODBC, and Perl. This helps the product to be very flexible and friendly on any architecture. In our environment and for our needs, there are no disadvantages.

19. How do your developers receive MySQL training?

To date, it has been in-house, via documentation and books. Most developers had a SQL background and were able to begin using the product immediately.

20. How do your administrators receive MySQL training?

To date, it has been in-house via documentation. However we have plans to attend classes to aid in tuning and configuration.

21. Please describe any situations where MySQL would not be a good choice as an open source alternative for a database.

Distributed enterprise applications require certain capabilities that MySQL may not support, due to its lack of built-in functions, such as enterprise management features (row-level locking, rollback, etc.). Mirroring of data is possible, but the mechanisms used are not as powerful as current commercial DBMS packages provide.

22. Please describe the application environment where MySQL is best suited.

Workgroup to mid-range environments using multi-tier application design, including Web development, lend themselves almost naturally to the product.

4 The NAIS and Other Open Source Alternatives

23. According to the interview/article appearing in GCN, plans to evaluate the Apache Web Server have been/are being made by the NAIS to correct limitations of the current Web Server in use. What are those current limitations that will be solved by Apache?

Our current Web server software has configuration and operability limitations that make management difficult. As the number of users has increased, we have found it more difficult to manage our servers effectively. We hope that Apache will offer us some relief.

24. What lessons have been learned from the integration and application of MySQL in the NAIS which will (or may be) applied to the selection, integration, and application of other open source alternatives?

Generally speaking, there are quite a few high-quality options available in open source software for a number of applications today. This number appears to be growing at a fast pace. Including open source products in evaluations is currently the norm within the NAIS technical team. We have found a number of the products to be very stable and capable of production use, and in certain situations more efficient than their commercial competitors. This appears partly due to the fact that in open source software, the quality and performance of the product in many cases are paramount, and the developers do not feel pressured to add unnecessary features that detract from the product's original goal. Add to this the availability of support and the fact that bugs are usually corrected very quickly, and it's very hard not to consider open source options seriously when evaluating products.

Appendix B Open Source Definition, Version 1.8[20]

The Open Source Definition

Version 1.8

> *The indented, italicized sections below appear as annotations to the Open Source Definition (OSD) and are **not** a part of the OSD. A plain version of the OSD without annotations can be found here.*

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

> ***Rationale:*** *By constraining the license to require free redistribution, we eliminate the temptation to throw away many long-term gains in order to make a few short-term sales dollars. If we didn't do this, there would be lots of pressure for cooperators to defect.*

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

> ***Rationale:*** *We require access to un-obfuscated source code because you can't evolve programs without modifying them. Since our purpose is to make evolution easy, we require that modification be made easy.*

3. Derived Works

[20] The Open Source Definition shown here appears exactly as it does on the Open Source Initiative's Web site; it has not been edited [OSI 01a].

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

> ***Rationale:*** *The mere ability to read source isn't enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.*

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

> ***Rationale:*** *Encouraging lots of improvement is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.*
>
> *Accordingly, an open source license* ***must*** *guarantee that source be readily available, but* ***may*** *require that it be distributed as pristine base sources plus patches. In this way, "unofficial" changes can be made available but readily distinguished from the base source.*

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

> ***Rationale:*** *In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore we forbid any open source license from locking anybody out of the process.*
>
> *Some countries, including the United States, have export restrictions for certain types of software. An OSD-conformant license may warn licensees of applicable restrictions and remind them that they are obliged to obey the law; however, it may not incorporate such restrictions itself.*

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

> ***Rationale:*** *The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. We want commercial users to join our community, not feel excluded from it.*

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

> ***Rationale:*** *This clause is intended to forbid closing up software by indirect means such as requiring a non-disclosure agreement.*

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

> ***Rationale:*** *This clause forecloses yet another class of license traps.*

9. License Must Not Contaminate Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open source software.

> ***Rationale:*** *Distributors of open source software have the right to make their own choices about their own software.*
>
> *Yes, the GPL is conformant with this requirement. GPLed libraries "contaminate" only software to which they will actively be linked at runtime, not software with which they are merely distributed.*

Appendix C Acronym List

ACM	Association of Computing Machinery
Bof	birds of a feather
BSD	Berkeley Software Distribution
CASE	computer-aided software engineering
CEO	chief executive officer
CERT/CC	CERT Coordination Center
CGI	Common Gateway Interface
cHTML	Compact Hypertext Markup Language
CMU	Carnegie Mellon University
COTS	commercial off-the-shelf
CPU	central processing unit
CSS	closed-source software
CVS	concurrent version system
DARPA	Defense Advanced Research Projects Agency
DB	database

DBMS	database management system
DDoS	distributed denial of service
d.o.f.	degrees of freedom
DoS	denial of service
EDSER	Economics Driven Software Engineering Research
ENIAC	Electronic Numerical Integrator and Calculator
FAQ	frequently asked questions
FCW	Federal Computer Week
FSF	Free Software Foundation
FTP	File Transfer Protocol
GCC	GNU ‘C’ Compiler
GCN	Government Computer News
GIMP	GNU Image Manipulation Program
GNU	GNU’s Not Unix *(a recursive acronym)*
GOTS	government off-the-shelf
GPL	General Public License
GUI	Graphical User Interface
HTML	Hypertext Markup Language

HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IE	Microsoft Corporation’s Internet Explorer
IEEE	Institute for Electrical and Electronics Engineers
IP	Internet Protocol
IPO	initial public offering
ISBN	International Standard Book Number
J2ME	Java 2 Micro Edition
JDK	Java Development Kit
MPM	Multi-Process/Multi-Threading Model
MR	modification request
NAIS	NASA Acquisition Internet Service
NASA	National Aeronautics & Space Administration
NCSA	National Center for Supercomputing Applications
NSPR	Netscape Portable Runtime
ODBC	open database connectivity
OSD	Open Source Definition
OSDN	Open Source Development Network

OSI	Open Source Initiative
OSS	open source software
PACT	Project for Advancement of Coding Techniques
PITAC	President’s Information Technology Advisory Committee
QA	quality assurance
RAD	rapid application development
RFC	request for comment
SEI	Software Engineering Institute
SQL	Structured Query Language
SWIC	Software Industry Center
UML	Unified Modeling Language
USL	Unix Systems Lab
WCRE	Working Conference on Reverse Engineering
WML	Wireless Markup Language
WWW	World Wide Web
XHTML	Extensible Hypertext Markup Language
XSLT	Extensible Stylesheet Language Transformations
XML	Extensible Markup Language

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE November 2001	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Perspectives on Open Source Software		5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) Scott Hissam, Charles B. Weinstock, Daniel Plakosh, Jayatirtha Asundi		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2001-TR-019
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-2001-019
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Open source software (OSS) is emerging as the software community's next *silver bullet* and appears to be playing a significant role in the acquisition and development plans of the Department of Defense (DoD) and industry. Yet, as with all previous silver bullets, there are problems with blindly embracing the OSS paradigm.

To become familiar with the benefits and pitfalls of using OSS, the Software Engineering Institute (SEI) undertook an internally funded study looking at it from various perspectives:

- the user of OSS
- the developer of OSS
- the organizations looking to deploy software systems comprised (partially or completely) of OSS components

During the period of this study, members of the SEI technical staff hosted meetings, conducted interviews, participated in open source development activities, workshops, and conferences, and studied available literature on the subject. Through these activities we have been able to support and sometimes refute common perceptions about OSS. This report is the result of our study.

14. SUBJECT TERMS open source software, OSS, free software, FSF, case studies, research, open source license, COTS			15. NUMBER OF PAGES 96
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL