THE ECONOMICS OF OPEN SOURCE SOFTWARE
A Model for Total Cost Analysis

Richard Lloyd Barnes
B.S., California State University, Sacramento, 2000

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

BUSINESS ADMINISTRATION
(MANAGEMENT INFORMATION SCIENCE)

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

SPRING
2004

THE ECONOMICS OF OPEN SOURCE SOFTWARE
A Model for Total Cost Analysis

A Project

By

Richard Lloyd Barnes

Approved by:

_____, Committee Chair
Sylnovie Merchant, PhD

Date:_____

Student: Richard Lloyd Barnes

I certify that this student has met the requirements for format contained in the University format manual, and that this Project is suitable for shelving in the Library and credit is to be awarded for the Project.

_____ _____

Chiang Wang, PhD, Graduate Programs Office Director Date

College of Business Administration

Abstract

of

THE ECONOMICS OF OPEN SOURCE SOFTWARE

A Model for Total Cost Analysis

By

Richard Lloyd Barnes

Software licensing fees are a major cost for enterprise Information Technology (IT) departments. Faced with restricted funding, a developing economy or simply the need to generate more profit, both private and governmental organizations are looking to substitute sources of computer systems and services. This paper is a researched discussion of a "free" alternative solution, Free/Open Source Software (F/OSS), and the costs and benefits that make it a practical choice as a replacement for existing software systems. An assessment of the culture that produces, maintains and promotes this free software, which is fundamental to the evolution of a new development model, is also included. The objective of this F/OSS study, and the final portion of the paper, is the presentation of an IT cost analysis model, which can evaluate the total cost of ownership (TCO) of up to three alternative systems simultaneously. The spreadsheet-based model,

developed by the author, is a hybrid of published analytical models and accepted capital budgeting methodologies, which are examined within this study. The model presents a structured framework for acquiring cost figures and calculates ten financial metrics from the assembled data. This paper concludes with instructions for using the model and recommendations for further consideration.

_____, Committee Chair
Sylnovie Merchant, PhD

Date:_____

Table of Contents

Page

List of Figures ix

List of Tables x

Software Specifications xi

Chapter

Introduction 1

Background 2

Categories of Software Licenses 12

Free Software – The GNU and GP Licenses 14

Open Source Software – The OSD 16

Proprietary Software 19

GNU/Linux 21

Open Source Software Development 23

The Open Development Model 24

Participant Motivation 27

Open Source Business Models 32

Open Source Viability 40

Performance 42

Scalability 43

Security 45

Reliability 49
Support 50
Market Share as a Measure 53
Open Source SWOT Analysis 54
Problem Description 61
Modeling IT Capital Investments 65
Hard versus Soft Costs 69
Methods of Analysis 75
The Total Cost Model 82
How the Model Works 84
Acquiring the Data 86
Navigating through the Model 88
Entering Data into the Model 89
Viewing the Results of the Analysis 92
Preferences 100
Resetting the Model 103
Conclusion 105
Appendix A - The GNU General Public License 112
Appendix B - The Open Source Definition 118
Appendix C - Linus Introduces Linux 121
Appendix D - Examples of Free/Open Source Software 123
Appendix E - SCO Group letter to IBM 126

Appendix F - Homogenous SWOT model of Free/Open Source Software 128

Appendix G - Excel Workbook Formulas used in the Total Cost Model 129

Appendix H - Visual Basic for Applications programming examples..... 132

References 138

List of Figures

Page

Figure 1- Market Share for servers across all domains 9

Figure 2 - Reasons to join an Open Source community 33

Figure 3 - Most important benefits of Open Source community participation 34

Figure 1 - Motivational difference between paid and volunteer developers 35

Figure 2 - Linux user server software ratings by Operating System 41

Figure 6 - The relationships between the Inputs and Outputs of a financial model.... 68

Figure 7 - Concepts and Instructions worksheet 86

Figure 8 – The Total Cost Model's menu and command button design. 89

Figure 9 - Common Data worksheet 94

Figure 10 - Alternative System Costs worksheet (3 year analysis period) 96

Figure 11 - Cost Analysis worksheet 98

Figure 12 - Graphical Analysis worksheet (display reduced to 35) 100

Figure 13 - Preferences / Workbook options dialog box 101

Figure 14 - Reset the model data dialog box 104

List of Tables

Page

Table 1 - Top Server Developers: NetCraft Web Server Survey May 2004..... 10

Table 2 - Simple Comparison of “free” Software Licenses (after DiBona, 1999)..... 19

Table 3 - Examples of Proprietary Software 20

Table 4 - Attributes of the Open Source Software Lifecycle 26

Table 5 - **Cost Element Taxonomy for Open Source Software**..... 72

Table 6 - Project Cost Matrix – identifying expenses attributable to a project..... 75

Table 7 - Description of the worksheets within the Total Cost Model. 82

Table 8 - Data requirements for the metrics as calculated by the Total Cost Model .. 88

Table 9 - Input data requirements by worksheet 92

Software Specifications

The Excel spreadsheet, Cost_Model.xls, which is the object of this project should be run under the PC version of Microsoft Excel 2000 or later. It may work fine under Excel 97, but has never been tested on any version older than 2000. According to Microsoft, the **minimum requirements** for running Excel 2000 are:

Computer:	IBM Compatible PC.
Processor:	Pentium (or compatible) processor, 75 MHz or faster (Windows 95 and 98); Pentium (or compatible) processor, 233 MHz or faster (Windows 2000, Windows XP).
Operating System:	Microsoft Windows95, Windows98, Windows98 Second Edition, Windows Millennium Edition (Windows Me), Windows NT 4.0 with Service Pack 6, Microsoft Windows 2000 with Service Pack 3, Windows XP, or later.
Memory:	16MB RAM for Windows 9x, 32MB RAM for Windows NT 4.0, 128MB RAM for Microsoft Windows 2000, Windows XP.
Video:	Super VGA (800 × 600) or higher-resolution monitor.

Introduction

Oppressive licensing fees and an oligopolistic enterprise software market have led many organizations and governments to examine less expensive and restrictive alternatives to commercial or proprietary software. Such solutions have existed for years, and have been utilized with great success, notably within the infrastructure of the Internet. This paper is a discussion of a "free" alternative solution, Open Source Software (OSS), and the costs and benefits that make it a practical choice to replace existing systems. An assessment of the culture that produces, maintains, and promotes this free software, which is fundamental to the evolution of a new development model, is also included. The outcome of this study is the presentation of a cost analysis model, which can evaluate the costs of implementing up to three alternative information systems solutions simultaneously. The spreadsheet based cost model, developed by the author, is a hybrid of published analytical models and standard Information Technology (IT) cost analysis methodologies examined within this paper.

Software licensing fees are a major cost for today's corporate IT departments. Despite the ubiquity of computer systems at all levels of business, these costs have remained constant for years or, in many cases, increased. A company with a lion's share of any software market can demand excessively high prices for new products or the licensing of existing ones. Further, the cost of support and end-user education has increased, as more entry-level users are required to use complex computing systems. The use of OSS solutions can release funds, which then can be used to advance the true purpose of an IT department – supporting the computing needs of the enterprise.

Switching from a familiar computing system is not a trivial project for any IT department. It requires the input, not only from IT professionals, but from users at all levels as well as corporate management. Managers will most likely make the decision to change based on financial measures such as the Payback Period, the Return on Investment (ROI) and the Total Cost of Ownership (TCO.) This paper will point out both the money costs and the productivity costs. Though both can be expressed in financial terms, user resistance and compatibility problems have a more human side. These factors along with end-user reeducation costs, support issues and security concerns are all addressed.

This work is not meant to be a criticism or conviction of any specific software or systems provider. It is an evaluation of the current business software market-share owners, their Open Source and Free software alternatives and the true costs of each. As such, there are comparisons to products of the Microsoft Corporation, in view of the fact they own many of the enterprise and home computing software markets. When possible other vendors with a smaller share of the market are mentioned for a more balanced discussion and an overall healthier research methodology. However, in the end, Microsoft remains the benchmark and one of the more probable subjects of any IT department's software TCO evaluation.

Background

As the computing and scientific disciplines that embraced it started to flourish at university research laboratories during the 1970's, software program code sharing was seen as a normal exchange of ideas by the academics and researchers. Actually, software sharing had existed for years, and it was said that hackers[1] shared code like cooks shared

their favorite recipes. Such practices seemed right and natural to academics and researchers. Eventually these researchers graduated or moved into the private sector. By the early eighties, the community of sharing known to many of the early pioneers of computing science had become cloaked by corporate legalities and non-disclosure agreements. The sharing of source code was now a punishable offense. (Stallman[1], 2003) (Kuwabara, 2000)

Proprietary software became the norm as profits were realized from software copyrights, patents and other trade agreements. In 1984 a programmer at the Massachusetts Institute of Technology (MIT) Artificial Intelligence laboratory, Richard Stallman (RMS, as he is known throughout the Internet), quit his position and set out to continue writing free software. "Free" he meant, not as in cost, but rather the freedom he had years before to do what he wished with the software he used. (DiBona., 1999) (Stallman[1], 2003)

Stallman commenced to create a completely free operating system (OS) with all its ancillary software. His project was named the GNU[2] project. Despite some misgivings about the UNIX operating system, he chose it as a basis for the GNU project's OS since UNIX programmers had a history of code sharing. The free sharing of code would be the foundation of all GNU software. Later the Free Software Foundation (FSF) was created as the principal sponsor and legal entity for the GNU project. To this day, it is supported only by the contributions of individuals; there are no corporate sponsors. In fact, the essays of RMS read somewhat like political manifestos and speak with fear of corporate control of the computer industry. (Raymond, 2001) (Stallman[1], 2003)

Regardless of his fanaticism, Stallman has advanced the software world by being the driving force behind two popular applications, EMACS the “kitchen-sink” text editor/word processor/web browser/programming tool and the GNU C language compiler, simply known as GCC. Both of these programs have been ported to numerous operating systems and are used by a vast number of people worldwide. (DiBona, 1999) (Stallman[1], 2003)

In 1996, a computer user and self proclaimed “old hacker”, Eric Raymond, was looking for an e-mail client that would serve his needs. He found one in a free software project that had become stagnated, and soon took over leadership of its development. The code became his to direct as it evolved into the program that would serve his needs. His experience working with this development project helped him to formulate theories on the growing Free Software movement. (Raymond, 2001)

In several essays, Raymond, who has become an OSS evangelist, compared Free/Open Source development to a bazaar whereas he likened the proprietary (or closed source) software development environment to that of a cathedral. He based his conclusions on his experience in leading the OSS project that developed the *fetchmail*[3] program. His book, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary,* is a collection of these essays and has been called both a bible and an anthropological analysis of the F/OSS community. (Raymond, 2001) (DiBona, 1999)

Raymond noted some principles that are common to all successful Open Source or Free Software projects:

- Every good software program starts by satisfying a need of the author (“scratching an itch”.)
- Great programmers know what code to reuse and what to rewrite.
- Involve your user base as co-developers.
- Release code updates early and often.
- The last duty of a F/OSS project leader is to pass the management of the project off to a capable new leader.

Raymond’s influential work in this area, lead him to a collaboration with Bruce Perens, another UNIX hacker and Free Software advocate. Together they created the Open Source Initiative (OSI) and wrote the Open Source Definition (OSD) that provided, in their view, a less radical approach to Free Software then that of the Free Software Foundation. They felt the style of Richard Stallman and the FSF was seen by many as counterculture, and their confrontational tactics were not business friendly. To further legitimize their efforts, they acquired an OSS certification mark and standardized the Open Source Software name within the Open Source Definition. (Raymond, 2001) (DiBona, 1999)

Raymond readily points out that the OSI agrees with the principles of the FSF, but the OSI was created to mainstream Free Software and make it friendlier to business interests. Since the formation of the OSI, free software has become more widely accepted; and the GNU operating system, Linux, has come to threaten Microsoft Window’s market dominance. (Raymond, 1999)

However, the incident that brought the vision of Open Source software to the masses was the rather uneventful release of a new UNIX-like operating system kernel in

1991 by a young Finnish college student. In that year, Linus Torvalds released his hobby, Linux, to the Internet. Linux took off as programmers from every corner of the Internet submitted code, tested and redistributed the program to new users. Soon the kernel was a stable, reliable and viable OS kernel, and the GNU Project finally had its operating system. The enthusiasm for Open Source software grew, as there was now a completely free, unencumbered GNU OS. (DiBona, 1999)

Obviously, the work of RMS, Eric Raymond, Linus Torvalds and the other free software revolutionaries did not occur within a vacuum. During the 1980's and 90's several large software companies came to dominate and drive the commercial software market. IBM had been around for decades and their work in computing was the basis for many modern systems. Their dominance and influence was to be expected. However, two new firms emerged in the late seventies that would transform the way businesses and consumers used computers, Apple Computer and Microsoft Corporation. It was in the face of the almost oligarchic hold these companies had along with several others, such as Oracle, Sun Microsystems (Sun) and Hewlett-Packard (HP) that Open Source code had to prosper. (Anonymous,[22], 2003)

These dominant and proprietary software systems tended to lock users into not only their software with all its restrictions, but also to specific hardware and data formats. Such arrangements have the tendency to make switching from one system to another, whether it is an operating system or word processor, an expensive task. F/OSS leads to open systems and open standards, which can be easily translated across platforms. Proprietary software also binds users to the cycle of upgrade fees. The United States Federal government could save at least one billion dollars in original licensing fees by

switching to Open Source software. Additionally, upgrade or maintenance fees do not exist with F/OSS. (McCullagh, 2002) (Wysong, 2000) (Raymond, 2001)

In the past few years, major Information Technology vendors and enterprises have embraced the use of, and the concepts behind, Open Source software as an alternative to traditional commercial/proprietary software. There also appears to be a strong drive from consumers outside of the United States for “software libre”, or free software. Many municipal, state and national governments have legislated either the mandatory use of, or the study of using F/OSS alternatives where possible. (Festa, 2001)

However, cost alone is not what makes F/OSS so inviting to organizations as a substitute for proprietary or Commercial Off the Shelf (COTS) solutions. The manner in which it is developed and maintained has defined a new model for software development. With F/OSS there is no single company determining the software's feature set, release schedule, or most importantly, demanding an adequate return on its investment. The lack of centralization in F/OSS development provides a much wider base of technical expertise, a more varied and substantiated feature set and more widespread and differentiated testing platforms. In addition, the talent pool that invests time to collaborate on such projects as the Linux operating system is beyond the resources of any “closed-source” developer. Based on its projected user base, no software company could possibly bear the financial burden of the 800 programmers who volunteered their talents to produce Eric Raymond’s Open Source program, *fetchmail*. (Anonymous[7], 2000) (Raymond, 2001)

Switching from proprietary computer systems such as those merchandised by Sun, HP and IBM to generic computer hardware running GNU/Linux can save users thousands

of dollars per server. In some cases, analysts have projected long-term savings of millions of dollars. However, even research originating from the same firm, International Data Corporation (IDC)[4], differs on whether OSS has a lower TCO. One paper published in 2001 under the sponsorship of Redhat[5] concludes: "…open source software such as Linux could lower the costs associated with software…When summed up, this could mean a lower total cost of ownership." Another IDC study initiated by Microsoft and released in 2002 offers a different conclusion: "Our in depth study suggests that Microsoft Windows 2000 offers a lower total cost than a Linux solution…the cost advantages are significant: 11-22% over a five year period." Disparities arising from skewed or highly biased studies make it difficult for managers to assess alternative solutions. (Bozman, et al, 2002) (Gillen, 2001)

Reliability and Market share are frequently used both as decision variables and as arguments against implementing Open Source solutions. An article in the March 2003 issue of *eWeek* magazine contained the following comments made by the CEO of a Virginia based IT vendor and service provider:

> "I don't see government adopting Apache to run any major, substantive systems…When you are running government systems you are accountable. You want proven systems." (Taft, 2003)

Apparently, this CEO does not know Apache is the most widely used web server with over 65% of the active web sever market world wide according to the NetCraft and E-Soft Web Server Surveys. Apache has been the number one web server since 1996 when it took first place from the public domain National Center for Supercomputing Applications (NSCA) web server. It continues to take market share from all other web

servers, commercial or otherwise (See Figure 1 and Table 1.) (NetCraft, 2003) (E-Soft, Inc., 2003)

Figure 1- Market Share for servers across all domains August 1995 – May 2004
(source: http://news.netcraft.com/archives/2004/05/index.html)

The benefits of Free or Open Source may not be just about price, but this aspect does very much define its allure. However, it should be completely understood that free software is not simply about price. Internet Explorer is free, and so too, are many other programs. This is not the characterization of free software this paper means to discuss. It is, as Richard Stallman protests, *free as in Free Speech not as in free beer*. It is a matter of principle rather than price. (Stallman[1], 2003)

Once the fitness to task or reliability of an OSS solution is determined, an IT manager can be assured of a rapid distribution of fixes whether for performance, security

or new features. The development model of OSS enables the quick release of new code since patches can be released at any time. Thus, a user will not have to wait for the next release for satisfaction…satisfaction that may come with a new version and a new license fee. Further, with the ability to view and change the underlying code of an application, an organization's programmers can tailor it to their own needs and extend its life as needed. (Kenwood, 2001)

Developer	April 2004	Percent	May 2004	Percent	Change
Apache	33329879	66.99	33892817	67.05	0.06
Microsoft	10691683	21.49	10858168	21.48	-0.01
SunONE	1661229	3.34	1644412	3.25	-0.09
Zeus	763302	1.53	754014	1.49	-0.04

Table 1 - Top Server Developers: NetCraft Web Server Survey May 2004
(source: http://news.netcraft.com/archives/web_server_survey.html)

Proponents of F/OSS employ several common arguments for its superiority over Proprietary or "Closed" source software:

- **Quality or Reliability** – Because F/OSS software is developed by persons with different skill levels, using different equipment with different vesting in the success of the software, there is an examination of the code from diverse viewpoints. Each participant has their own specific interest or stake in the project that guides their involvement. This condition does not guarantee that all bugs will be eliminated, however it greatly increases the chances they will at least be exposed. The number of active participants on many OSS projects dwarfs the number that a commercial company could use and still create a profitable piece of software.

- **Speed of Fixes and Updates** – One of the tenets of OSS development is "release early, release often". Patches, fixes or enhancements generally are released as soon as they are approved by the project leader. This is counter to commercial software developers who work closely with marketing departments to decide when the next version should be released. F/OSS security patches are often released before there is a widely known problem.
- **Cost** – Generally, there is no initial cost except the cost to download the software. However, there are companies that sell F/OSS bundled with installation or general use support services or customized for specific duties. OSS has no upgrade costs, only the ongoing maintenance support costs that a typical application or system might normally have.
- **Verifiability** – Because users can examine the source code, organizations with strict security requirements can audit the software to ensure that it is actually doing only what it is supposed to do. Closed source systems are used trusting that the developer is coding in their best interests.
- **Freedom** – Open source applications tend to run on numerous computer platforms and generally adhere to open industry standards. Since most support is offered by third parties, an organization need never be restricted to one vendor. Users could switch to another support vender without changing the software. F/OSS use also affords the freedom to move from one hardware platform to another without buying new software or

investing in extensive user training. (Wysong, 2000) (Raymond, 2001) (DiBona, 1999)

Although many licenses exist which define the attributes of Open Source Software, it can generally be defined as software that has no licensing fee, is developed mainly by volunteers and whose source code is published and available for modification by an end-user. The success of such products as the Linux operating system and the Apache web sever has shown OSS to be both reliable and viable. (Peeling, et al. 2001)

Categories of Software Licenses

Some of the terms used in conjunction with OSS, such *open source* and *free* do not describe any specific license, but are used as general descriptors of a class of software with two common features; no licensing fees are charged and the program's source code is freely available to the user. The fact that no licensing fees are charged to use the application nevertheless does not prohibit any charges for distributing the software. In fact, the FSF actually promotes *distributing free software for a fee*. (Anonymous[9], 2002) (Stallman[2], 2003)

There are currently over 40 software licenses approved by the OSI, each with its own requirements and restrictions. However, they all adhere to the OSD; which is reproduced in *Appendix B*. (Anonymous[20], 2003) (DiBona 1999)

For the purpose of this study, Open Source Software (OSS), or Free/Open Source Software (F/OSS), will be considered any software that:

- Does not have a licensing fee.
- Makes available its source code and allows modifications to it.

- Can be redistributed without restriction.

Though it is at times rather dogmatic or rigid, the FSF uses clear distinctions in an area that is not always so distinct. The following is a listing of software types as published by the Free Software Foundation based on the software's license:

- **Free software** is free to use, copy, alter or sell (with or without modifications) as long as any distribution contains the full source code to the program.
- **Open Source** is usually, but not always, a more limited license than that of Free software. Some OSS has license restrictions due to the use of copyrighted libraries or those placed by owner of the copyright. Although it is free, or gratis, the numerous licenses for OSS have differing limitations and conditions.
- **Proprietary software** is not free. In contrast to Free software, you cannot use, distribute, or modify the software, except under the strict terms of the owner's license. Proprietary means it is owned by someone.
- **Commercial software** is developed and owned by a business. Most commercial software is *proprietary* and not actually sold but rather leased. You do not own the program(s); you are simply given a limited license to use the software. Source code is not generally available for commercial software.
- **Shareware** is a type of commercial software that allows a user to try a "crippled" or time limited version to ascertain the fitness to task of the program(s). After the trial period, you are obligated to purchase the

software and adhere to the terms of the seller's license. Most shareware does not make the source code available to users.

- **Public domain software** is not copyrighted, but the source code may not be available, thus it can still be "proprietary" in nature.
- **Freeware** can mean almost any "free of charge" software. It is not *Free* as in the FSF definition, as it does not usually give the user the source code or allow any modification to it. However, since there is no charge for its use, the software can typically be redistributed without any limitations. (Anonymous[6], 2002)

The minor differences of opinion concerning licensing terms and conditions among F/OSS advocates are continually an issue. Note this statement posted on the PostgreSQL[6] website:

> "People often ask why PostgreSQL is not released under the GNU General Public License. The simple answer is because we like the BSD license and do not want to change it. If you are keen to read more about this topic, then please take a look in the Archives at any of the many threads on this subject, but please don't start yet another debate on the subject!" (Anonymous[21], 2003)

Free Software – The GNU and GP Licenses

In 1984, Richard Stallman, who was working at MIT, started the GNU Project to create and develop a completely free operating system and promote free software. The name GNU (pronounced "guh-NEW") came from the hacker tradition of using recursive acronyms. The term GNU means "Gnu's Not UNIX". Today the GNU system is mainly known as Linux; however, Linux is just the core of the system. Thousands of GNU

applications make up the available system and provide nearly every type of software application. See *Appendix D* for examples of GNU software. (Anonymous[4], 2003) (Anonymous[9], 2002)

Free software, Stallman believes, represents knowledge, and should be freely shared, thus enriching all of computer science. Proprietary software companies, he feels, are actually hindering the advance of software technology by hiding behind vague intellectual property laws. This idea comes from years of experience in software development where programmers freely traded code. Part of the GNU philosophy is just that; freely available source code is necessary for the advancement of science. It also embraces the idea of reusable code and eschews reinventing the wheel. (DiBona, 1999) (Anonymous[13], 2003)

To promote the idea of Free software and the GNU project the Free Software Foundation (FSF) was formed in 1985 as a tax-exempt umbrella organization for the GNU project and the development and distribution of GNU and non-GNU (but still free) software. Numerous GNU software programs have been written or maintained by FSF employees. (Stallman[1], 2003)

The “free” of Free software is a very often misunderstood concept. In fact, the Free Software Foundation's web site actually encourages people to charge for "free software." This is because charging for free software does not make it less free, you still have the explicit license (under Gnu’s General Public License and many other licenses) to do what you want with the software (with various minor restrictions.) The FSF sees the selling of free software as a way to finance the furthering of the product while maintaining its “free” status.

The GNU General Public License (GPL), which is the basis of the FSF licensing policies, provides the end user with four essential “rights”:

- The ability to use the program for any purpose.
- Freedom to redistribute the software for free or for a fee.
- Access to the complete source code for the program and all future changes to it.
- The right to modify any part of the source or use portions of it in other programs.

The GPL gives the end user the right to distribute a modified version of the software, but it must be done without restricting or revoking any of the rights, which were originally granted. Thus, all modifications (and hopefully improvements) are returned to the community for everyone's benefit. GNU software uses a form of copyright known as the “copyleft” and, is not public domain software. The idea behind the copyleft is to maintain the freedom of the software, not to restrict its use as with a typical intellectual property patent or copyright. (Anonymous[11], 2003)

While those at the FSF say: “Free software is a matter of liberty, not price”, the OSI “does not have a position on whether ideas can be owned, whether patents are good or bad, or any of the related controversies.” (Anonymous[11], 2003)

Open Source Software – The OSD

The Open Source software movement began as a more liberal (or more conservative depending on your viewpoint) approach to software licensing. It was spawned as the result of Netscape’s plan to give out the source code for its browser

software. Netscape asked Eric Raymond to consult on the release of their code to the public because of his writings and work in the name of Free software. He and some of his colleagues knew this would be an excellent opportunity to present the virtues of Open Source software and offer a new license paradigm. (DiBona 1999)

The founders of the Open Source movement were concerned with Richard Stallman's' radical, and at times socio-political, ideas concerning software freedom. His beliefs were suited to many hackers and other counterculture types, but Raymond was especially concerned that they would not be so inviting to conservative business people. He believed a new license was desirable. (DiBona 1999) (Raymond 2001)

The Debian Linux distribution had been created to supply a completely Free GNU/Linux software collection. Most of the software they included was as free as the FSF required, but some used licenses other than the GPL. Because of this, the Debian developers wrote the *Debian Free Software Guidelines* to direct the selection of software for inclusion in their compilation. It was a document that essentially defined what was to be considered Free software and what would be non-free. In early 1998, the Debian Free Software Guidelines became the basis for the Open Source Definition. (DiBona, 1999)

Bruce Perens, the author of the Open Source Definition (OSD) calls it is a bill of rights for computer users. It defines what a software license must grant you to be classified as Open Source software. It is a description rather than a license and provides guidelines for the over 40 licenses currently approved by OSI. The Open Source Initiative was formed in 1998 as a public benefit corporation (not-for-profit) to manage the new Open Source certification mark and to promote Open Source ideals. (DiBona, 1999) (Anonymous[9], 2002)

Regardless of the specific license, OSD compliancy guarantees:

- The right to copy and distribute copies of the program.
- Access to the program's source code
- The right to make improvements to the source code

These may seem similar to the rights bestowed by the GPL as they are seeking the same ideal. However, the OSD embraces many licenses that are not palatable to the Free Software advocates. It gives more latitude to the owners of the code, but never preempts the previous three rights. They are the unalienable rights of Open Source Software. Table 2 offers a comparison of several free software licenses. (DiBona, 1999)

Proprietary Software

Essentially this class of software is “owned” by someone. Although there may be no charge for using it, as with Microsoft's Internet Explorer, it is not “free”. True, it is free of cost, but the user does not have any access to the program’s source code. What makes software truly proprietary is that you are restricted from doing anything with it except by agreement with the “owner”. The owners of proprietary software consider their code to be trade secrets, meaning that they have actually copyrighted programming algorithms. Table 3 illustrates examples of proprietary software. (Anonymous[6], 2002)

License	***Can be mixed with non-free software***	***Modifications can be taken private and not returned to you***	***Can be re-licensed by anyone***	***Contains special rights or privileges for the original copyright holder over your modifications***
GPL				
LGPL	√			
BSD	√	√		
X Window	√	√	√	
NPL	√	√		√
MPL	√	√		
Public Domain	√	√	√	

Table 2 - Simple Comparison of “free” Software Licenses (after DiBona, 1999)

The traditional method of selling COTS or proprietary software is actually the issuance of a license to use the software. The user purchases the right to use the software much as you would purchase a book. As with a book, you have a physical copy of the

book and can read it as you like, you may not take it apart and use its pieces as you wish. This is basically the same as with software; like written works, the code and the underlying techniques (algorithms) are considered intellectual properties (see *Appendix E*.) These ownership rights give the holder a direct value and a ROI for the software, which is the intent of most business enterprises. (Hecker, 2000)

Category	**Software**	**Developer / Vendor**
Operating Systems	Windows	Microsoft Corporation
	Solaris	Sun Microsystems
	OS/2	IBM
	Palm OS	PalmSource Corporation
Productivity	Microsoft Office	Microsoft Corporation
	Lotus SmartSuite	Lotus / IBM
Network	Lotus Notes	Lotus / IBM
	Netware	Novell Corporation
Database	Oracle	Oracle Corporation
	SQL Server	Microsoft Corporation
Internet	Internet Explorer	Microsoft Corporation
	WS_FTP	Ipswitch Corporation

Table 3 - Examples of Proprietary Software

Commercial software can be open source and yet not proprietary, so it is important to maintain a differentiation at this point. For Example, the MySQL database software is open source and owned by MySQL, AB. It can be used without any licensing fee under the GPL or under a commercial license that allows for less restricted use by the customer. In other words, you can use the database software free under the GPL as long as you do not violate its terms. However, distributing the MySQL database with a non-free application violates the GPL and you would need to license the database software under the commercial license. In the case of MySQL, this is an important distinction as

databases are regularly used by front-end applications and other programs that do not qualify under the GPL. (Anonymous[17], 2003)

GNU/Linux

As previously mentioned the GNU Project was formed to produce a complete, and absolutely free, operating system. This not only meant the applications to run under the system but also the OS kernel. The proposed GNU system was named the HURD (as in a herd of gnus) and was to consist of servers or processes that ran on top of the Mach microkernel. This kernel, which was developed at Carnegie-Mellon University, has become the foundation for several operating systems including Apple Computer's new OS/X operating system for Macintosh computers. (DiBona, 1999) (Norton and Story, 2000)

However, by 1991, the HURD was still not ready for use and was thought to be years away from a stable release. The GNU project, on the other hand, already had many applications which, when bundled with an OS, would make a completely free system. The GNU library of applications included a C language compiler, which could be used to create or port[7] additional software to run under a GNU OS. (DiBona, 1999)

A solution was introduced on August 25, 1991. On that date a Finnish undergraduate Computer Science student at the University of Helsinki named Linus Torvalds posted an interesting message on the Minix[8] user newsgroup:

> “Hello everybody out there using minix -
> I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready.”

See *Appendix C* for the complete text of several of Linus' early newsgroup posts. (Hasan, 2002)

Linus had been using Minix, which is a UNIX-clone OS developed as a teaching tool and used by many students as a UNIX alternative. However, it was not free. He chose the name Linux for his new OS as a combination of Linus and Minix. However, by the time Linus posted his first historic message in 1991, Linux contained no code borrowed from Minix; it was new and unique. In the Fall of that same year, Linus, with the help of many volunteers from the Internet, had created a kernel that could run BASH (the Bourne Again shell) and the GNU C language compiler, GCC. This made it usable and provided the foundation for a UNIX compatible OS that was truly free and open source. Linux was adopted as the GNU operating kernel and officially became known as GNU/Linux. (DiBona, 1999)

Today Linux is still just an OS kernel. What many people think of as “Linux” is a combination of this kernel and the GNU applications that run under and provide services through it. Still more people think of Linux as the set of applications bundled together and merchandised by companies such as RedHat, SuSE[9], MandrakeSoft[10], and many others. While these compilations do contain the Linux operating system and GNU software, they are not Linux, but rather Linux distributions (or distros.) Although most companies sell their distros, they customarily follow the tradition of Linus' original

release; always free for the download. However, since many distributions actually contain some commercial, proprietary and other non-free software the download versions of these distributions are not as complete as the retail versions. (Anonymous[16], 2003)

There are distros that provide complete web servers, development platforms or basic desktop systems. Linux runs handheld computers, gaming consoles and is the OS used by the TiVo television recording/replay devices. Linux powered computers helped create the graphics used by Pixar Studios for its movies *Monsters, Inc*. and *Toy Story*; and brought the Tolkien classic, *The Lord of the Rings*, to life on film. Linux has gained the attention and sponsorship of such companies as IBM, HP, Sun and Oracle. It is big business and provides profits to many companies, but it is still free. This is the paradox of free software. (Greene, 2003) (Anonymous[16], 2003)

Open Source Software Development

> "If builders built houses the way programmers built programs, the first woodpecker to come along would destroy civilization." - Gerald P. Weinberg (Anonymous[19], 2003)

Cooper Stevenson suggests that Open Source development or, the Open development model, is similar to the standardization of the IBM Personal Computers (PCs) of the early 1980's. The publishing of PC standards by IBM allowed for fierce competition, which drove innovations while the cost to consumers decreased dramatically. Such competition, he reasoned, forces vendors to continually improve their product or lose any market share they owned. This notion alone, outside of any true

economic effect or profit, would be a substantial benefit to the Industry as a whole. (Stevenson, 2003)

The Open Development Model

Open Source development is simply another software development methodology, not a new technology. There are two major concepts which make OSS development a distinctly new software methodology. Both are intertwined to a point that they tend to drive each other. The first is the type of license an OSS project uses, and the other is the structure of the project's development. The specific license will dictate how the project is managed, to what degree volunteer help is used and lastly how and when the project's product is released and distributed. (DiBona, 1999) (Stoltz, 1999)

Open Source development is a distributed model of development, though the true framework depends on the licensing of the software. As stated earlier, there are over 40 OSS licenses compliant with the OSD. While each offers users basic rights similar to those given under the GPL, the control of the development or the future of the code can be different. Each OSS project has an owner, who either actually owns a copyright against the project, or is simply a leader who manages the software's evolution. These individuals or entities guide the project by making decisions about the development of the code and the direction of its evolution. Some owners that are commercial entities have staff programmers working on OSS projects alongside the customary volunteers. (DiBona, 1999) (Raymond, 2001)

Many have the impression that Open Source development is merely the creation of free software by volunteers sharing code and ideas via the Internet. While this concept is somewhat true, the process has evolved and has become more complex. Although

every project is customized to achieve specific goals, there are characteristics common to most F/OSS projects:

- Peer review
- Prompt and continual feedback
- Skilled and motivated developers
- User involvement in all aspects of the project
- Rapid release of new or improved code versions
- Parallel development and debugging (developers working separately on the same code) (Schweik, 2003) (Peeling, 2001)

As shown in Table 4, the lifecycle of a F/OSS project involves many more workgroup organizational and policy definition tasks than the average in-house or commercial project. Proprietary "cathedral" type projects do need organization, but they have the advantage of previously created policies, which can easily be reused. One of the major decisions affecting the future policy decisions of a project is the selection of the license that will govern the software. The ultimate demise of a F/OSS project, as shown in Table 4, is when there is no longer any interested party to take over the project lead. However, GPL projects may be revived at a later date if someone wishes to take the helm. (Schweik, 2003)

Raymond uses the metaphor of a *cathedral* and a *bazaar* to point out the differences between F/OSS and closed source (or proprietary) software development. The cathedral is a model of centralized planning and activities with highly structured schedules, guidelines and procedures. In Cathedral development, the programmers produce their code in the quiet solitude of secrecy. The software development follows in

a cohesive manner, a set of prioritized guidelines for features and functions and all work is closely coordinated to achieve specific goals. Closed source, proprietary software is the product of cathedral development. (Lancashire, 2001) (Raymond, 2001)

OSS Lifecycle Attributes

Stage 1: Initiation
• Individual initiator or small group (e.g., 2-4 people)
• Highly skilled development team
• Modular design
• The kernel with some promise
• Virtual team may grow in size as progress is made (e.g., "trusted lieutenants")
Stage 2: Going "Open"
• Open source license
• Credibility of the initial kernel and participant base
• Communication systems
• Version control systems
• Initial governance structures and institutional design established
  o Operational rules
  o Collective-choice rules
  o Constitutional-choice rules
Stage 3: Project Growth, Stability or Decline
• Recruitment strategies
• Size of community with a "personal itch" for the product
• Project growth:
  o "OS network" — regional coordinators, communities of users expands
  o Real success — expands to "OS Enterprise"
  ▪ Large network of developers/users
  ▪ Core staff
  ▪ Magazines, sponsors, organizational infrastructure
  ▪ e.g., Linux, Apache, PHP
• Project stability:
  o Participation remains stable over time
• Project decline/death:
  o Loss of participant interest

Table 4 - Attributes of the Open Source Software Lifecycle
(Source: Schweik, 2003)

The bazaar, on the other hand, is loud and noisy with everyone trying to get attention at the same time. There is no apparent hierarchy and the participants come and

go as they please. Anyone can participate. It appears to be unstructured, chaotic and on no one's time schedule. Because it is so chaotic not everyone is heard or even recognized, however it does act as a community, much like a neighborhood marketplace. Bazaar style development demands that updates or patches are applied immediately and released quickly in order to find more flaws or add new features. (Kuwabara, 2000) (Lancashire, 2001)

The bazaar concept suggests that all bugs are shallow and can be exposed quickly due to the many eyes that are set on the code. In the Open Source development model the individual who originally finds a bug may not be the same person who understands it and ultimately fixes it. Discovering it is the bigger challenge, although in Open development they both occur quickly. “Given enough eyeballs, all bugs are shallow” has been called Linus’ law. Proof that it is a valid precept is the Linux kernel itself. With its complexity and such a large base of developers, Linux is relatively bug free. (Raymond, 2001)

Brook’s Law, suggests that the complexity and communication costs of a development project increase with the square of the number of developers, and has been considered by many in the software industry to be gospel. Yet if this were completely true large and complex projects, like the development of Linux, would be impossible. Therefore, with the bazaar style of development it seems that as long as it can be managed by the leader or the core group of developers the more participants the better. (Bezroukov, 1999) (Anonymous[19], 2003)

Participant Motivation

A major criticism of F/OSS development involves the motivation of the project participants. This concern along with criticism of the skill or experience of contributors

leads many arguments in favor of traditional development models. However, the F/OSS movement attracts some of the brightest minds in Computer Science. Very often, these hackers have day jobs and still find time to participate in the drudgery of programming, and to act as advocates for the cause. Studies have been undertaken to determine the experience or quality of participants. The quantitative results of one such research study suggest the typical F/OSS project member is 30 years old and has over 11 years of programming experience. (Wheeler, 2003)

Another significant argument in opposition to F/OSS is that no one is obligated to a project. At any time, it is argued, a project leader or any of the core developers could leave. Moreover, if that person is a strong personality their departure could stall, or worse, end the active development of the project. However, there appears to be a strong peer acceptance or ego drive among F/OSS coders. It seems more plausible that once involved in any project as a core developer or owner, the more committed that individual becomes. In such a case, the thought of leaving the project one has fostered for so long might trigger a fear that development may falter, further binding them to it. (Bezroukov, 1999)

Figures 2, 3 and 4 display the results of surveys with OSS project participants in the areas of personal and job related motivations and the benefits of their participation in the projects. According to these studies, the primary incentive to join a project and the primary benefit expected both relate to increased development skills and knowledge of the subject (Figures 2 & 3.) However, the motivations differed based on whether a participant was paid or not. Volunteers were more inclined to work on a project if it provided intellectual stimulation or improved their skill set. Paid participants were

motivated by the work related functionality, or simply, the ability to perform their job better (Figure 4.) (Kim, 2003)

Participants in a F/OSS project must have trust in the leader or core group of the project. This is important as not every enhancement or offering from the contributors will be used in a sanctioned release. This confidence in the leader's judgment makes peer recognition more valuable as it comes from a respected source. Trust also prevents persons from becoming disgruntled and taking the project in another direction. This occurrence, known as "forking", is a splitting of a program's development into diverging paths, usually resulting in two separate programs. It is considered one of the more hazardous problems in F/OSS development and has been called intellectual mutiny. (Nuvolari, 2001) (Schweik, 2003)

Motivational surveys of the of developers who participate in F/OSS projects seem to make it clear that altruism or a need to share the wealth is not a primary incentive. This is not to say their choice of project has nothing to do with charity, but there seem to be other inducements:

- A majority of hackers who work on F/OSS projects do so to enhance their jobs as system administrators, webmasters, or other IT professionals.
- Many companies have assigned in-house programmers to work on F/OSS projects as a part of there business strategy. Sun employees, for instance, work on the OpenOffice.org code base, which is used in Sun's StarOffice commercial product.
- Student and professional academics write a great deal of F/OSS. Much of it is used in research and has a narrow use, but some is contributed as

F/OSS. Mosaic, the forerunner of the Netscape web browser, was written over the Christmas break by University of Illinois student Marc Andreessen. Participating in a F/OSS project offers Computer Science students professional experience in software development.

- Another motivation, and the most important as Eric Raymond sees it, is joining or initiating a project because you need or want the application. All Linus Tovalds wanted was a free UNIX-like operating system he could use at home. (Miller, 2003)

Open Source project participation has been shifting from mostly a volunteer venture to include more commercially supported paid membership. Volunteerism is still the mainstay of Open Development, but more enterprises are seeing the benefit of working to improve the software on which they depend. The myth of the isolated hacker writing code for F/OSS projects has proven to be just that. The majority of people who work on these projects do so to enhance their jobs or improve their skills (Figures 2 and 3.) Many Linux programmers who contribute code work for companies like IBM, HP, Intel and Advanced Microdevices (AMD.) This investment in F/OSS projects adds value to the sponsors own products and their use of Linux. The Apache web server, for example, has gained from the expertise of the many web administrators who benefit from the stable, reliable and low-cost system. In both of these cases, the programmers also benefit by acquiring an in-depth knowledge of the software, which in turn is a competitive advantage to their employers (Figure 4.) (Wheeler, 2003) (Miller, 2003)

Companies that “Open Source” their software should notice an increased speed with which their code evolves. When Netscape released its source code to the public, it

was immediately showered with fixes and enhancements from the Internet. This added base of developers comes at a very small cost. Volunteers in a F/OSS project have their own computing equipment, software and network connection. There is literally no direct cost for these additional employees. Moreover, the pool of talent may include hackers so experienced or educated that they could not or would not be added to the average development project. This lowered overhead allows smaller organizations to enter a market to compete against a much larger company with a dominant product. (DiBona, 1999) (Anonymous[14], 2003) (Anonymous[19], 2003) (Lerner, 2000)

Linus Torvalds has written that Linux has succeeded because it is based on good design and a good development model. From his first messages (*Appendix D*) it can be seen he is professing the Open Development model. (DiBona,1999)

Open Source Business Models

> Consumption is the sole end and purpose of all production; and the interest of the producer ought to be attended to, only so far as it may be necessary for promoting that of the consumer.
>
> The Wealth of Nations (Anonymous[25], 2003)

It is a basic tenet of economics that all things being equal consumers will buy the product with the lowest cost. One of the problems with this idea as it applies to computer software is that considering all software equal certainly relegates it to the status of a commodity. Since feature sets and implementations do vary it is hard to see the truth of this idea, yet software is becoming commoditized. Under these circumstances, could anyone suggest the consumer pay a higher price because it is best for market owners such as Microsoft or Oracle? In a capitalistic system the laws of economics will prevail and, developers that cannot compete will have either to lower their prices, provide greater value or leave the market. (Prasad, 2001)

Money saved by software consumers from lowered licensing fees will be spent elsewhere, so the lack of a price for F/OSS does not spell the end of capitalism. In fact, the funds may be used to purchase other IT products, such as hardware, on which to run the software. Such a simple change in spending behavior illustrates one strategy that businesses can utilize to make money with F/OSS. (Prasad, 2001)

Figure 2 - Reasons to join an Open Source community (Source: Kim, 2003)

Figure 3 - Most important benefits of Open Source community participation (Source: Kim, 2003)

Figure 1 - Motivational difference between paid and volunteer developers (Source: Kim, 2003)

Much of the income from the sale of software to a large enterprise is based on the service and support fees that can be integrated into a complete solution contract. Many organizations prefer to buy software as a complete solution rather than purchasing mere licenses. This is not to say that every product requires ongoing support, in fact many COTS package can be licensed and loosed upon employees without any issues. However, larger, more complex systems such as servers and database applications require sophisticated installation, tuning and maintenance services. (Anonymous[9], 2002)

Enterprise buyers look to ISVs (independent software vendors) and VARs (value added resellers) to provide this additional support. These channel partners add an indirect value to the software desired by corporate customers. The consideration of a software solution by any enterprise will often hinge on the strength of this added value. (Anonymous[9], 2002)

The OSD virtually prohibits any effort to capture the direct sale value from any covered software. Though it is still possible, it is not as easy to do. However, there are five principle and proven ways for commercial enterprises to extract an indirect value from F/OSS. Each of these business models has worked, through both the IT boom of the nineties and the lean markets of today. (Raymond, 2001) (Anonymous[19], 2003)

The five business models are:

1. **Support Sellers** – In this model, a business relies on the aftermarket needs of its customers. The F/OSS is given away, but the distribution media is sold along with service and support. Value is added to the software by commercial name backing (branding), end-user support and consulting services. This model is known to the F/OSS community as the “Give Away the Recipe, Open a

Restaurant" approach. Both RedHat and SuSE Linux distributors have used this model successfully for several years.

2. **Loss Leader / Market Positioner** – This model dictates that you give away the F/OSS (possibly at a loss) in an effort to position your own commercial software within a given market. This strategy was used by Netscape to ensure that Microsoft could not fully monopolize the web browser market. Sun uses the same tactic with its commercial version of the OSS office suite OpenOffice.org. OpenOffice.org is a fully OSD compliant office suite under its own license and is a substitute for the market owner, Microsoft Office. The Sun version, StarOffice, includes several proprietary modules that enhance the office suite (like the grammar and spellcheckers, and a commercial database application) and is sold as a COTS product. Both applications are derived from the same code base.
3. **Widget Frosting** – Using this approach, a company gives away F/OSS as a way to add value to its own product, usually hardware. By leveraging the F/OSS development community the company saves money on software development and acquires better support by having users determine what drivers and other accessories are desirable for their product. For example, a company that sells servers might sell a web server preinstalled with Apache or an e-mail server using Sendmail. These applications are included at a minimal or no cost, and could easily save the end-user hours of work. The simple bundling of the software with their product adds value to the hardware. The F/OSS community also wins with this model by acquiring more hardware support for their software, something that once was rare.

4. **Accessorizing** – The easiest to understand strategy is the one in which a business sells F/OSS companion products such as hardware, computer systems, F/OSS CD compilations and, of course, books. Popular F/OSS accessories also include the obligatory coffee mugs, T-Shirts and other hacker trinkets. The most successful and widely known user of this model is O'Reilly Associates[11]. They have extended their product line to cover not only printed F/OSS materials, but also developer conferences, online library subscriptions and other educational venues. (Anonymous[19], 2003)
5. **Free the Future, Sale the Present** – A vendor using this model releases the source code of a program (along with precompiled binaries) to the public as would any OSD compliant developer. However, the license, which permits redistribution and customization of the source code, prohibits any commercial use without a license fee. The license further guarantees that the software will become truly free at a later date, or should the original owner of the license go out of business. The use of the term original owner is important since this prevents the inheritance and thus the continued closure of the program. This model essentially guarantees the software will become open sourced some time in the future, and this ensures its continued existence.

Companies who use F/OSS to market their own products can expect benefits that will put them in a competitive advantage against businesses who remain traditional IT vendors. The main benefit and, probably the most important of all, is intimacy with the customer. Being close to the end-user provides the advantage of immediate feedback from the market. The F/OSS community is not a trivial group. Requests for new

hardware or software support or for new technologies can rapidly be incorporated into the project, as customers are often development team members. (Anonymous[19], 2003)

Another often-overlooked benefit is the broader market achieved through the porting of an application to different system platforms. It is common in the F/OSS development community for projects to have ports to various operating systems submitted by team members. Some of these versions have such a small user base that having in-house developers do the porting would be a losing financial proposition. However, the porting from one OS to another, no matter how small a market, exposes the software to more customers. (Anonymous[19], 2003) (Raymond, 2001)

A company planning to enter a software market that already has a dominant proprietary application has a much better chance of succeeding as a F/OSS product. For example, the reuse of existing code by a new F/OSS product dramatically reduces the time to market of that new product. This and the lower cost of the development team, produce an extreme competitive advantage for the newcomer. Competing as another proprietary product would increase the costs of entering the market and thus put a greater burden on a new product with unknown capabilities. Such new competition will help the software industry as a whole produce better and less expensive software. (Daffara, 2000) (Stevenson, 2003)

Finally, F/OSS business models do not suffer from the same financial problems as the proprietary models. As mentioned previously F/OSS can easily muster numbers of programmers and testers that could never be rationalized by any commercial software company. To the disbelief of many, these large teams of developers can develop systems of any scale. For example, the source for RedHat Linux version 7.1 included over 30

million lines of code. This number is estimated to represent 8,000 person years or an investment of over one billion dollars. The resources needed to keep a F/OSS project going is never limited to the “bottom-line”. There is no marketing department scheduling release dates based on the corporation’s need for income at given points in time. (Wheeler, 2003)

Open Source Viability

In today’s mature software market, traditional life-cycle cost models cannot be reliably used for pre-purchase decision-making. Indirect costs, which are harder to measure, are a more critical component of software selection. Establishing these costs is likely to involve a consideration of such performance values as scalability, reliability, manageability, and other operational and functional variables. Performance variables used in evaluating systems should always be corroborated and obtained from unbiased sources. (Kenwood, 2001) (Wheeler, 2003)

With F/OSS, end-users are not at the mercy of a developer’s “vision” which might lead them away from their own corporate needs. The threat of a vendor leaving the market, abandoning a product or ending support for a product is virtually eliminated with F/OSS. The independence from the vendor of the software might be the greatest argument for the viability of F/OSS. (Anonymous[28], 2003)

An IDC survey of current Linux users from March 2000 compared it to Windows NT and UNIX (considered for the study as a single OS.) The results show Linux rated higher than Windows NT within the areas of performance, availability, quality, security, management and scalability. Windows NT was rated higher where the reputation of

Microsoft and its extensive user base were major factors namely, branding, reputation, application choice, and ease of use. Although UNIX received higher scores than Linux in several categories, the fact that the Open Source OS beat a popular proprietary competitor speaks to its own growing reputation. The data collected is plotted on the graph in Figure 5. (Kenwood, 2001)

Figure 2 - Linux user server software ratings by Operating System (US respondents)
(Source: Kenwood, 2001)

One of the strongest cases for the practicality, utility and viability of F/OSS comes from Microsoft's own internal documents. In October 1998, a confidential and internal memorandum was leaked that noted the threat of OSS to Microsoft's own software. It includes the following opinion, as written by Microsoft employee Vinod Valloppillil:

> "OSS poses a direct, short-term revenue and platform threat to Microsoft... Additionally, the intrinsic parallelism and free idea exchange in OSS has benefits

that are not replicable with our current licensing model and therefore present a long term developer mindshare threat." (Stoltz, 1999)

The document containing this quote became known as the *Halloween Document* due to the date of its release. Other internal Microsoft documents subsequently leaked over the last few years carry on a discussion of the same subject. They continue to state the threat of and superiority of F/OSS. (Nuvolari, 2001) (Anonymous[24], 2003)

Performance

The performance of computer software means its ability to use a system's resources in an efficient manner while performing its programmed tasks. These resources include the processor, memory and disk sub systems. Performance of this type is easy to measure and statistically evaluate. Efficient use of system resources means little, however, if the software is not performing its particular tasks punctually, accurately and economically. Thus, good performance can be judged a blend of both good software design and the implementation of that design. (Kenwood, 2001)

As discussed earlier, many project contributors are commercial users who have a stake in the performance and reliability of the software. It is in their best interest to produce efficient processes and algorithms, which in the end provide a well performing system. It cannot realistically be argued that they would be so careless as to spend time producing an application that runs less efficiently than COTS. (Raymond, 2001) (Miller; 2003)

Not only do the participants of a F/OSS project desire a better performing application; end-users also want compliance to standards, efficiency and reliability. To this end, the Linux project has continually received enhancements from paid

programmers working for companies such as IBM, HP, Computer Associates, Intel and AMD. Both AMD and Intel, for example, contributed to ensure the quality of 64 bit and multiprocessor central processing unit (CPU) support services for Linux, which ensures the ability to use their products on Linux based computers. This type of cooperative involvement has become a commercial necessity for many firms. (Miller; 2003)

A further indicator of the performance superiority of F/OSS over its proprietary counterparts, though not scientific in the least, is market share. In several major software categories, F/OSS either dominates or owns a significant and growing share. Although market share and end-user participation in F/OSS development cannot guarantee a high level of performance, it can be construed by taking these facts together that some superior performance qualities do exist. (Wheeler, 2003)

Scalability

The scalability of software, and particularly of an operating system, refers to its adaptability to growing demands. This term is also used to refer to the ability of systems (or software) to be used on multiple platforms of various sizes, or on systems that use multiple processors. Regardless of the usage, the basic concept of software scaling is its adaptability to growth. (Wheeler, 2003)

The most commonly discussed applications with regards to scalability are database management systems (DBMS) and operating systems. GNU applications, because of their design, can and have been ported to systems of all sizes from wristwatches to supercomputers. The advantage of F/OSS is that the source code is available to any programmer, scientist or technician who wants to optimize it for their

respective system. With a proprietary application users must wait for a system port to become economically feasible, and then must wait through the development cycle as the system is adapted. Some systems are so unique that no software vendor would be willing to expend the resources necessary to produce a reliable port. (Wheeler, 2003) (Anonymous[13], 2003)

A recent benchmark test by *eWEEK Labs/PC Labs* rated the Open Source DBMS, MySQL, a close second to Oracle corporation's Oracle 9i. The rating was given with the mention that "Oracle9i and MySQL had the best performance and scalability…" This successful outcome came while facing such competition as Microsoft's SQL Server 2000, IBM's DB2 and Sybase's Adaptive Server Enterprise. The testers also noted that only Oracle 9i and MySQL were able to run the 8-hour stability test without any issues. The other DBMS systems required workarounds to complete the same test. All of MySQL's competitors in this study have well known credentials and are not marginal products; they also come with costly licensing fees. (Wheeler, 2003) (Dyck, 2002)

Because the source code is freely available F/OSS can be tailored to most any specific need or system. F/OSS projects use open standards and programming languages that are highly portable. It is within the reach of virtually any organization to have an application customized from F/OSS. Examples of this flexibility can be seen in the large number of disparate systems that use F/OSS. (Anonymous[13], 2003) (Raymond, 2001) GNU/Linux is used on such diverse platforms as:

- Personal Digital Assistants (PDAs)
- Minicomputers
- Mainframes

- Massive Computing Clusters (over 100 CPUs)
- Supercomputer implementations
- Cellular telephones
- Television recorders and playback appliances
- Portable MP3 players
- Game consoles
- Network Appliances
- and, even a wrist watch created by IBM (Wheeler, 2003)

Many vendor initiated comparisons have been made between Linux and Microsoft's Windows products. In most of these cases, the resulting statistics favor the sponsor of the research. However, one undeniable advantage of the Linux OS is its ability to run on older hardware. Linux can run on i486 (and older) PCs, which makes it especially economical as hardware lifetimes can be drastically lengthened. Such backward compatibility even surpasses all variants of UNIX systems. This example of scalability points to the flexibility of the system, which exists by design. (Peeling, 2001)

Linux is not the only highly scalable free operating system. NetBSD[12] is another free UNIX-like OS that has been ported to run on over 50 platforms. When it comes to F/OSS scaling, the main obstacle is the motivation and creativity of the user. (Wheeler, 2003) (Anonymous[13], 2003)

Security

The assumption that software whose source code remains hidden from the public is more secure than OSS is provably false. This idea most probably stems from the belief

that if you lock something up it is less likely to be stolen. Closed code, some suggest, hampers intruders and limits the extent of a security breach or the resulting damage. The truth is, a malicious hacker does not need access to the actual source code to attack a system. This is continually proven as Microsoft's proprietary systems are regularly and almost routinely exploited. A hacker can decompile a proprietary program to reveal a close facsimile of the source which can then be examined; or they could simply run tests and observe the behavior of the application to guess how the underlying algorithms and processes work. (Raymond, 2001) (Whitlock, 2001) (Wheeler, 2003)

Source code that is closed to everyone but the owner lacks a vital component of software security – verifiability. With closed source software, security flaws or potential breaches are usually made known only after an incident. In addition, only the owner can fix security issues and customers must take their word that the problem was fixed, or worse did not create another vulnerability. (Raymond, 2001) (Whitlock, 2001)

Typically, by the time a security problem is found within a proprietary application, it has been breached at least once. In fact, exploitation of the vulnerability is most often the method of discovery. By the time the owner or developer is notified of the problem it can be assumed that it has been exploited more than once, and that this will continue until a fix is released. Unfortunately, continued breaches can almost be assured by the publication of a security warning. The period between the discovery of a problem and the release of a fix is one concern; another is the often substantial time between the release of a fix and its application by end-users. Some users may never acquire the patch, or apply it. (Viega, 1999)

In 2001 after the Code Red worm cost enterprises over one billion dollars, the Gartner Group[13], issued a recommendation that enterprises using Microsoft's Internet Information Server (IIS) switch to Apache or iPlanet servers. The fact that IIS systems were attacked 1400 times more frequently in 2001 than were Apache servers may have prompted this warning. Security flaws were discovered so frequently with IIS that patches were issued weekly. The security of the Apache server lies in its use of a fundamental rule of server administration, that of "least privileges". Rarely is one group or person given complete administration (admin) rights over an Apache system, and server processes always run with the least possible privileges. This protocol is prevalent throughout most of F/OSS where it applies, and is one of the reasons very few viruses thrive on UNIX and Linux systems. (Wheeler, 2003)

The matter of software viruses, trojans and other malicious programs has become a major security concern of IT managers. Anti-virus software is a billion dollar annual market, and losses from virus related incidents cost millions annually. There are over 60,000 viruses known to infect Windows machines. Less than 100 affect Macintosh, commercial UNIX variants and GNU/Linux systems combined. The prevalence of viruses on Windows may simply be because there are more of these systems in use, which allows for an easier spread of infection. On the other hand, it could be the design of the Windows products that allows untrusted scripts, macros or system processes to execute with more system privileges then required However, many argue that the authors of viruses have a grudge against Microsoft and are obsessed with ending its tyrannical reign, but this theory does not explain why Windows systems are so easily breached and damaged. The lack of effective protection of system files and folders, and the application

of root[14] (or administrator) privileges are probably the most evident flaws. These are common security precautions in Unixes and Linux. Their absence in the Windows OS amounts to a flaw in its basic design which suggests poor security planning. (Wheeler, 2003) (Peeling, 2001)

Another source of security problems is the intentional work of the developers themselves. Software companies have been known to insert "backdoors" into their products which allow users with the correct password or other credentials to gain access to the system at the root or administrator level. These access points can be used by legitimate personnel in the performance of their duties, but could also be used by anyone with the right knowledge or skill. In one such instance, Borland International had a backdoor in their *InterBase* product for over 6 years. Since no outsiders could review the code, it was untouched until Borland released *InterBase* as an Open Source project. The problem was discovered a few months later and fixed within a month. Microsoft had a similar backdoor placed in its FrontPage Web Server which remained hidden for over 4 years. Closed code hid these vulnerabilities from users. The possibility of such a dangerous piece of code existing in F/OSS is unthinkable, since it would most likely be discovered before the code was assimilated into a release version by the core group or leader. Additionally, inserting such detrimental code into a F/OSS project could ruin the reputation of that participant. Considering ego and peer acceptance are major factors in project participation, such issues seem unlikely. (Whitlock, 2001)

Other developer issues exist, consider the simple transmission of confidential data by an application. Flawed protocols or faulty transport agents could expose data at any phase of transmission and allow persons other than the specified receiver to access the

information. Closed code does not allow a user to determine if their transmission includes only the data they wish to send, or if the algorithms employed are defective in any way. Any breach, developer instituted or otherwise, is extremely significant when it applies to governmental use of software systems with their inherent confidentiality issues. (Festa, 2001)

Good programmers know that certain programming library calls are inherently insecure and should always be avoided in specific instances. The peer review method ensures such problems can be seen by the more advanced programmers and brought to the attention of the less experienced ones. Thus, problems are fixed and young hackers are trained to code more securely. The peer review process of F/OSS development promotes efficient, error free and secure code simply by the threat that others might discover your mistake. Code submissions are more carefully made under this threat of exposure. The "many eyeballs" concept ensures more varied scrutiny of the source code than can be found in a commercial project. (Viega, 1999) (Raymond, 2001) (Whitlock, 2001)

Reliability

As with some of the other software characteristics previously discussed, the term reliability can be interpreted to define slightly different software qualities. Both robust and dependable are synonyms for reliable, but this concept also includes code which is secure, free of defects and suitable to its designated task. (Anonymous[13], 2003)

When competing against proprietary software in reliability studies, F/OSS has shown considerable benefits in many circumstances. The results of a 2000 Swiss study of websites showed that Microsoft's IIS had twice the downtime of Apache served sites.

The research covering a 3-month period studied over 100 popular websites. Extrapolation of this data over a longer period shows a greater divide. Another study compared two Linux distributions with Windows NT Server 4.0 over a 10-month period, running identical hardware with identical processes and services running. During the test period, the NT server crashed an average of every six weeks, and required about 30 minutes of work to return it to service after each crash. However, neither Linux server ever went down during the same period. This lead the researchers to the conclusion that Linux is more stable than Windows NT, at least within the environment of their test. (Wheeler, 2003)

Many such studies and surveys can be called to testify to the reliability, stability or robust nature of F/OSS, but the best evidence comes from the Internet itself. The infrastructure of the Internet is heavily dependent on F/OSS. Programs like Linux, MySQL, Apache, SendMail and bind, and scripting languages like Perl and PHP handle a major share of Internet traffic. The explosive growth of the Internet in the last 10 years speaks to the immense reliability, scalability and flexibility of the F/OSS used as its core system software. (Anonymous[19], 2003)

Support

Software support is the assistance given to users that enables them to utilize the software to perform “as advertised”. Technical assistance is often needed to install, maintain or operate software, and is usually offered as part of the cost of the product’s license. Such support is traditionally accessed through telephone hotlines, e-mail, newsgroups, online user forums and, in certain instances, direct on-site service. Some

proprietary software companies employ third party vendors to provide the technical support for their products. Other firms, unrelated to the developer, often provide separate support access for end-users and IT professionals, as well. (Anonymous[9], 2002) (Wheeler, 2003)

Opponents of F/OSS claim that the developers or project owners lack the ability to support their applications sufficiently. The belief that F/OSS support is inferior to that offered by proprietary software companies infers that these companies offer good support. However, many proprietary software companies offer only initial installation support with the purchase of a license. Typically, to continue receiving support after that brief initial period an end-user must purchase a support contract. Paid contract support is also an option with most many F/OSS products. (Bezroukov, 1999) (Anonymous[13], 2003)

F/OSS offers two main types of support, paid assistance and free online user community venues such as newsgroups. Each kind provides the same level of support as would be expected of a proprietary vendor. However, since support staffed by volunteers may not always provide timely or even adequate answers, purchasing F/OSS from a vendor that offers support is a better alternative for those requiring such services. Providing technical support to users is one of the more successful business opportunities that F/OSS offers entrepreneurs, and it can be reasonably assumed that an independent service and support vendor would provide an adequate response to its customers needs, since its revenue depends on continued patronage. (Wheeler, 2003) (Anonymous[9], 2002)

End-user support has a lifecycle based on economics, and companies eventually end support for products that are no longer revenue producers. F/OSS support vendors,

however, can offer support as long as there is a market for their services. Customer demand drives the continued support of F/OSS, not corporate revenue streams. (Kenwood, 2001)

Another F/OSS support fallacy concerns project participant's willingness to spend time documenting the software and writing user manuals. Some suggest that this drudgery is not the work for which hackers join a project. Lakhani and von Hippel, however, found that F/OSS participants perform the mundane and less glamorous tasks such as technical documentation writing and the development of help systems and Graphical User Interfaces (GUI) for the same reason programmers write code. This motivation extends to providing support through active participation in newsgroups and other online user to user assistance venues. (Bonaccorsi, 2002)

In 1997 InfoWorld awarded its Best Technical Support Award to the Linux user community as a whole. The award web page gave one of the best cases for the support offered by the F/OSS community. The commentary offered a satirical analogy of a customer calling Microsoft technical support, and ironically points out the differences between the support offered by a vendor of proprietary software and the support that can be had through F/OSS channels:

> "…imagine, if you will, that the Microsoft staffer on the line directs you to a Web page where you can download free of charge the latest release of SQL Server and a free copy of C++ in order to solve your problem, and then follows this up a week later by e-mailing you a program that was written in his/her spare time that extends your program in some new way ..." (Foster, 1997)

The point of this story is that this sort of assistance is shared daily among F/OSS users, though it seems unfathomable to relate to Microsoft. Though it may seem to take more effort, F/OSS users can structure their own preferred style of support, be it e-mail, Internet Relay Chat (IRC), newsgroups, or third party books or manuals. The driving force behind F/OSS, the Internet and online collaboration, gives this support its vitality, . and F/OSS support is virtually open 24 hours a day, 7 days a week. All that is needed to access support which often is superior to proprietary software company's options, is a connection to the internet. But with systems as reliable and well performing as Linux, Apache and MySQL and the many other F/OSS programs in wide use throughout the world, technical support becomes less important. (Wheeler, 2003) (Foster, 1997) (Kenwood, 2001) (Anonymous[13], 2003)

Market Share as a Measure

It is a common belief that the owner of a specific software market is the superior product within that group. If this premise is true then many F/OSS applications are both mainstream and superior products. F/OSS owns several major software markets, and does so with an almost complete dominance. (Wheeler, 2003) (Anonymous[9], 2002)

Recent surveys of software markets have shown:

- *Apache* was the #1 web server with a market share of 59.91 percent.
- *Sendmail* was the most widely used e-mail server (a 42 percent market share).
- *bind* was used by most Domain Name Servers on the Internet (a 95 percent market share).

- *PHP* was the number one server-side web scripting language (a 24 percent market share).
- *OpenSSH,* an implementation of the SSH security protocol, had a 66.8 percent market share. The proprietary version, SSH, had a market share of only 28.1 percent.
- *GNU/Linux* had a of 28.5 percent share of all servers on the Internet, making it number one.

(Wheeler, 2003)

Gnu/Linux has become such a market force that established UNIX vendors, such as IBM, HP and Sun have started promoting it. IBM has gone so far as porting Linux to run on its AS390 mainframe computers, and naming it the successor to its own brand of UNIX, AIX. A recent study by Forrester Research[15] surveyed 2500 IT managers about their understanding and use of OSS. The results of the research revealed that 56 percent currently use OSS and 6 percent more plan on using it within two years. Recognition by such high-level vendors and IT decision makers will fuel the momentum that will only grow the market share of F/OSS. (Wheeler, 2003) (Whitlock, 2001)

Open Source SWOT Analysis

A SWOT (**s**trengths, **w**eaknesses, **o**pportunities, **t**hreats) analysis is a strategic tool for auditing a business and defining its stature within its market or environment. It is a first stage planning technique that assists business analysts in cataloging issues that influence the organization. The following SWOT analysis has been created from previously discussed material and specifically with information from the sources noted at

the end of the listing. The analysis provides a brief inventory of how F/OSS holds up against the IT world as a whole. This assessment tool, as it is applied here to F/OSS, is both subjective and general due to product variations. In other words, this SWOT analysis will be valid for the entire range of F/OSS and may not be an fully accurate analysis of any specific application.

The point here is not to present an in depth analysis of F/OSS as it stands in its current environment, but to inventory issues within the four SWOT categories for consideration in the discussion of F/OSS as a whole. The opportunities and threats are issues that exist from F/OSS's external environment including, but not limited to customers, competitors, suppliers, distributors, barriers to entry, and substitutes. Classifying these attributes inherently leads to some general opinions and conclusions regarding the future of F/OSS. A list of some general conclusions appear at the end of this section.

The F/OSS SWOT analysis listing follows. (a Traditional SWOT diagram / model is shown in *Appendix F*.)

Strengths:

- Inexpensive or free software – no license fee.
- F/OSS is freely distributable, no legal penalty for making copies.
- No license management costs or license related legal risks.
- Less dependence on a specific product or vendor.
- Increased competition within software markets reduce costs and increase quality.

- Improved security, as source code can be verified and certified by outside sources.
- Open standards and formats simplify transactions and reduce data storage incompatibilities.
- Modular, customizable software, users choose modules and customize to suit needs.
- Evolving software, user supported and user driven.
- Rapid release rate; fixes, patches and new versions released often.
- Parallel debugging/development (“Many Eyes”).
- Software often performs better then commercial competition. (e.g. Linux, Apache)
- 24 x 7 technical support via the Internet (e.g. newsgroups, mailing lists, online forums and message boards).
- Availability of the source code, gives users assurance that the disinterest of a vendor, or their complete exit from the market will not effect the software’s availability or viability.
- Users do not have to wait for bug fixes on a vendor’s schedule. Fixes and new features can be produced by end-users and released to the public at will.
- Users are not bound to the strategic decisions made by a vendor.
- Because OSS vendors compete against one another to provide support, the quality of support increases while the end-user cost of support decreases.
- Immense programming expertise garnered by Internet connectivity.

- Large distribution base – Free Internet download sites and independent vendors.
- Corporate support for development, distribution and consultancy services (e.g. IBM, HP, Sun, Oracle).
- Code re-use reduces development time and provides predictable results.

Weaknesses:

- Potential need for extensive migration to replace proprietary systems.
- May require (new) in-house development and maintenance skills within the enterprise.
- Product selection could be difficult (limited marketing).
- Possible interoperability problems with proprietary software formats or standards.
- No brand name assurance.
- Resistance among decision makers, IT personnel and end-users (fear of change).
- F/OSS licenses do not permit ownership.
- More highly educated or skilled administrators needed to maintain F/OSS.
- No GUI or weak GUIs are the norm as many F/OSS programmers prefer command line interfaces.
- Most F/OSS faces the risk of code forking, which ultimately produces different versions of the same product and thus demands a decision on which “fork” to accept.

Opportunities:

- Easy entry into software markets with an existing dominant product or vendor.
- “Going Open” gives vendors the ability to dramatically increase the developer base of existing closed source programs.

- Closed source software released to the F/OSS community gains not only developer support, but that of end-users and independent vendors.
- Governments require sizeable software installations on limited budgets.
- Providing software systems to Third World and other cash poor governments, school systems and non-profit agencies.

Threats:

- Disinterest in a specific F/OSS project could stall or end active development.
- Release of new F/OSS can appropriate current project participants and stall or end active development.
- Customized F/OSS programs can fragment or fork the existing project and undermine its evolution.
- Hybrid open-source projects that build on true F/OSS and are released as commercial ventures.
- Organized FUD (Fear, Uncertainty and Doubt) marketing campaigns by commercial software companies.
- Lawsuits involving intellectual properties (e.g., SCO versus IBM16; see Appendix E) or software patents. Legal actions in this area could set precedents which adversely affect the entire F/OSS community.
- Governmental or enterprise security requirements (or certification) that is rejected by the F/OSS community.

- Version control amongst integrated or compatible programs cannot always be assured as with COTS. (Varghese, 2003) (Kenwood, 2001) (Anonymous[28], 2003)

A review of the F/OSS SWOT analysis as presented in previous listing and *Appendix F* presents the following general conclusions:

1- Third party accessory and F/OSS distribution vendors can thrive by providing added value to “free” products.

2- Opportunities exist to promote and implement modern computing technology using F/OSS in Third World, public sector and non-profit agencies (e.g., schools, governmental agencies, hospitals, etc.); entities that otherwise could not afford the products.

3- Intellectual property rights, software patents and licensing problems will continue to threaten F/OSS as proprietary vendors and developers slowly resign to the needs and will of the market. Infringement issues are easily uncovered and resolved with Open Source software.

4- Opportunities for new and innovative software products will grow as the barriers to entry are lowered.

5- In a F/OSS centric market, computing standards will be driven more by users, and not controlled or imposed by a monopoly.

6- Most legal threats to F/OSS will be mitigated as it becomes mainstream and is more widely accepted as a viable and economically feasible IT solution and business model.

7- Service and support vendors, and IT consulting will become a vibrant and growing market sector, possibly over taking commercial development in monetary volume.

Problem Description

The discussion to this point has been concerned with the nature and benefits of Free and Open Source Software (F/OSS) as compared to its closed source or proprietary equivalents. It has been, admittedly, an argument championing the former. This prejudice was intended to elevate F/OSS to a status equal with COTS or other closed source software. The idea of equivalency is a major element of this research, as it would be worthless to provide a cost analysis between disparate solutions. Now, the focus will shift to the financial view of such comparisons; the quantitative analysis of alternatives. The decision to implement or migrate to a new software infrastructure is a capital investment; and as with any well-planned investment, benefit and risk analyses should be performed to determine the feasibility and profitability of the proposition. (Helfert, 1994)

In order to make realistic and economically sound decisions regarding a company's future IT expenditures, business executives need accurate data on which to base their conclusions. In addition, it is often necessary to justify proposals to non-IT decision-makers by using familiar business metrics that remove technical jargon and reveal the bottom-line. A recent study reports that eighty percent of the companies surveyed use some form of business or financial analysis prior to implementing new IT related projects. Many of the instruments are common analytical models and methodologies familiar to any student of business. These tools, which embrace a wide range of business case methodologies, include Return on Investment (ROI), Cost Benefit

Analysis (CBA), Net Present Value (NPV), Internal Rate of Return (IRR), and Total Cost of Ownership (TCO) analyses. (Schmidt[3], 2003) (Meals, 2002) (Kenwood, 2001)

An analysis that provides decision makers with appropriate and accurate data should include an evaluation of all costs, both direct and indirect, and all benefits, obstacles, risks and scenarios as discovered within the proposed project's life cycle. A valid study should also assess whether the proposal fits the organization's established business strategy and define any deviation from its objectives or policies. While many analytical methods exist that produce such narratives and financial metrics, they do not all give a comprehensive or accurate picture of the subject. A Business Case Analysis (BCA) provides the background, assumptions, risk factors and other internal and external environmental assessments that are part of a sound, fact-based study. (Schmidt[3], 2003) (Helfert, 1994) (Meals, 2002)

A BCA attempts to predict the results of a specific business decision. However, many business case analyses fail due to vague definition and incomplete or limited scrutiny. Just as external variables affect the outcome of a sound BCA so, too, do internal factors such as the type of organization being evaluated. A non-profit or governmental agency, for example, has a different focus and different needs than a profit-making corporation. Such differentiations also extend to a public company versus a privately held one. Any activity that affects the project belongs in a case study. (Anonymous[3], 2003) (Schmidt[3], 2003) (Meals, 2002)

Financial metrics, such as the ROI, NPV and IRR, which are valuable as justification tools, provide the quantitative or fiscal components of a BCA. Depending on

the audience of the study, or the type of organization it describes, the decision to accept the proposal may rest with one or more of these resultant metrics:

- Net Cash Flow (Full valued, Incremental or discounted)
- Net Present Value (NPV) of the forecasted Cash Flow
- Internal Rate of Return (IRR)
- Return on Investment (ROI)
- Payback period of the expenditure
- Total Cost analyses; e.g., Total Cost of Ownership (TCO), Total Capital Costs (TCC), Total Operating Expenses (TCE)
- Cost by a specific event or entity; e.g., Cost per transaction, per person, per seat, etc.

(Anonymous[3]; 2003) (Meals, 2002)

The Gartner Group, who has been credited with first using the acronym, made the TCO analysis a highly valued decision support and justification tool. A TCO study is a case analysis of the direct and indirect financial costs incurred by using a specific product or combination of products. A sound TCO analysis consists of more than the evaluation of the costs of acquisition, or implementation of a system; it also considers the costs for support, maintenance, downtime and out-sourced contracts; essentially all of the costs incurred during a system's entire lifecycle. (Stevenson, 2003) (Gross, 2003) (Cohen, 2003)

Several other cost-based analytical instruments have been proffered by consultancy firms and in some cases the producers of IT products. One tool, slightly different from the TCO and relying more heavily on the ROI analysis, is Giga

Information Group's[17] Total Economic Impact (TEI.) The TEI analysis combines several modeling techniques and claims to return a "holistic" risk-adjusted ROI that is an appropriate decision support tool for both vendor and customer alike. The Standish Group International[18] enhanced the TCO with their Comparative Economic Normalization Technology Study (CENTS.) CENT studies do not utilize any vendor-supplied data, only input from customers. Thus, it purports to give a more accurate view of the actual costs as seen by the consumer or end-user. Even Microsoft has offered its own version of a TCO-like case analysis. Their Rapid Economic Justification (REJ) method offers managers a quick method for evaluating a proposed deployment or other IT decision. This tool promotes the idea that IT decisions cannot wait the months required for a traditional study to be concluded. (Cohen, 2003)

The abundance of models and methodologies for appraising financial justification data marketed by IT research firms can easily dwarf the choices that comprise the original investment question. It seems the principal problem for IT decision makers when planning a software purchase, upgrade or migration is how to acquire accurate, relevant and complete data with which to evaluate their software investment options.

The investment analysis model presented in this paper has been compiled utilizing both pre-existing models and theories and concepts examined during the research. This instrument, hereafter referred to as the *Total Cost model* or the *model*, is a descriptive spreadsheet model. It was constructed to yield accurate and dependable financial decision metrics with as minimal input as possible. Though the general goal of most private sector enterprises is to maximize profits, and that of public sector agencies is to obtain the maximum function with a minimum cost; they in effect are seeking the same

ideal, efficient use of capital. The Total Cost model should work well with both types of organizations. It should not be forgotten, however, that with any financial justification or decision support tool, the decision variables or resulting metrics are only as good as the data supplied. Minor changes to an input variable can distort the results dramatically. As the saying goes, “Garbage in, garbage out.” (Herbst, 1982)

Modeling IT Capital Investments

Many books are available which discuss the analysis and evaluation techniques for capital investments, also known as “capital budgeting”. This paper will not attempt to provide any discussion on this subject except within the context of acquiring input variables and describing financial metrics. Further readings on financial analysis and capital budgeting techniques can be found in the extensive reference listing at the end of this paper. (Stevens, 1983) (Helfert, 1994)

Since the ultimate goal of most every business venture is to maximize profits, investment alternatives are frequently compared using financial analysis models which can be quantifiably evaluated. The simplest form of business decision is where alternative cash outlays are compared to their return on the investment. However, no single financial metric can provide a complete picture of a capital investment and therefore be used reliably as the sole decision variable. The Total Cost model addresses this issue by providing a number of the most popular analytical metrics. (Lumby, 1988)

A financial model is an abstraction of specific financial or cost-centric events that may occur during the situation being studied. In the preparation of the any financial model it is essential to assess all of the significant and driving factors, and to refrain from

oversimplifying the circumstances. A simplistic or over-generalized model may not accurately describe all situations to which it is applied, thus leaving open the possibility of miscalculations and factual misrepresentations. By employing a well-constructed model, analysts can test variations on the represented system and evaluate the quantitative results that are returned. A valid cost model is an essential component of any business case analysis that examines capital investments. This is specifically true with any IT investment as the costs and benefits are often hard to isolate and present in easily comparable terms. (Anonymous[1], 1992) (Lumby, 1988) (Schmidt, 2003)

The Total Cost model is a spreadsheet based financial analysis of the costs involved in implementing or migrating to a new software system. Although it can be used to model a total implementation, it can be easily modified through the input of values, to model any partial or hybrid change. Since much of its input will come from accounting data already available to the user, it is essential to completely understand how this data affects the results and how it can promote inaccuracies.

One of the primary activities of a capital investment business case analysis is the discovery and isolation of related costs. Any costs or purported cost figures that are not obtained through standard accounting procedures should not be used. To avoid any accounting events which may skew the output it is recommended that all cost input figures be based on annual charges. Figure 6 shows the accounting relationships of a capital investment model. This illustration shows that the three major types of financial model input are:

- The organization’s accounting or financial data, e.g. cash flows and cost of capital.

- Data related to the proposed investment, e.g. acquisition costs, prerequisites and constraints.
- Historical data of the company and projections for its future.

All of this input data can be gleaned from a properly executed business case analysis. (Helfert, 1994) (Schmidt, 2003)

Since decisions concerning IT capital investments concern future events, it is important to include only those inputs that can actually be changed by the application of - the proposal. Such costs include preliminary studies of the alternative projects prior to their inception. These "sunk" costs would be realized regardless of the alternative selected and cannot accurately or legitimately be expensed to the selected investment. (Helfert, 1994) (Herbst, 1982)

Although the previous commentaries on F/OSS would lead one to believe it costs less than proprietary software because there is no license fee; it is not completely free. In fact, when enumerating the cash flows during the lifecycle of a F/OSS system, it is critical to remember it is not cost-free. Because software license fees or acquisition costs are often less than five percent of an organization's system cost, they will only comprise a small portion of the TCO. Therefore, the input variables of a TCO model will be dominated by system costs such as, customization, maintenance, support, training, administration, and special hardware needs throughout the system's lifecycle. (Wheeler, 2003) (Hahn, 2002)

Figure 6 - The relationships between the Inputs and Outputs of a financial model.
(Source: Helfert, 1994)

When selecting and using any business model it must be understood that models and other decision tools can be skewed to support most any position. Neutrality in both the selection and use of decision support models is essential for reliable results, which then can be impartially reviewed. The proper use of any investment model should provide pertinent decision variables that can be evaluated by management. However, models alone are not sufficient to justify a capital investment. Capital investments generally involve a complex set of issues that must be understood and resolved to an end beneficial to the company. Capital investment decisions may have impacts beyond the

actual investment itself and its influence may also extend past the investment's lifecycle. All capital decisions are long term commitments which will affect an organization for years and are not easily undone. It is essential to remember that the decision not to invest is also a decision. (Wheeler, 2003) (Daffara; 2000) (Helfert, 1994) (Herbst, 1982)

Hard versus Soft Costs

A well executed IT capital budget case study will include an inventory of costs other than the obvious direct costs. It will also identify any factor of the proposed investment that impacts the organization and attempt to give it a financial valuation. Such a case analysis that includes a TCO study would include factors and valuations that can be classified within three groups:

- Costs
  - Direct (e.g., license fees, implementations costs and consulting fees)
  - Indirect (e.g., downtime and training)
- Benefits (i.e., performance, reliability and security)
- Intangible factors (e.g., quality of support, futz factor and loss of productivity)

The Total Cost model concentrates on and evaluates the first item, costs. The other factors, benefits and intangibles, are equally important and should be addressed in a complete business case analysis, of which a cost model is but a part. (Kenwood, 2001)

A cost model is essentially an organized listing of all costs, their impacts and drivers. Direct costs are generally emphasized in investment case studies because of their

ease of discovery and comparison. They are the tangible factors which are most easily followed through the system's lifecycle. However, the less visible costs or benefits such as performance, downtime, reliability and productivity losses due to the introduction of new software components need to be explained and put into quantifiable terms. Although many indirect costs have attributes and relationships that are difficult to describe or quantify, they may be significant factors and should be investigated and considered in the decision making process. Indirect costs are often included within a stated direct cost, but the relationship of these expenses should be examined by the case analysis. For example, labor expense due to lost productivity is an indirect cost of a project, but there is no additional cost to apply to the analysis as the added cost should already be included in the total (direct) labor cost. These indirect costs are not so much a new cost item as they are an adjustment to the cost of labor as a whole. This sounds obvious, but knowing where costs originate is the purpose of any business analysis, and the origin of some may not be so evident. It is essential to discover as many indirect costs and define their relationships to other costs as possible. (Kenwood, 2001) (Schmidt[1], 2003) (Herbst, 1982) (Anonymous[5], 2003)

The direct and indirect costs of a F/OSS implementation are illustrated in the *Cost Element Taxonomy for Open Source Software* by Kenwood (Table 5.) Direct costs of a software system can be easily measured and categorized throughout the lifecycle, however, indirect and other intangible costs are not as easy to isolate. Even though factors such as performance, security and scalability do have a substantial economic impact; they are difficult to quantify, and thus face the possibility of exaggeration or

misstatement. Any beneficial factors which cannot be evaluated in accounting (or financial) terms should be left in the business case analysis for review. (Kenwood, 2001)

A major consideration when evaluating investments, is the benefits versus the cost of each alternative. A system that costs more to acquire and implement, but sharply increases productivity may prove to be the wiser investment. Thus, a higher cost system may be the better investment in contradiction to metrics such as ROI or TCO. The evaluation of costs should also include the purchase or lease of the system or portions of it from third party vendors, which may increase the cost of acquisition, but in return reduce other costs considerably. However obtained, all benefits should be weighed against the risks or costs of the specific alternative. This is especially true with F/OSS since many hard to quantify benefits such as the ability to modify or customize code are some of its major advantages. (Kenwood, 2001)

Benefits should be included in case studies regardless of their ability to be quantified or assigned any financial value. Every effort should be tried to place a value on a benefit worth mentioning in the case. Some ways to quantify a benefit that is not easily quantifiable are:

- Place a financial value on its effects.
- Set the value equal to that of the alternative solution.
- Set the value equal to the cost of NOT applying the solution.

(Schmidt[1], 2003)

Direct Costs
Software and Hardware
Software
Purchase price
Upgrades and additions

			Intellectual property/licensing fees
	Hardware		
			Purchase price
			Upgrades and additions
Support Costs			
	Internal		
			Installation and set-up
			Maintenance
			Troubleshooting
			Support tools (e.g., books, publications)
	External		
			Installation and set-up
			Maintenance
			Troubleshooting
Staffing Costs			
	Project management		
	Systems engineering/development		
	Systems administration		
			Vendor management
	Other administration		
			Purchasing
			Other
	Training		
De-installation and Disposal			
Indirect Costs			
	Support Costs		
	Peer support		
	Casual learning		
	Formal training		
	Application development		
	Futz factor		
Downtime			

Table 5 - Cost Element Taxonomy for Open Source Software (Source: Kenwood, 2001)

A close scrutiny of training or support factors can reveal unexpected costs and benefits. The current condition and type of system being replaced is an important factor that can impact the cost of a new implementation. UNIX users would find it easier to transition to Linux or another UNIX-like system than would users of Microsoft's Windows systems, for example. Retraining costs and lost productivity would be higher in the latter case. Additionally, IT staff members will typically train and accept new

systems faster than the typical end-user, but their education comes at a greater cost. However, such hidden expenses are not only an issue with new implementations, as every new software release or upgrade requires some user training which results in lost productivity. Seamless, cost-free transitions are not always the case when simply upgrading an existing system. (Cohen, 2003) (Anonymous[9], 2002) (Scott, 2002)

Other factors that can affect the TCO, such as personnel matters, are more obscure. High employee turnover can increase the TCO by as much as fifty percent due to additional training and IT costs involving the new users. The Gartner Group terms another issue the "Futz factor", which is the indirect cost of employees using an organization's computing assets for personal use during work hours (see Table 5.) These expenses cannot be accurately measured, but should be considered in the overall evaluation of alternatives. (Hahn, 2002) (Kenwood, 2001)

Other intangibles may be the real or perceived characteristics of the proposed system. Detractors of F/OSS, for example, point out the lack of applications that are supported by F/OSS systems and the complexity of an end-user's desktop system. However, this can be a benefit to a certain extent as it may reduce the effect of the Futz factor, as users have less access to games and other diversionary pastimes. This alone can yield great savings when compared to a "user-friendly" system with a plethora of entertainment and recreational programs or simply the friendliness to allow diversions from one's duties. (Kenwood, 2001)

In order to accurately define costs, managers and analysts need to have a comprehensive classification, or matrix of costs, benefits and other intangibles occurring throughout the system's lifecycle. Table 6 shows a simplified version of a project cost

matrix which is similar to the cost matrix used in the Total Cost model. This table is also a sort of cost model in itself, though it should be expanded upon as it is very shallow and does not cover every possible category. A matrix such as this will help to track all expenses and other financial impacts and should aid in avoiding a duplication of cost allocations, or worse, overlooking a cost altogether. The taxonomy presented by Kenwood (Table 5) can be used as an initial inventory of expenses, but a valid case and cost model must view all of these costs as they are allocated throughout the proposed system's lifecycle. Table 6 takes the listing shown in Table 5 and expands it to breakout the costs by the three major periods of a system's lifecycle (excluding the end of life period which primarily concerns the system's disposal and replacement.) The Total Cost model provides a more detailed listing, which is generic by design and does not address costs according to the system's lifecycle, but rather divides them by the annual periods of the study. Completing a cost matrix such as illustrated in Table 6 is part of a thorough cost study and will provide input for the Total Cost model. (Kenwood, 2001)

	Proposed Project's Lifecycle		
Cost Category	**Acquisition and Implementation**	**Operation**	**Maintenance**
Direct Costs			
Software			
Hardware			
Support			
Personnel			
Contract Services			
Facilities/Overhead			
Additional Costs			
Indirect Costs			
User Training			
User Support			

Table 6 - Project Cost Matrix – identifying expenses attributable to a project.

Methods of Analysis

Two major decisions must be made when constructing a cost model similar to the Total Cost model. First, the resulting metrics that provide both descriptive and practical information must be selected. The specific measures or metrics used by an organization should be selected by its management, since the same results will mean different things to different organizations and managers. (Morrow, 1991)

Secondly, the input data, based on the requirements of the specific financial analysis must be determined. In fact the choice of the organizations' financial input data, though adequately prescribed by the formulas used in the model, are the most critical, and in some cases, the hardest to acquire. Allocated costs should be avoided when they have been arbitrarily established. Consistency regarding input variables is a must. A TCO analysis, as in the form of financial analysis presented by the Total Cost model, is sensitive to both the data and the assumptions made about the investment. (Wheeler, 2003) (Helfert, 1994) (Schmidt[1], 2003)

Variables that are significant in the capital budget decision process are:

- The initial cost of the project
- The useful life of the project
- Net incremental cash flows attributable to the project
- The project's salvage or disposal cost.
- and, the organization's cost of investment capital

These parameters together with other variables, both qualitative and quantitative, can provide the data needed for a reasonable and well-grounded investment decision. However, as mentioned previously, the acquisition of accurate data with which to populate these parameters is not a simple or even well-defined task. (Herbst, 1982) (Helfert, 1994)

Capital budgeting requires the use of net cash flows, as opposed to accounting profits. To accurately portray any investment option only the cash inflow, or receipt that can be directly attributed to the project must be used. Cash flow figures are not easily obtained in a large organization, and may require external expertise to define. However, without this distinction the "profits" attributed to a project may not actually be a result of its implementation, but rather some other source altogether. Or, the inflow may even be the result of a synergistic effect of the project and another asset within the organization. (Herbst, 1982)

Although the proposed model is called a "Total Cost" model, it presents metrics other than the simple TCO. Some of the metrics presented by the model could be enhanced by employing other formulas, such as better representing the Time Value of Money (TVM), which suggests that a dollar received today is worth more than a dollar received five years from today. This concept is true apart from inflation, since cash held today could be invested at a given rate making it more valuable that the dollar received in the future. However, considering inflation or any other economic conditions merely confuses the idea that money today can be used to make more money. Of course this would be in the future, and according to the TVM would be worth less than the originally invested dollar. Other methods (or formulas) do exist which return different metrics;

nevertheless, the formulas used in the Total Cost model are the methods found most often during this research. These metrics are listed in Table 8 along with the input data they require. The actual spreadsheet formulas used in the model are displayed in *Appendix G*. (Stevens, Jr. 1983)

In order to understand the following definitions of the financial analyses used by the Total Cost model, a clarification of some common terms must be made. First, and possibly most importantly, the model uses cash flows that occur at the end of each year and are uneven. Since a number of the resulting metrics can use either uniform or uneven cash flows this is important to understand; especially since the results can vary significantly depending on the type of cash flow. Several of these analyses refer to the *MARR*, or the Minimum Accepted Rate of Return. This is the minimum interest rate that an organization expects to receive from any investment and is sometimes called the "hurdle" rate. The cost of capital variable as mentioned in the following metric definitions should represent an accurate estimation of future finance rates and not be based on historical values. The Total Cost model does not require a specific tax weighted value, but a before tax interest rate is suggested for input values for simplicity. (Stevens, Jr., 1983)

The analytical metrics returned by the Total Cost model are:

- **Total Cost of Ownership –** The definition of TCO is not at all well established, except that it is the sum of the cost drivers, or cash outlays for the subject of the study during the length of that study or for its complete lifecycle. The disparity concerns both the selection and manipulation of the input data, but it is generally agreed that the solution with the lower

TCO is the best choice (all other things being equal.) The major drawback to the TCO methodology is that it is dependent on any assumptions or input data, which vary depending on the exact model used.

- **Decision rule:** The alternative with the lowest TCO should be selected.

- **Simple Return on Investment (ROI)** – The Simple ROI states the rate of return of all cash flows over the period of analysis. This formula does not take into account the time value of money. That is, money returned earlier in an investment is worth more than money received at the end. The formula which is used in this model for the Simple ROI calculation is:

  ROI = (incremental inflows - incremental outflows) / incremental outflows

  - **Decision rule:** The alternative with the larger Simple Return on Investment is the better investment.

- **Discounted Return on Investment (ROI)** – The Discounted ROI states the rate of return of all cash flows over the period of analysis with those cash flows discounted to their present value. The timing of cash flows is critical to an accurate discounted ROI measure because it is a time value based calculation, and money returned earlier in an investment is worth more than money received later. The discounted ROI formula used in this model is:

  ROI = (Discounted incremental inflows – Discounted incremental outflows) / Discounted incremental outflows

  - **Decision rule:** The alternative with the larger Discounted Return on Investment is the better investment.

- **Payback Period** – This is one of the more basic capital budgeting measures; it simply states when the project will pay for itself. Usually stated in months or years it is the point at which the cash outlays match the revenue or income attributable to the project. The Total Cost model returns the Payback Period in years, with the fraction of a year (months) in decimal form. The Payback method has several major drawbacks that should be taken into account when comparing its metrics. First, it does not recognize the cost of funds utilized or consumed by the investment; secondly it does not consider any return or cash inflows beyond the payback period, and finally it does not acknowledge the time value of money. Although it has these limitations, it is often used as a measure of the effectiveness of an investment.
    - **Decision rule:** The investment that meets or approximates the organization's requirement for payback should be selected. Faced with a selection of alternatives the investment with the lowest Payback period should be chosen
- **Discounted Payback Period** – This form of the Payback calculation uses cash flows that have been discounted to a present value using the currently selected cost of capital. Just as with the simple Payback method it, too, states when the project will pay for itself in years.
    - **Decision rule:** The investment that meets or approximates the organization's requirement for payback should be selected. Faced with

a selection of alternatives the investment with the lowest Payback period should be chosen

- **Internal Rate of Return (IRR**) **–** This is a Time Value of money metric related to the Net Present Value (NPV.) It represents the discount (interest) rate at which the NPV is zero. The IRR is the rate of return provided by the project over its lifetime or the analysis period. Since the returned metric is based on an NPV of zero, it could be assumed that the IRR is calculating the break-even point at the specified interest rate. However this is not a true breakeven point as the returned interest rate might actually cost the organization money if it is not equal to or higher then the MARR.
    - **Decision rule:** Investments should only be made in projects where the IRR value is greater than or equal to the organization's MARR.
- **Modified Internal Rate of Return** (**MIRR) –** This metric considers both the cost of the investment (cash outflows) and the interest received on reinvestment of cash inflows. The finance rate is the organization's cost of capital and the reinvestment rate is the interest rate received on money reinvested (usually the MARR.) The MIRR is an Excel function and the actual algorithm or formula is not available, though it is a standard formula found in most spreadsheet applications. The MIRR function returns the Modified Internal Rate of Return for a series of periodic cash flows.
    - **Decision rule:** Because this metric reinvests all cash inflows from the investment, the returned value should be greater than or equal to the

organization's predefined interest rate for a return on an investment (or MARR.)

- **Net Present Value (NPV) –** The NPV uses the Present Value of money concept to represent the sum of cash flows as adjusted for the time value. The NPV calculation uses the MARR as a baseline to sum the net cash flows to a present value. A positive NPV can be viewed as the excess profit above the return of the original investment and the reinvested returned cash (reinvested at the MARR.) A negative metric reveals the amount of capital lost through the investment including any returned cash reinvested at the MARR. The simplified formula for the NPV is:

  NPV = discounted cash inflows - discounted cash outflows

  - **Decision rule:** Investments should only be made in projects with an NPV greater than or equal to zero. That is, only positive values.

- **Net Present Value (NPV Sum of Discounted net cash flows)** – This NPV measure is derived by summing the discounted net cash flows for each year in the analysis.

  - **Decision rule:** Investments should only be made in projects with an NPV greater than or equal to zero.

- **Profitability Index (PI) –** This ratio is a measure of a proposal's profitability per dollar invested. A value of 1 or higher indicates an investment yield greater than the cost of the project based on the cost of capital (discount rate.) The Profitability Index formula is:

  PI = discounted cash inflows / discounted cash outflows

- **Decision rule:** The investment with the highest PI has the best profitability and should be selected.

(Anonymous[10], 2003) (Helfert, 1994) (Finney, 1994) (Wheeler, 2003) (Clark.,1984) (Cohen, 2003) (Stevens, Jr., 1983) (Larson, 1993)

The Total Cost Model

Whether you call the Total Cost Model a financial model, a workbook or a spreadsheet, it is nothing more than a tool for comparing input data and showing possible numerical relationships. All of its results can be interpreted in different ways, just as the same statistics can be used to validate opposing opinions.

Worksheets		
Order	**Name**	**Purpose**
1	Common Data	Current, general and common organization data
2	Alternative System Costs	Cost data for each Alternative (3 total worksheets)
3	Cost Analysis	Cost analysis based on data from worksheets 1 & 2
4	Graphical Analysis	Graphical display of the cost data from worksheet 3
5	Concepts and Instructions	General model user help

Table 7 - Description of the worksheets within the Total Cost Model.

The Total Cost model is a spreadsheet (or workbook) that consists of eight worksheets, or tabbed pages (see Table 7.) It was created using Microsoft Excel 2003, version 11. Since it allows the simultaneous comparison of three alternative projects, it required a good deal of programming code to provide logical flow, error checking, data validation and arithmetic calculations to be significant. See *Appendix H* for examples of

the Visual Basic for Applications (VBA) programming code used to control the workbook. The Total Cost model is contained in the file named "Cost_Model.xls". All screenshots in the following text were captured on a Windows XP system running Microsoft Excel 2003; however, the workbook was also tested using Excel 2000 running under the Windows 2000 operating system with no problems or anomalies noted. No guarantee is made that the workbook will perform as described on older versions of Excel, so care should be taken to further insure that the results obtained on such systems are accurate. Many of the programmed functions are interdependent, thus it is recommended that the spreadsheet be used without any modifications.

In an effort to protect users from macro (or programming code) viruses embedded within workbooks, Excel employs a *Macro Security* scheme which has four levels of protection. The two highest levels only permit macros from "trusted" sources to be run. Since the model will not function as designed without the use of its macros, Excel's Macro Security level should be set to Low to ensure that the model works properly. It is also important to remember that Excel cannot detect any specific virus, and all of its warnings about macro viruses are general warnings about a possible problem.

The Total Cost model provides the following features:

- The model supports results and income of up to $999,999,999,999.99 (One cent less than one trillion dollars).
- Study lengths can be customized as needed (one, three, four or five years).
- One to three alternative systems can be evaluated simultaneously.
- Each system can add up to five user defined cost fields.

- Worksheets can easily be printed to serve as a hard copy worksheet or scratchpad
- Four charts are predefined, though the user can add more using Excel's built-in charting capabilities
- Ability to print or e-mail the results or the entire workbook from the worksheet's custom menu (see Navigating through the Model)
- A user Options or Preferences interface that allows customizing the following settings:
    - Automatic workbook backup at startup
    - Display of the workbook in full screen mode
    - Show the splash dialog during startup
    - Show row and column headers
    - Screen display percentage (ZOOM)

Many calculated cells have been placed in "hidden" columns to protect them from unintended manipulation. These cells are essentially scratchpad calculations and values. For a better understanding of how the Total Cost Model collects and calculates its values, unhide columns J through P on the Proposed System Data sheets; and columns L through P on the Cost Analysis sheet to view these cells.

How the Model Works

The true beauty of the Total Cost model is that the input can be continually refined and updated as the user acquires new information or means of stating their accounting

data. The main concern when preparing your accounting data for input into the model should always be: The output is only as reliable as the input.

There are six basic steps to using the Total Cost model:

1. Determine what data you will use and how that data will be acquired.
2. Assemble the data as required by the model.
3. Enter the general organization data on the first worksheet, "Common Data".
4. When a name is entered in a "Proposed System" cell on the Common Data worksheet a new worksheet will be displayed to contain the cost values of that alternative system. The new worksheet will be named after the value entered into the Proposed System cell.
5. On each Proposed System worksheet, enter the system specific costs; each worksheet defines the cost variables for one system.
6. Finally, view the worksheet "Cost Analysis" for the results of the analysis. This worksheet is automatically updated when all of the appropriate data is entered. Since some calculations do not require every piece of data that can be entered into the model, results will be presented once there is sufficient data to calculate that value. Charts are displayed on the worksheet “Graphical Analysis” which becomes visible as soon as there are results to plot.

Each worksheet has help information specific to it below the input areas. More comprehensive help is available on the “Concepts and Instructions” worksheet (Figure 7.) Further assistance is available through Excel’s commenting feature, which is used extensively throughout the model.

Acquiring the Data

Before using the Total Cost Model, you should familiarize yourself with capital investment analyses such as ROI, IRR, and NPV.

Once you understand how to use this model and the information it provides, the first and most important step is the acquisition of input data. Though this information may come from an organization's accounting records, some users may find it easier to take the standard accounting data and conform it to the needs of the model.

Figure 7 - Concepts and Instructions worksheet

It is best to use revenue cash flows that represent the difference between the "business as usual" figures and those achieved with the implementation of the proposed project (investment.) Thus, the revenues should represent the benefits provided by the new system and the expenses, the costs directly assignable to it. However, if cash flows

cannot be attributed directly to computer systems or other IT managed resources, use the company's total revenues. Though this may not be as accurate, it will provide a valid comparison since each alternative will have the same revenue. As with any projected cost model such as this, there will always be some disagreement in its execution.

Another important question to answer when deciding on which accounting figures to use is; should you use only marginal (additional) or total costs. The idea behind capturing capital investment metrics is exposing and identifying the added or marginal costs behind a proposed investment. For example, should the complete expense for all of the organization's employee end-user training be used or just the amount of change required by the proposed project? This is up to the user to decide, however, consider the data collection and usage as a whole and be consistent. Total expenses should include any marginal costs so they would be acceptable. To provide accurate results the same type of numbers should be used throughout the analysis, this includes the Common organization data on the first worksheet.

Metric	**Required Data**
Total Cost of Ownership (TCO)	Cash outflows, project expenses, or cash outlays
Simple Return on Investment (ROI)	Evenly spaced (regular intervals) series of periodic cash flows (incremental inflows and outflows)
Discounted Return on Investment (ROI)	Evenly spaced (regular intervals) series of periodic Discounted cash flows (incremental inflows and outflows)
Payback Period	Evenly spaced (regular intervals) series of periodic cash flows (incremental inflows and outflows)
Discounted Payback Period	Evenly spaced (regular intervals) series of periodic Discounted cash flows (incremental inflows and outflows)
Internal Rate of Return (IRR)	Evenly spaced (regular intervals) series of periodic cash flows (incremental inflows and outflows)
Modified Internal Rate of Return (MIRR)	Evenly spaced (regular intervals) series of periodic cash flows (incremental inflows and outflows), and a cost of capital and reinvestment interest rate (MARR)
Net Present Value (NPV)	An interest rate required by the organization, and a evenly

	spaced (regular intervals) series of periodic cash flows incremental inflows and outflows)
NPV - Sum of Discounted Net Cash flows	An interest rate required by the organization, and a evenly spaced (regular intervals) series of periodic Discounted cash flows (incremental inflows and outflows)
Profitability Index (PI)	Evenly spaced (regular intervals) series of periodic cash flows (incremental inflows and outflows)

Table 8 - Data requirements for the metrics as calculated by the Total Cost Model

Some of the model's calculations require that the cash flow be stated as incremental flows; however, the insertion of total data is supported, though it will return a lesser set of resulting metrics. Regardless of the manner of selecting data, the model will produce valid metrics based on the data. The model simply provides the user with financial metrics calculated using the data supplied to it; all results are based on the input data.

Navigating through the Model

Other than the VBA programming code (macros) which provides structured data entry and advanced error control, the Total Cost Model functions as any Excel workbook. It is a collection of tabbed worksheets displayed in a multiple document interface (MDI.) Each worksheet is activated and brought to the front of the other worksheets by clicking on its tab. Most every standard Excel navigation technique is supported, though some actions may either be disallowed or hampered by macros. This is especially evident as all of the worksheets are "protected" by default using Excel's worksheet level protection, which prevents changes them. In the case of the Total Cost model, the worksheet protection prevents a user from changing its formatting and underlying formulas. This

feature can be over-ridden, but it is best to leave it activated, as any user can inadvertently damage the model's structure, thus rendering its results unreliable.

Access to the extended functionality of the Total Cost model is afforded by the selection of either a menu item or a command button (see Figure 8.) Each worksheet has hyperlinks or buttons that will provide services or run processes. Most every navigation aid and function provided by buttons located on the worksheets is also available from Excel's main menu. The "Total Cost Model" menu is located next to the standard Excel Help menu. It supports many functions and is context based, so only menu items which are relevant to your current activity are available.

Figure 8 – The Total Cost Model's menu and command button design.

Entering Data into the Model

The variables on the Common Data worksheet apply to the entire organization and are applied throughout the model. These are the constants used in evaluating each system's own cost variables. To start entering data in this worksheet, first select a length

of the analysis in years (the Period), then, enter the organization's cost of capital. From this point on the required numbers are basic accounting or money figures. Hover the mouse pointer near the small red triangles located in some of the cells to display the cell comments that explain the required input or the calculations that have been produced (see Figure 9.)

Although most of the Common Data input is straightforward, the model's data validation routines will catch many types of input errors that may occur. The following input data are required by the Total Cost model (see Table 9):

- **Analysis Period -** Choose the number of years for the cost analysis; one, three, four, or five years only. A one-year period works well when only one set of inflows and outflows are available.
- **Cost of Capital -** The organization's cost for borrowing investment capital. If there are several rates available, use an average. Remember that this is a *future* interest rate.
- **Reinvestment Interest Rate (MARR) -** This value is the interest rate paid to the organization for reinvested cash. This could be bank interest or the required rate of return for investing in other projects. This value is also known as the MARR, or the Minimum Accepted Rate of Return.
- **Total Current Revenue (Year 1 of analysis)** – This value is the organization's Total Revenue for the first year of the study. This should be the total of all revenue attributable to the proposed system for the first year, however if this number is not available use the organization's total revenue. As long as comparable figures are used, there should be an

accurate comparison between the competing systems. Future cash inflows (revenues) can be estimated by entering a "Projected Yearly Growth Rate", or the annual figures can be manually entered. Using the Growth Rate will provide a constant growth in revenue, which may or may not be realistic for the organization or industry. Holding the revenue constant will make for easier interpretation of the TCO results. Revenue numbers weigh heavier in the ROI, IRR, NPV and Payback calculations.

- **Projected Yearly Growth Rate (optional)** - Use this variable instead of providing values separately for each year under "Yearly Total Revenues". This value will be used to compound each year's revenue to produce the "Yearly Total Revenues" values. The starting value for these calculations is the "Total Current Revenue", which is the first year's revenue within the analysis.
- **Yearly Total Revenues** – These values are either automatically calculated by compounding them using the growth rate as previously entered, or by manually entering values into the cells.
- **Proposed Systems** - These values will only be used to "label" the results. They have no meaning to the financial metrics and can be anything alphanumeric, the shorter and more informative the better. For the proper display within the model, the names should be twenty characters or less.

Other than entering cost values within the cells of the Alternative System Costs worksheet, the only other feature to note is the use of user-defined cost fields. Up to five custom system cost fields can be added to the Proposed System worksheets. Each user-

defined field will allow itemizing expenses by each Period in the study, just as with the other fields. Once added on one worksheet, every proposed system will have the same fields available on their respective worksheet, so this only needs to be done once for the entire study. After the rows have been added both the field label and all of the values for each year can be changed.

Adding, modifying or deleting user-defined cost fields is performed by clicking the Add/Edit Fields button, at the end of the static cost categories (next to the label "User Defined Costs" category label.) This opens a dialog box from which you can add, modify or delete up to five custom fields. The interface is very simple to master, however, for safety's sake it should be tried before entering any critical data into the model.

Worksheet	**Required input**
Common Data	Analysis Period Cost of Capital Reinvestment Interest Rate (MARR) Total Current Revenue (Year 1 of analysis) Projected Yearly Growth Rate Yearly Total Revenues Proposed System Names
Alternative System Costs	System costs by Year, then by Cost category User defined costs as needed (optional)

Table 9 - Input data requirements by worksheet

Viewing the Results of the Analysis

All of the financial metrics calculated by the Total Cost model come from cash flows that occur at the year's end and are not uniform (uneven.) This is important to remember as metrics such as the NPV and the Payback period, for example, can be calculated both with uniform and uneven cash flows. Any study of the model's results

should take the input data into account. More insight into how the model calculates its values can be obtained by examining the formulas it uses. These formulas, as copied directly from the workbook's cells, are displayed in *Appendix G*.

Figure 9 - Common Data worksheet

Attempting to make generalized interpretations of the results produced by the Total Cost model would be futile, and in fact misleading. Capital budgeting requires that the decision makers predict the future, and there will always be differing visions. Additionally, the selection of alternative investments relies on both intuition and a knowledge of the specific situations being studied, which makes financial analyses just a portion of the complete decision process. The descriptions of the financial analyses (see Methods of Analysis) used by the Total Cost model should be sufficient for any user to draw elementary conclusions based on the calculated metrics. The Cost Analysis worksheet provides a minimal analysis of the "Best Performing System" which simply selects the alternative that meets or exceeds the decision rule as stated on both the worksheet and in the previous discussion. This evaluation is provided as a convenience and may not be accurate considering the total scope of the decision in question. (Larson, 1993)

The results of the Total Cost model are displayed numerically on the Cost Analysis worksheet (Figure 10) and graphically using charts on the Graphical Analysis worksheet (Figure 11.)

The Cost Analysis worksheet contains information in three distinct areas:

- Itemized Total System Costs by cost category – This listing shows the totals of each cost category for each alternative (or proposed) system, and the total cost for the systems

Microsoft Excel - Cost_Model.xls

Proposed System: OpenOffice

Reset Values

Help for this Worksheet

	Year	1	2	3
Software				
	Initial Purchase Cost:	$0.00	$0.00	$0.00
	Additional Licenses and Subscriptions:	$0.00	$0.00	$0.00
	Upgrade and Maintenance Fees:	$0.00	$0.00	$0.00
	Add-on or Utility Application Licenses:	$0.00	$0.00	$0.00
	System Management Software:	$0.00	$0.00	$0.00
	License Management:	$0.00	$0.00	$0.00
	Other Software Expense:	$0.00	$0.00	$0.00
Hardware				
	Original Purchase Cost:	$0.00	$0.00	$0.00
	Additional Hardware required:	$0.00	$0.00	$0.00
	Data Storage and Storage Systems:	$0.00	$0.00	$0.00
	Telecommunication Infrastructure and Systems:	$0.00	$0.00	$0.00
	Other Hardware Expense:	$0.00	$0.00	$0.00
Personnel				
	System Design and Engineering:	$0.00	$0.00	$0.00
	Project Management:	$0.00	$0.00	$0.00
	Temporary Employees (Design & Implementation):	$0.00	$0.00	$0.00
	Vendor and Contract Management:	$0.00	$0.00	$0.00
	System Design and Engineering:	$0.00	$0.00	$0.00
	Ongoing System Administration:	$0.00	$0.00	$0.00
	Application development (in-house):	$0.00	$0.00	$0.00
	Specialized Contract Personnel:	$0.00	$0.00	$0.00
	Other Personnel Expense:	$0.00	$0.00	$0.00
Contract Services				
	Design and Implementation Contracts:	$0.00	$0.00	$0.00

Common Data | OpenOffice - System 1 | StarOffice - System 2 | Microsoft Office - System 3 | Cost Analysis

Ready NUM

Figure 10 - Alternative System Costs worksheet (3 year analysis period)

- System Cash Flows by Year - This section displays each alternative system's Expenses (Outflows), Revenues (Inflows), Cumulative Incremental cash flow, and the Net Incremental cash flow by year. Totals of all these yearly figures are also calculated, with the exception of the Cumulative Incremental cash flow, which by definition has no total.
- Financial Metrics – The ten resulting metrics, or financial analyses, are presented with a comparison against each alternative using the decision rule as a basis. The best performing metric, according to the decision rule as previously stated, is noted.

The Cost Analysis worksheet also includes several command buttons that provide the following additional functionality beyond the calculation of the financial metrics:

- **View Graphs** – Moves the focus to the Graphical Analysis worksheet, which shows the analysis' results plotted on Charts.
- **Print the Results** - Opens the Cost Analysis worksheet in an Excel Print Preview window. The worksheet is ready to print with margins and page breaks already set, however these settings can be changed to match both your printer and your preferences.
- **Mail the Results** – This button opens a form that allows the sending of specific portions of the workbook, or the entire file, to any valid e-mail address. This is a convenient way to share the results or transfer the Cost Model to another computer.

Microsoft Excel - Cost_Model.xls

Help with the Results

View Graphs...

Print the Results...

Mail the Results...

ReCalculate the Results...

Proposed Systems:	OpenOffice	StarOffice	Microsoft Office
Software	$1,000.00	$0.00	$300,050.00
Hardware	$0.00	$0.00	$0.00
Personnel	$5,000.00	$0.00	$0.00
Contract Services	$4,700.00	$0.00	$0.00
Support and Training	$24,300.00	$54,500.00	$11,900.00
Additional Costs	$0.00	$0.00	$0.00
User Defined Costs	$0.00	$0.00	$0.00
Total System Costs	($35,000.00)	($54,500.00)	($311,950.00)

System Cash Flows by Year:		Year 1	Year 2	Year 3	Year 4	Year 5	Totals
OpenOffice	Expenses (Outflows)	($35,000.00)	$0.00	$0.00	N/A	N/A	($35,000.00)
	Revenues (Inflows)	$500,000.00	$522,500.00	$546,012.50	N/A	N/A	$1,568,512.50
	Cumulative Incremental Cash Flow	$465,000.00	$987,500.00	$1,533,512.50	N/A	N/A	
	Net Incremental Cash Flow	$465,000.00	$522,500.00	$546,012.50	N/A	N/A	$1,533,512.50
StarOffice	Expenses	($54,500.00)	$0.00	$0.00	N/A	N/A	($54,500.00)
	Revenues	$500,000.00	$522,500.00	$546,012.50	N/A	N/A	$1,568,512.50
	Cumulative Incremental Cash Flow	$445,500.00	$968,000.00	$1,514,012.50	N/A	N/A	
	Net Incremental Cash Flow	$445,500.00	$522,500.00	$546,012.50	N/A	N/A	$1,514,012.50
Microsoft Office	Expenses	($311,250.00)	($350.00)	($350.00)	N/A	N/A	($311,950.00)
	Revenues	$500,000.00	$522,500.00	$546,012.50	N/A	N/A	$1,568,512.50
	Cumulative Incremental Cash Flow	$188,750.00	$710,900.00	$1,256,562.50	N/A	N/A	
	Net Incremental Cash Flow	$188,750.00	$522,150.00	$545,662.50	N/A	N/A	$1,256,562.50

Financial Metrics	OpenOffice	StarOffice	Microsoft Office	Best Performing System:	Best Result
Total Cost of Ownership (TCO)	$35,000.00	$54,500.00	$311,950.00	OpenOffice	$35,000.00
Simple Return on Investment (ROI)	4381.46%	2778.00%	402.81%	OpenOffice	4381.46%
Discounted Return on Investment (ROI)	4020.41%	2546.13%	362.42%	OpenOffice	4020.41%
Payback Period (years)	1.0	1.0	1.0	3-way Tie	1.0
Discounted Payback Period (years)	1.0	1.0	1.0	3-way Tie	1.0
Internal Rate of Return (IRR)	1426.38%	912.15%	132.73%	OpenOffice	1426.38%
Modified Internal Rate of Return (MIRR)	451.94%	342.31%	85.06%	OpenOffice	451.94%

Figure 11 - Cost Analysis worksheet

- **ReCalculate the Results** – This button forces the Cost Analysis worksheet to refresh by recalculating it's values.

The last analysis page, "Graphical Analysis", provides four predefined charts of the model's resulting metrics. Though the results are presented in clearly defined and generally accepted financial metrics, graphical representations may enhance the users understanding of these numerical values. Figure 12 shows the Graphical Analysis worksheet reduced to a screen display of thirty-five percent to show the entire chart on one screenshot. As with any Excel workbook the user can add or modify charts as they wish. The charts are provided as a sample of ways to contrast the results of the model. Briefly explained, the four predefined charts are:

- **Total Costs during "x" Years of Analysis** - This chart plots the total costs for each system on a line chart. Since a line chart does not provide any valid information for a one-year analysis, it is not available in this case.
- **Comparison of Internal Rate of Return Metrics** – This is a comparison of the IRR with the MIRR for each alternative system displayed as a column (vertical bar) chart.
- **Breakdown of Total Cost for…** - The percentages breakdown of each cost category for the alternative systems is displayed in pie charts.
- **Comparison of all Investment Return Measures** – This is a column chart comparing the rate of return metrics for each alternative. The values displayed on the chart are:
    - Simple Return on Investment (ROI)

- Discounted Return on Investment (ROI)
- Internal Rate of Return (IRR)
- Modified Internal Rate of Return (MIRR) by system

Figure 12 - Graphical Analysis worksheet (display reduced to thirty-five percent)

Preferences

The Preferences or Workbook options form is a central location for performing several routine configuration or file maintenance procedures. It is accessed either by clicking the Preferences command button on the Common Data worksheet, or by selecting the “Preferences/Options...” menu item from the Total Cost Model menu. The Preferences form offers the following features:

The System information section provides a display of the version of Microsoft Excel and the operating system currently running on the PC, and the amount of memory that the Total Cost model is currently using (in bytes.)

Figure 13 - Preferences / Workbook options dialog box

The Current Filename section displays the filename of the currently loaded Total Cost Model file. Clicking the “Change Filename or Path...” button opens the standard Windows *Save As* dialog box with the current filename used for the filename to be saved. The button labeled “Backup the Workbook...” runs a preset backup procedure that also opens the Windows *Save As* dialog box, but appends the current date as an identifier to the filename. The latter method still allows for user changes to both the filename and the

destination (or file path) prior to saving the file. All of these file procedures allow the user to cancel the operations once they have been called.

The Program Options section configures several features that can be set using Excel's own menus or controls. However, they are made available here for convenience and the settings are saved to the Custom Document Properties of the Total Cost model. By using the Custom properties feature of an Excel document, the settings are guaranteed to be used regardless of where the Total Cost model is opened (as long as a compatible version of Microsoft Excel is used.)

The first four settings are toggle settings, meaning they are either on or off. The last one, the Screen Display or Zoom value, is changed by either entering a value or clicking the spinner buttons at the right side of the input box. The five option settings are:

- **Automatically create a backup copy at startup** – This runs the auto backup routine and prompts for permission to backup the workbook file when you open the Total Cost model.
- **Display this workbook in Full Screen mode** – This toggles between full screen and standard view. Full Screen mode may hide your favorite toolbar and the worksheets tabs so be aware that navigating through the workbook may be cumbersome when using this setting.
- **Show the splash dialog during startup** – Prevents the Splash dialog box with the disclaimer from appearing when you open the Total Cost model workbook.

- **Show Row and Column headings** – This setting toggles the display of the row and column headings, which may give some much needed additional viewing area on a small display.
- **Set the Screen display (ZOOM)** – The values entered here either increase or decrease the display size of the workbook view, and must be between 10 and 400 and. The values are percentages, so a value of 100 is twice the size of 50.

Resetting the Model

In order to reuse the Total Cost model or "zero-out" all of the settings of a proposed system, a user could simply edit the values themselves. However, this can be time consuming, and, leaving only one orphaned cost variable could skew the results of the next analysis, providing erroneous data. To facilitate the accurate resetting of the values within the workbook several routines are available. All of these procedures will reset the input values of the selected component changing the current values to the default values, either zero or a blank (an empty cell.) The only exception to this is that the value of the analysis period is set to five as its default value. Resetting the Common Data worksheet will remove all of the Alternative System Costs worksheets since they are the product of having a Proposed System name entered on that sheet.

To reset the entire model, or just the Common Data or Proposed System worksheets click the "Reset Values…" button at the top of the input area of the Common Data worksheet or from the Total Cost menu. This will display the "Reset User supplied data" dialog box (see Figure 14.) From this form, you can select to reset:

- **The Common Data Only** – resets only the data on the Common Data worksheet
- **All System Data** – resets the Proposed System worksheets data to zero and removes any user-defined fields.
- **All Model Input Data** – this will reset ALL of the data in the Cost model to their defaults.
- **Show the splash dialog during startup...** – toggles the display of the Total Cost Model's Splash dialog that is shown while the workbook loads.

Each of the active Alternative System worksheets also has a button located at the top of its input area that will initiate a resetting of the current worksheet only. There are confirmation dialog boxes to protect a user from inadvertently deleting input data during these processes however; they should still be performed with caution, as any deletion is permanent.

Figure 14 - Reset the model data dialog box

Conclusion

The research presented in this paper shows F/OSS to be a viable alternative to proprietary or closed source software systems. Additionally, the Total Cost model is proffered as a valuable tool with which to evaluate its true cost to the organization. Approaching any investment question with preconceived opinions unjustly skews the analysis and renders it worthless. Previously mentioned research points out that no model can single-handedly be used to justify an IT system investment, and that both intuitive knowledge of the situation and business case studies are required to present a comprehensive set of decision variables. Properly used financial models only provide a limited set of metrics that should be evaluated by management within their decision process as a whole. Capital investments generally involve a complex set of issues and the Total Cost model can be a valuable tool in reducing this complexity.

The research reviewed for this study noted that F/OSS provides numerous advantages over proprietary or COTS including, but not limited to:

- Inexpensive or free software – no license fee.
- No license management costs or license related legal risks.
- Less dependence on a specific product or vendor.
- Users are not bound to the strategic decisions made by a vendor.
- Increased competition within software markets reduce costs and increase quality.
- Improved security, as source code can be verified and certified by outside sources.

- Open standards and formats
- Evolving software, user supported and user driven.
- Rapid release rate; fixes, patches and new versions are released often
- Immense and highly motivated programming expertise provided by Internet connectivity.

F/OSS does have weaknesses that should be weighed against its strong points in any balanced IT investment case. These drawbacks include:

- Potential need for extensive migration to replace proprietary systems.
- May require in-house development and maintenance skills within the enterprise.
- Product selection could be difficult (limited marketing).
- No brand name assurance.
- Resistance among decision makers, IT personnel and end-users (fear of change).
- F/OSS licenses do not permit ownership.
- More highly educated or skilled administrators may be needed to maintain F/OSS.

F/OSS has a history that promotes both innovation and cooperation. It is consistently used in demanding and evolutionary environments such as the infrastructure of the Internet. Specific applications may or may not be appropriate for a given task or organization, however they are worth considering for any software implementation.

For critics of spreadsheet based financial models, it should be noted research has revealed that possibly twenty to forty percent of all spreadsheets contain some type of

error. This has been corroborated by other earlier studies cited by Panko. One such study suggests that the error rate is up to ninety percent on spreadsheets that are larger than 150 rows of data. However, an increase in the number of errors found by both developers, consultants or researchers, may be due to better methods of auditing, or is possibly due to increasing end-user development of complex spreadsheets. (Panko, 1998)

Such errors should not apply to the Total Cost model since it does not rely on overly complex and customized formulas, and the data range of each worksheet is well below 50 rows. All of the model's calculations are available to view and no programming is hidden from the user. So, as with F/OSS, the source code of the Total Cost model is verifiable; and it is free, as in speech, not as in beer.

A striking aspect of this work with F/OSS and cost analyses is that the cost model was created using a market dominating closed source spreadsheet application, Microsoft Excel. It may even seem ironic that Microsoft's own product could be used to reveal it's own uncompetitive costs (should this be the case.) But, Excel is the most widely used spreadsheet application and thus provides a greater set of potential users.

Future work in the area of software TCO should include the creation of a model using a F/OSS application such as OpenOfice.org's *Calc* or the GNU applications *KSpread* or *Gnumeric*. This would allow organizations to perform a cost analysis without having to purchase any software. Another possible project would be to construct a web-based cost model using PHP, Perl and MySQL. The Total Cost Model could also be revised to produce a more generic tool that would be applicable to any capital investment analysis. Regardless of any new facade given to the model, variations and improvements are always be possible.

Endnotes

[1] The term *hacker* is not a pejorative word to computer scientists. It means an enthusiastic programmer, not one who writes malicious code. For a complete definition see: http://info.astrian.net/jargon/terms/h/hacker.html

[2] GNU is a recursive acronym for “GNU's Not UNIX”; it is pronounced "guh-NEW”.

[3] fetchmail is remote-mail retrieval and forwarding utility program maintained by Eric S. Raymond. The fetchmail homepage is found at: http://www.catb.org/~esr/fetchmail

[4] International Data Corporation (IDC) is a worldwide research and market intelligence company. The IDC homepage is: http://www.idc.com

[5] RedHat Software is a publicly traded distributor of Linux and Linux related products. The RedHat homepage is: http://www.redhat.com

[6] PostgreSQL is an OSS Database Management System, its homepage is: http://www.postgresql.org

[7] Porting is the rewriting or recompiling of a program to enable it to run under a different operating system or hardware platform.

[8] Minix information, including how to download a copy of the software can be found at the Minix homepage: http://www.disi.unige.it/person/DoderoG/minix/info-20.htm

[9] Suse Linux is a German distributor of Linux and Linux related products. Their homepage is: http://www.suse.com/us/index.html

[10] Mandrakesoft is the French vendor of the Mandrake distribution of the Linux operating system and Linux related products and services. Their homepage is: http://www.mandrakelinux.com/en

[11] O'Reilly & Associates, Inc is an extensive source for F/OSS books, conferences and other learning venues. Their homepage is: http://www.oreilly.com

[12] NetBSD is a product of the NetBSD Project. The software is licensed under the BSD License and is OSD compatible. The NetBSD Project homepage is: http://www.netbsd.org

[13] The Gartner Group is an international IT research and consulting firm. The company's homepage is: http://www.gartner.com

[14] The ***root*** user is a UNIX term for the highest-level user; it is comparable to an administrator on a Windows system. The root has no restrictions on its activity on a system, and therefore is rarely used. Most system administration is performed by users with specific privileges that are less than those of the root user.

[15] Forrester Research, Inc is an IT business research and consulting firm, of which the GIGA Information Group is a wholly owned subsidiary. The company's homepage is: http://www.forrester.com

[16] The SCO Group (SCO) is a software and enterprise solution provider. In March 2003, they filed a lawsuit against IBM concerning the illegal use of their UNIX intellectual-property (source code and algorithms) within the Linux operating system. The property in question is source code allegedly provided by IBM to the Linux project. The homepages of these two corporations are: http://www.sco.com and 9+8 http://www.ibm.com/us

[17] Giga Information Group is a subsidiary of Forrester Research, Inc., and provides technology investment decision support tools to companies and institutions. The company's homepage is: http://www.gigaweb.com/homepage/

[18] Standish Group International is an international IT research and consulting firm. The company's homepage is: http://www.standishgroup.com/

Appendix A - The GNU General Public License

Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

PREAMBLE

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead). You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in

effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program.) Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy

of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of

that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED

ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Source URL:http://www.fsf.org/copyleft/gpl.html

Appendix B - The Open Source Definition

Version 1.9

The indented, italicized sections below appear as annotations to the Open Source Definition (OSD) and are ***not*** *a part of the OSD.*

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

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. The License Must Not Restrict 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. Software linked with GPLed libraries only inherits the GPL if it forms a single work, not any software with which they are merely distributed.

*10. The License must be technology-neutral

No provision of the license may be predicated on any individual technology or style of interface.

Rationale: *This provision is aimed specifically at licenses which require an explicit gesture of assent in order to establish a contract between licensor and licensee. Provisions mandating so-called "click-wrap" may conflict with important methods of software distribution such as FTP download, CD-ROM anthologies, and web mirroring; such provisions may also hinder code re-use. Conformant licenses must allow for the possibility that (a) redistribution of the software will take place over non-Web channels that do not support click-wrapping of the download, and that (b) the covered code (or re-used portions of covered code) may run in a non-GUI environment that cannot support popup dialogues.*

Source URL: http://www.opensource.org/docs/definition.phpAppendix

Appendix C - Linus Introduces Linux

This appendix presents two “historic” newsgroup postings from Linus Torvalds, the creator of the Linux operating system. Note that the grammar, punctuation and spelling have not been altered and remain as retrieved from the source URL. It is implied by the source that these are the original messages in their original format. (Hasan, 2002)

On August 25, 1991 Linus Torvalds posted the following message to the MINIX news group (comp.os.minix), many see this as the “birth” of the Linux operating system.

```
From:  torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups:  comp.os.minix
Subject:  What would you like to see most in minix?
Summary:  small poll for my new operating system
Message-ID:  <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date:  25 Aug 91 20:57:08 GMT
Organization:  University of Helsinki

Hello everybody out there using minix -
I'm doing a (free) operating system (just a hobby, won't be big
and professional like gnu) for 386(486) AT clones. This has been
brewing since april, and is starting to get ready. I'd like any
feedback on things people like/dislike in minix, as my OS
resembles it somewhat(same physical layout of the file-system
(due to practical reasons)among other things). I've currently
ported bash(1.08) and gcc(1.40), and things seem to work. This
implies that I'll get something practical within a few months,
and I'd like to know what features most people would want. Any
suggestions are welcome, but I won't promise I'll implement them
:-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-
threaded fs. It is NOT protable (uses 386 task switching etc),
and it probably never will support anything other than AT-
harddisks, as that's all I have :-(.
```

Linus released Linux, version 0.01, in September 1991. After some enthusiastic collaboration with other programmers via the Internet, version 0.02 was released on October 5th with the following newsgroup posting:

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: Free minix-like kernel sources for 386-AT
Message-ID: <1991Oct5.054106.4647@klaava.Helsinki.FI>
Date: 5 Oct 91 05:41:06 GMT
Organization: University of Helsinki
```

Do you pine for the nice days of minix-1.1, when men were men and wrote their own device drivers? Are you without a nice project and just dying to cut your teeth on a OS you can try to modify for your needs? Are you finding it frustrating when everything works on minix? No more all-nighters to get a nifty program working? Then this post might be just for you :-)

As I mentioned a month(?) ago, I'm working on a free version of a minix-lookalike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be depending on what you want), and I am willing to put out the sources for wider distribution. It is just version 0.02 (+1 (very small) patch already), but I've successfully run bash/gcc/gnu-make/gnu-sed/compress etc under it.

Sources for this pet project of mine can be found at nic.funet.fi (128.214.6.100) in the directory /pub/OS/Linux. The directory also contains some README-file and a couple of binaries to work under linux (bash, update and gcc, what more can you ask for :-). Full kernel source is provided, as no minix code has been used. Library sources are only partially free, so that cannot be distributed currently. The system is able to compile "as-is" and has been known to work. Heh. Sources to the binaries (bash and gcc) can be found at the same place in /pub/gnu.

Source URL:
http://ragib.hypermart.net/linux/#In%20The%20Beginning

Appendix D - Examples of Free/Open Source Software

As of May 18, 2004 there were 3,153 free software packages indexed in the complete directory located at: http://www.gnu.org/directory

Some of the programs in this listing are not GNU applications, but do use one of the licenses approved by the Open Source Initiative: http://www.opensource.org/licenses

Program category	***Description***	***License***
Database	**MySQL** – a Relational database management system that supports the standardized Structured Query Language.	GPL / Dual License
	PostgreSQL – an Object-Relational DBMS	BSD License
Graphics	**The GIMP** – The GNU Image Manipulation Program is a photo retouching and graphics-authoring program.	GPL
	ImageMagick - Image display and manipulation program.	Freely Redistributable License
Productivity	**Gnucash** - Personal and small business money-management software.	GPL
	Evolution - GNOME mail client, calendar, contact manager, and PIM.	GPL
	KOffice – an integrated office productivity suite, that includes a word processor, spreadsheet, presentation and drawing applications and several other advanced programs. Native to the KDE desktop environment.	GPL
	Gnumeric - Math and spreadsheet program feature compatible with commercial applications.	GPL
	OpenOffice.org – a multi-platform office productivity suite providing a word processor, spreadsheet, presentation manager, and drawing program.	GPL / Dual License

Program category	***Description***	***License***
Internet	**Apache** – an HTTP/1.1 compliant web server	Apache Software License
	Mozilla - a graphical web browser that is being developed by volunteers with the cooperation and support of Netscape.	Netscape Public License (NPL) or the Mozilla Public License (MPL)
Programming	**GCC** - The GNU Compiler Collection is a full-featured ANSI C compiler supporting C, C++, Java, and Fortran.	GPL
	Python - an interpreted, object-oriented programming language.	Python License, compatible with the GPL
	Perl - a high-level, general-purpose scripting language.	The Artistic license
	PHP - a general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. In many ways like Microsoft's Active Server Pages (ASP).	The PHP License
Editors	**AbiWord** - Word processor from the AbiSource Project desktop suite.	GPL
	Emacs - Emacs is the extensible, customizable, self-documenting real-time display editor.	GPL
	Bluefish - an HTML editor with support for many scripting languages.	GPL
System	**Linux** - a UNIX-like operating system, which is properly named "GNU/Linux”.	GPL
	FreeBSD - a multi-platform operating system derived from BSD UNIX developed at the University of California, Berkeley.	
	SAMBA – SMB file-sharing implementation for Windows, OS/2, NFS, or Netware servers.	GPL

Program category	***Description***	***License***
Utility	**Info-Zip** (**UnZip** and **Zip**) – Compression utilities whose code has been used in other programs, such as Pretty Good Privacy (PGP) and WinZip.	A Modified BSD License

Sources:

FSF/UNESCO Free Software Directory: Boston, MA; Retrieved on August 30, 2003 from:
http://www.gnu.org/directory/

Info-Zip; Retrieved on April 29, 2003 from:
http://www.info-zip.org/pub/infozip/

Linux is a Registered Trademark of Linus Torvalds

Appendix E - SCO Group letter to IBM

The SCO Group, which claims ownership of the UNIX operating system's intellectual properties, purportedly sent the following letter to all of its Linux customers. At one time it was available for public viewing on the SCO website (http://www.sco.com/scosource/letter_to_linux_customers.html), but has since been removed. The letter shown here is addressed to IBM, whom the SCO Group actually sued.

May 12. 2003

Mr. Lucio A. Noto
Audit Committee Chair
International Business Machines Corporation
New Orchard Road
Armonk, NY 10504

Dear Lucio:

SCO holds the rights to the UNIX operating system software originally licensed by AT&T to approximately 6,000 companies and institutions worldwide (the "UNIX Licenses"). The vast majority of UNIX software used in enterprise applications today is a derivative work of the software originally distributed under our UNIX Licenses. Like you, we have an obligation to our shareholders to protect our intellectual property and other valuable rights.

In recent years, a UNIX-like operating system has emerged and has been distributed in the enterprise marketplace by various software vendors. This system is called Linux. We believe that Linux is, in material part, an unauthorized derivative of UNIX.

As you may know, the development process for Linux has differed substantially from the development process for other enterprise operating systems. Commercial software is built by carefully selected and screened teams of programmers working to build proprietary, secure software. This process is designed to monitor the security and ownership of intellectual property rights associated with the code.

By contrast, much of Linux has been built from contributions by numerous unrelated and unknown software developers, each contributing a small section of code. There is no mechanism inherent in the Linux development process to assure that intellectual property rights,

confidentiality or security are protected. The Linux process does not prevent inclusion of code that has been stolen outright; or developed by improper use of proprietary methods and concepts.

Many Linux contributors were originally UNIX developers who had access to UNIX source code distributed by AT&T and were subject to confidentiality agreements, including confidentiality of the methods and concepts involved in software design. We have evidence that portions of UNIX System V software code have been copied into Linux and that additional other portions of UNIX System V software code have been modified and copied into Linux, seemingly for the purposes of obfuscating their original source.

As a consequence of Linux's unrestricted authoring process, it is not surprising that Linux distributors do not warrant the legal integrity of the Linux code provided to customers. Therefore legal liability that may arise from the Linux developments process may also rest with the end user.

We believe that Linux infringes on our UNIX intellectual property and other rights. We intend to aggressively protect and enforce these rights. Consistent with this effort, on March 7, we initiated legal action against IBM for alleged unfair competition and breach of contract with respect to our UNIX rights. This case is pending in Utah Federal District Court. As you are aware, this case has been widely reported and commented upon in the press. If you would like additional information, a copy of the complaint and response may be viewed at our web site at www.sco.com/scosource.

For the reasons explained above, we have also announced the suspension of our own Linux-related activities until the issues surrounding Linux intellectual property and the attendant risks are better understood and properly resolved.

Similar to analogous efforts underway in the music industry, we are prepared to take all actions necessary to stop the ongoing violation of our intellectual property or other rights.

SCO's actions may prove unpopular with those who wish to advance or otherwise benefit from Linux as a free software system for use in enterprise applications. However, our property and contract rights are important and valuable: not only to us, but to every individual and every company whose livelihood depends on the continued viability of intellectual and intangible property rights in a digital age.

Yours truly,

THE SCO GROUP

By: Darl McBride
President and CEO

Source URL:http://www.groklaw.net/article.php?story=20031127100124265

Appendix F - Homogenous SWOT model of Free/Open Source Software

Software Strengths • Inexpensive or free software – no license fee. • F/OSS is freely distributable, no penalty for making copies. • No license management or license related legal risks. • Less dependence on a specific product or vendor. • Increased competition within software markets reduce costs and increase quality. • Increased security - source code can be verified and certified. • Open standards and formats reduce incompatibilities. • Modular, customizable software, users choose modules and can customize to suit. • Evolving software, user supported and user driven. • Rapid release rate; fixes, patches and new versions updated frequently. • Parallel debugging/development. • 24 x 7 technical support via the Internet (newsgroups, mailing lists, etc.). • Software viability un affected by vendor's business condition. • Not bound to the strategic decisions of a vendor. • Quality of support increases while the end-user cost of support decreases. • Immense programming expertise. • Large independent distribution base. • Corporate support for development, distribution and consultancy services. • Code re-use reduces development time and provides predictable results	**Weaknesses** • Potential need for extensive migration to replace proprietary systems. • May require in-house development skills and maintenance within the enterprise. • Product selection could be difficult (no marketing departments). • Possible interoperability problems with proprietary software formats or standards • No brand name to use as a guarantee. • Resistance among decision makers, IT personnel and end-users (Fear of Change). • F/OSS licenses do not permit ownership. • More highly educated or skilled administrators needed to maintain F/OSS. • No GUI or weak GUIs are the norm as many F/OSS programmers prefer command line interfaces. • Risk of code forking, which ultimately produces different version of the same product and thus demands a decision on which "fork" to accept.
Opportunities • Easy entry into software markets with an existing dominate product or vendor. • "Going Open" gives vendors the ability to dramatically increase the developer base of existing closed source programs. • Closed source software that is released to the F/OSS community gains support of not only developers, but end-users and independent vendors. • Governments require sizeable software installations on limited budgets • Opportunities are greater with poorer governments (Third World) with developing school systems and agencies, Proprietary or COTS license fees are not within any budget.	**Threats** • Disinterest in a specific project can stall or end active development. • Initiation of new F/OSS projects may appropriate existing project participants and stall or end active development. • Hybrid open-source projects that build on true F/OSS and are released as commercial ventures. • Organized FUD (Fear, Uncertainty and Doubt) marketing campaigns by commercial software companies with "unlimited" budgets. • Lawsuits involving "borrowed" code (e.g. SCO) or vague software patents. • Governmental or enterprise security requirements (or certification). • Version control amongst integrated or compatible programs cannot always be assured. • Customized F/OSS programs can fragment or fork the existing project.

Appendix G - Excel Workbook Formulas used in the Total Cost Model

Formula references to “System_1” refer to the value of *system 1* as defined on the Common Data worksheet. Formulas for each system use the named ranges, *System_1*, *System_2* and *System_3* respectively. All functions used in the following formulas are standard Excel functions with the exception of the two custom Payback Period functions listed below (written by the author and displayed as function signatures with comments):

- **SPayBack** (CumFlow As Range, NetFlow As Range, period As Integer) As Variant - takes the required arguments and returns the SIMPLE Payback Period in years
- **DPayBack** (CumFlow As Range, NetFlow As Range, drate As Double, period As Integer) As Variant - takes the required arguments of Discounted cashflows and returns the DISCOUNTED Payback Period in years

Metric	**Spreadsheet Formula (Excel 2003)**
Total Cost of Ownership (TCO)	=IF(ISBLANK(System_1),"",IF($C$10<0,ABS($C$10),"N/A"))
Simple Return on Investment (ROI)	=IF(ISBLANK(System_1),"",IF(AND(OR(Period={1,3,4,5}),ISNUMBER((SUM($C$14:OFFSET($C$14,0,0,1,Period))+SUM($C$13:OFFSET($C$13,0,0,1, Period)))/ABS(SUM($C$13:OFFSET($C$13,0,0,1,Period))))),(SUM($C$14:OFFSE T($C$14,0,0,1,Period))+SUM($C$13:OFFSET($C$13,0,0,1,Period))) /ABS(SUM($C$13:OFFSET($C$13,0,0,1,Period))),"N/A"))

Metric	**Spreadsheet Formula (Excel 2003)**
Discounted Return on Investment (ROI)	=IF(ISBLANK(System_1),"",IF(AND(OR(Period={1,3,4,5}),ISNUMBER((SUM($M$12:OFFSET($M$12,0,0,Period,1))+SUM($M$21:OFFSET($M$21,0,0, Period,1)))/ABS(SUM($M$21:OFFSET($M$21,0,0,Period,1))))), (SUM($M$12:OFFSET($M$12,0,0,Period,1))+SUM($M$21:OFFSET($M$21,0,0, Period,1)))/ABS(SUM($M$21:OFFSET($M$21,0,0,Period,1))),"N/A"))
Payback Period	=IF(ISBLANK(System_1),"",IF($H$13=0,"N/A",SPayBack($C$15:$G$15,$C$16:$ G$16,Period)))
Discounted Payback Period (years)	=IF(ISBLANK(System_1),"",IF($H$13=0,"N/A",DPayBack($M$30:$M$34,$M$39: $M$43,Cost_of_Capital,Period)))
Internal Rate of Return (IRR)	=IF(ISBLANK(System_1),"",IF(AND(OR(Period={1,3,4,5}),ISNUMBER(IRR (OFFSET($M$46,0,0,Period,1),-0.9))), IRR(OFFSET($M$46,0,0,Period,1),- 0.9), "N/A"))
Modified Internal Rate of Return (MIRR)	=IF(ISBLANK(System_1),"",IF(AND(OR(Period={1,3,4,5}),ISNUMBER(MIRR(OFFSET($M$46,0,0,Period,1),Reinvest_Rate,Cost_of_Capital))),MIRR(OFF SET($M$46,0,0,Period,1),Reinvest_Rate,Cost_of_Capital),"N/A"))
Net Present Value (NPV)	=IF(ISBLANK(System_1),"",IF($H$13=0,"N/A",IF(AND(OR(Period={1,3,4,5}), ISNUMBER(NPV(Cost_of_Capital,(OFFSET($M$46,0,0,Period,1))))), (NPV(Cost_of_Capital,(OFFSET($M$46,0,0,Period,1)))*(1+Cost_of_Capital)),"N/A")))
NPV - Sum of Discounted Net Cash flows	=IF(ISBLANK(System_1),"",IF($H$13=0,"N/A",SUM($M$39:$M$43)))

Metric	**Spreadsheet Formula (Excel 2003)**
Profitability Index (PI)	=IF(ISBLANK(System_1),"",IF(AND(OR(Period={1,3,4,5}),ISNUMBER((SUM($M$12:OFFSET($M$12,0,0,Period,1))/(-1*SUM($M$21:OFFSET($M$21,0,0,Period,1)))))),(SUM($M$12:OFFSET($M$12,0,0,Period,1))/(-1*SUM($M$21:OFFSET($M$21,0,0,Period,1)))),"N/A"))

Appendix H - Visual Basic for Applications programming examples

The first code example is triggered to run when a particular cell's value is changed. The cell address is trapped by the Select Case statements and the code specific to that case is initiated. This event procedure is shown as it displays various VBA functions, Excel functions and the use of custom programmed functions (such as Hide_years and Refresh_Charts).

The second code example is of the DPayBack custom function. This function calculates the discounted Payback period and returns that value, or "N/A" as its result.

```
Private Sub Worksheet_Change(ByVal Target As Range)
'   this sub will change the name of the alternative system worksheets
'   or add or delete their respective sheets as needed

On Error Resume Next

Dim intErrorCount As Integer    '   a counter for the errHandler

Application.EnableEvents = False    '   turn OFF event tracking
Application.DisplayAlerts = False   '   hide any error or confirmation messages
Application.ScreenUpdating = False  '   turn OFF screen updates or refreshes

    Select Case Target.Address '    these addresses must be absolute addresses

    Case "$D$3" '   the Period has changed!
         '   hide the system year columns based on the value of Period
        Call Hide_years(Sheet_1.Range("Period"))
        Call Refresh_Charts(Sheet_1.Range("Period"))

    Case "$D$8", "$D$9" '   The "Projected Yearly Growth Rate" value if this changes we alter the
formulas
```

```
        If Not IsEmpty(Sheet_1.Range("D8").Value) And Sheet_1.Range("D9").Value > 0 Then
            Sheet_1.Range("Total_Year_2").Formula =
"=IF(Period>=3,IF(Growth_Rate>0,IF(Total_Year_1>0,(Total_Year_1*(1+Growth_Rate)),0),Total_Year_1),""N/
A"")"
            Sheet_1.Range("Total_Year_3").Formula =
"=IF(Period>=3,IF(Growth_Rate>0,IF(Total_Year_1>0,(Total_Year_2*(1+Growth_Rate)),0),Total_Year_1),""N/
A"")"
            Sheet_1.Range("Total_Year_4").Formula =
"=IF(Period>3,IF(Growth_Rate>0,IF(Total_Year_1>0,(Total_Year_3*(1+Growth_Rate)),0),Total_Year_1),""N/A
"")"
            Sheet_1.Range("Total_Year_5").Formula =
"=IF(Period>4,IF(Growth_Rate>0,IF(Total_Year_1>0,(Total_Year_4*(1+Growth_Rate)),0),Total_Year_1),""N/A
"")"
        Else
            Sheet_1.Range("Total_Year_2").Formula = "0"
            Sheet_1.Range("Total_Year_3").Formula = "0"
            If Sheet_1.Range("Period").Value > 3 Then
                Sheet_1.Range("Total_Year_4").Formula = "0"
            Else
                Sheet_1.Range("Total_Year_4").Formula = "N/A"
            End If
            If Sheet_1.Range("Period").Value > 4 Then
                Sheet_1.Range("Total_Year_5").Formula = "0"
            Else
                Sheet_1.Range("Total_Year_5").Formula = "N/A"
            End If
        End If

    Case "$B$16"    '    System 1
        If Sheet_1.Range("B16").Value <> "" Then
            Worksheets(2).Visible = True
            Worksheets(2).Name = Sheet_1.Range("B16").Value & " - System 1"
            ThisWorkbook.CustomDocumentProperties.Item("System_1").Value = Worksheets(2).Name
            Sheet_1.Range("C16").Activate
            Sheet_1.Range("C16").Select
        Else
            If Sheet_1.Range("C16").Value <> "" Then
                MsgBox "Please enter a name for the FIRST Proposed System!", vbOKOnly, "First system
not defined!"
                Application.Undo
```

```
            ElseIf Worksheets(2).Visible = True Then
                '   do you really want to close the sheet and rest the data to ZERO?
                If MsgBox("Do you want to close the System 1 worksheet and RESET all of it's data?", _
                vbYesNo, "Close the System 1 worksheet?") = vbYes Then
                    Worksheets(2).Name = "System_1"
                    '   call the reset function here!
                    Call Reset_Systems(2)
                    Call Refresh_Charts(Sheet_1.Range("Period"))
                    Worksheets(2).Visible = False
                    ThisWorkbook.CustomDocumentProperties.Item("System_1").Value =
Sheet_1.Range("B16").Value
                Else ' put the name back in the cell
                    Application.Undo
                End If
            End If

            Sheet_1.Range("B16").Activate
            Sheet_1.Range("B16").Select
        End If

    Case "$C$16"    '    System 2

        ' if there isn't a value in the first cell
        If Sheet_1.Range("B16").Value = "" Then
            MsgBox "Please enter a name for the FIRST Proposed System!", vbOKOnly, "First system not
defined!"
            Sheet_1.Range("C16").Value = ""
            Sheet_1.Range("B16").Activate
            Sheet_1.Range("B16").Select
        ElseIf Sheet_1.Range("C16").Value <> "" Then
            Worksheets(3).Visible = True
            Worksheets(3).Name = Sheet_1.Range("C16").Value & " - System 2"
            ThisWorkbook.CustomDocumentProperties.Item("System_2").Value = Worksheets(3).Name
            Sheet_1.Range("D16").Activate
            Sheet_1.Range("D16").Select
        ElseIf Sheet_1.Range("C16").Value = "" Then
            If Worksheets(3).Visible = True Then
             '   do you really want to close the sheet and rest the data to ZERO?
                If MsgBox("Do you want to close the System 2 worksheet and RESET all of it's data?", _
                vbYesNo, "Close the System 2 worksheet?") = vbYes Then
```

```
                    Worksheets(3).Name = "System_2"
                    '   call the reset function here!
                    Call Reset_Systems(3)
                    Call Refresh_Charts(Sheet_1.Range("Period"))
                    Worksheets("System_2").Visible = False
                    ThisWorkbook.CustomDocumentProperties.Item("System_2").Value =
Sheet_1.Range("C16").Value
                Else ' put the name back in the cell
                    Application.Undo
                End If

                Sheet_1.Range("C16").Activate
                Sheet_1.Range("C16").Select
            End If
        End If

    Case "$D$16"    '    System 3

        If Sheet_1.Range("B16").Value = "" Then  ' if there isn't a value in the first cell
            MsgBox "Please enter a name for the FIRST Proposed System!", vbOKOnly, "Second system not
defined!"
            Sheet_1.Range("D16").Value = ""
            Sheet_1.Range("B16").Activate
            Sheet_1.Range("B16").Select
        ElseIf Sheet_1.Range("C16").Value = "" Then   ' if there isn't a value in the second cell
            MsgBox "Please enter a name for the SECOND Proposed System!", vbOKOnly, "Second system not
defined!"
            Sheet_1.Range("D16").Value = ""
            Sheet_1.Range("C16").Activate
            Sheet_1.Range("C16").Select
        ElseIf Sheet_1.Range("D16").Value <> "" Then
            Worksheets(4).Visible = True
            Worksheets(4).Name = Sheet_1.Range("D16").Value & " - System 3"
            ThisWorkbook.CustomDocumentProperties.Item(3).Value= Worksheets(4).Name
            Sheet_1.Range("D16").Activate
            Sheet_1.Range("D16").Select
        ElseIf Sheet_1.Range("D16").Value = "" Then
            If Worksheets(3).Visible = True Then
             '   do you really want to close the sheet and rest the data to ZERO?
                If MsgBox("Do you want to close the System 3 worksheet and RESET all of it's data?", _
```

```
                vbYesNo, "Close the System 3 worksheet?") = vbYes Then
                    Worksheets(4).Name = "System_3"
                    '   call the reset function here!
                    Call Reset_Systems(4)
                    Call Refresh_Charts(Sheet_1.Range("Period"))
                    Worksheets("System_3").Visible = False
                    ThisWorkbook.CustomDocumentProperties.Item("System_3").Value=
Sheet_1.Range("D16").Value
                Else ' put the name back in the cell
                    Application.Undo
                End If
                Sheet_1.Range("D16").Activate
                Sheet_1.Range("D16").Select
            End If
        End If

    End Select

    Application.ScreenUpdating = True   ' turn back on screen updates or refreshes
    Application.DisplayAlerts = True    ' turn back on any error or confirmation messages
    Application.EnableEvents = True     ' turn back on event tracking

End Sub
```

The next code example is of the DPayBack custom function. This function calculates the discounted Payback period and returns it, or “N/A” as its result.

```
Public Function DPayBack(CumFlow As Range, NetFlow As Range, drate As Double, period As Integer) As Variant
'   take the required arguments of Discounted cashflows and return the DISCOUNTED Payback Period in years

Application.Volatile True

If CumFlow(1) > 0 Then
    DPayBack = 1
ElseIf CumFlow(2) > 0 And period > 1 Then
    DPayBack = 1 + (Abs(CumFlow(1)) / NetFlow(2))
ElseIf CumFlow(3) > 0 And period > 1 Then
    DPayBack = 2 + (Abs(CumFlow(2)) / NetFlow(3))
ElseIf CumFlow(4) > 0 And period >= 4 Then
    DPayBack = 3 + (Abs(CumFlow(3)) / NetFlow(4))
ElseIf CumFlow(5) > 0 And period = 5 Then
    DPayBack = 4 + (Abs(CumFlow(4)) / NetFlow(5))
Else
    DPayBack = "N/A"
End If

End Function
```

References

Anonymous[1]; 1992; *Preparing Financial Models*; American Institute of Certified Public Accountants, Inc.; New York, New York

Anonymous[2]; 2002; *Linux vs. Windows Total Cost of Ownership Comparison*; Cybersource Pty, Ltd.; retrieved from the Internet on 10/6/2003 from: http://www.cyber.com.au/cyber/about/linux_vs_windows_tco_comparison.pdf

Anonymous[3]; 2003; *Financial Justification*; Solution Matrix, Ltd.; retrieved from the Internet on 10/3/2003 from: http://www.solutionmatrix.com/fjgo.html

Anonymous[4]; April 3, 2003; *Gnu's Not UNIX!*; Free Software Foundation, Inc.; Boston, MA; retrieved from the Internet on 3/6/2003 from: http://www.fsf.org/

Anonymous[5]; *Beyond the Price Tag*; International Business Machines Corporation; retrieved from the Internet on 11/7/2003 from: http://www-1.ibm.com/mediumbusiness/sbcus/rc/featurearticle.jsp?id=4590

Anonymous[6]; December 29, 2002; *Categories of Free and Non-Free Software*, Free Software Foundation, Inc; Boston, MA; retrieved from the Internet on 3/21/2003 from: http://www.fsf.org/philosophy/categories.html#TheGNUsystem

Anonymous[7]; February 2000; *How Open Source Software Development Works*; LinuxCare, Inc.; San Francisco, CA; retrieved from the Internet on 3/21/2003 from: http://www.itpapers.com/ (logon required)

Anonymous[8]; January 2001; *Innovation Without Incompatibility*; LinuxCare, Inc.; San Francisco, CA retrieved from the Internet on 3/21/2003 from: http://www.itpapers.com/ (logon required)

Anonymous[9]; June 2002; *Free/Libre and Open Source Software: Survey and Study*; Berlecon Research GmbH; Berlin and the International Institute of Infonomics; retrieved from the Internet from: http://www.infonomics.nl/FLOSS/report/

Anonymous[10]; June 4, 2003, last revision; *Financial Metrics for the Business Case*, (Excel Spreadsheet); Solution Matrix Ltd; Boston, MA; retrieved from the Internet on 10/16/2003 from: http://www.solutionmatrix.com/downloads/FinancialMetrics/FinancialMetrics.zip

Anonymous[11]; March 7, 2003; *The Free Software Definition*; Free Software Foundation, Inc; Boston, MA; retrieved from the Internet on 3/21/2003 from: http://www.fsf.org/philosophy/free-sw.html

Anonymous[12]; May, 1999; *Cost-Benefit Analysis Guide For NIH IT Projects*; National Institutes of Health, United States Department Of Health And Human Services; retrieved from the Internet on 10/3/2003 from: http://irm.cit.nih.gov/itmra/cbaguide.html

Anonymous[13]; no date; *Benefits of Using Open Source Software*; GBdirect, Ltd, West Yorkshire, United Kingdom; retrieved from the Internet on 4/21/2003 from: http://ebusiness.gbdirect.co.uk/OpenSourceMigration/benefit.html

Anonymous[14]; no date; *History of the OSI*; Open Source Initiative (OSI); Copyright 2003; retrieved from the Internet on 4/10/2003 from: http://www.opensource.org/docs/history.php

Anonymous[15]; no date; *Jobs for Hackers: Yes, You Can Eat Open Source*; Open Source Initiative (OSI); Copyright 2003; retrieved from the Internet on 4/10/2003 from: http://www.opensource.org/advocacy/jobs.php

Anonymous[16]; no date*; Linux distributions*; Linux Online; retrieved from the Internet on April 11, 2003 from: http://www.linux.org/dist/index.html

Anonymous[17]; no date; *MySQL Licensing Policy*; MySQL AB; Uppsala, Sweden; retrieved from the Internet on 4/20/2003 from: http://www.mysql.com/products/licensing.html

Anonymous[18]; no date; *NetCraft Web Server Survey*; Netcraft Ltd; Bath, England; Current data located at: http://www.netcraft.com/survey/

Anonymous[19]; no date; *Open Source Case for Business*; Open Source Initiative (OSI); Copyright 2003; retrieved from the Internet on 4/10/2003 from: http://www.opensource.org/advocacy/case_for_business.php

Anonymous[20]; no date; *Open Source License Information*; Open Source Initiative (OSI); Copyright 2003; retrieved from the Internet on 4/10/2003 from: http://www.opensource.org/licenses/index.php

Anonymous[21]; no date; *PostgreSQL License webpage*; PostgreSQL, Inc; retrieved from the Internet on 3/10/2003 from: http://www.postgresql.org/licence.html

Anonymous[22]; no date; *Products*; Open Source Initiative (OSI); Copyright 2003; retrieved from the Internet on 4/10/2003 from: http://www.opensource.org/docs/products.php

Anonymous[23]; no date; *Shared Source: A Dangerous Virus*; Open Source Initiative (OSI); Copyright 2003; retrieved from the Internet on 4/10/2003 from: http://www.opensource.org/advocacy/shared_source.php

Anonymous[24]; no date; *The Halloween Documents*; Open Source Initiative (OSI); Copyright 2003; retrieved from the Internet on 4/10/2003 from: http://www.opensource.org/halloween

Anonymous[25]; no date; *The Wisdom of Adam Smith*; Adam Smith Institute; retrieved from the Internet on 9/2/2003 from: http://www.adamsmith.org/smith/quotes.htm

Anonymous[26]; no date; *Why "Free" Software is too Ambiguous*; Open Source Initiative (OSI); Copyright 2003; retrieved from the Internet on 4/10/2003 from: http://www.opensource.org/advocacy/free-notfree.php

Anonymous[27]; October 2002; *Open-Source Software In E-Government*; Danish Board of Technology; retrieved from the Internet on 10/3/2003 from: http://www.tekno.dk/pdf/projekter/p03_opensource_paper_english.pdf

Anonymous[28]; no date; *The Open Source Case for Customers*; Open Source Initiative (OSI); Copyright 2003; retrieved from the Internet on 4/10/2003 from: http://www.opensource.org/advocacy/case_for_customers.php

Ante, Spencer E.; Greene, Jay; Hamm, Steve; Kerstetter, Jim; March 3, 2003; *The Linux Uprising*; BusinessWeek; pp 78-84; The McGraw-Hill Companies, Inc.; New York, NY

Bessen, James; July 2002; *Open Source Software: Free Provision of Complex Public Goods*; Research on Innovation; retrieved from the Internet on 4/12/2003 from: http://researchoninnovation.org/opensrc.pdf

Bezroukov, Nikolai; December 1999; *A Second Look at the Cathedral and Bazaar*; First Monday, volume 4, number 12; retrieved from the Internet on 4/20/2003 from: http://firstmonday.org/issues/issue4_12/bezroukov/index.html

Bonaccorsi, Andrea and Rossi, Cristina; February 2002; *Why Open Source Software can Succeed*; Laboratory of Economics and Management, Sant'Anna School of Advanced Studies, Pisa, Italy; retrieved from the Internet on 4/10/2003 from: http://opensource.mit.edu/papers/rp-bonaccorsirossi.pdf

Bozman, Jean; Gillen, Al, Kolodgy, Charles; Kusnetzky, Dan; Perry, Randy; and Shiang, David; October 2002; *Windows 2000 Versus Linux in Enterprise Computing*; IDC; retrieved from the Internet: http://www.microsoft.com/windows2000/migrate/unix/tco.asp

Brooks, Jason; August 2003; *Novell Bets on Linux*; eWeek; p. 33; Ziff-Davis Media; online version available at: http://www.eweek.com/article2/0,3959,1225808,00.asp

Cirillo, Rich; July 15, 2002; *Ballmer: Linux Changed Our Game*; VARBusiness; CMP Media Inc.; retrieved from the Internet on 4/6/2003 from: http://vb.channelsupersearch.com/news/var/36355.asp

Clark, John j., Hindlang, Thomas J. and Pritchard, Richard E.;1984; *Capital Budgeting*; Prentice-Hall, Inc.; Englewood Cliffs, New Jersey

Cohen, Nancy; no date; *Tei And Sympathy*; Open: The Strategic Guide to Open Source; CCI Communications; Newton, MA; retrieved from the Internet on 3/24/2003 from: http://www.open-mag.com/9843483279.htm

Daffara, Carlo and Gonzalez-Barahona, Jesus M., editors; April 2000; *Free Software / Open Source; Information Society Opportunities for Europe?*; European Working Group on Libre Software; retrieved on April 28, 2003 from: httphttp://eu.conecta.it/paper.pdf

Dalle, Jean-Michael and Jullian, Nicolas; January 2001; *Open-Source vs. Proprietary Software*; École Nationale Supérieure of Telecommunications, France; retrieved from the Internet on 4/21/2003 from: http://opensource.mit.edu/papers/dalle2.pdf

DiBona, Chris, Ockman, Sam and Stone, Mark; January 1999; *OPENSOURCES: Voices from the Open Source Revolution*; O'Reilly and Associates; Sebastopol, CA

E-Soft, Inc.; April 2003; *Web Server Survey*; Current data located at: http://www.securityspace.com/s_survey/data/200303/index.html

Farbey, Barbara; Finkelstein, Anthony; May 2001; *Evaluation in Software Engineering: ROI, but More than ROI*; University College London, United Kingdom; Presented at the Third International Workshop on Economics-Driven Software Engineering Research; retrieved from the Internet on 4/20/2003 from: http://www.cs.virginia.edu/~sullivan/edser3/finkelstein.pdf

Festa, Paul; August 29, 2001*; Governments Push Open-Source Software*; CNET Networks, Inc; retrieved on April 12, 2003 from: http://news.com.com/2100-1001-272299.html

Finney, Robert G.; 1994; Every Manager's Guide to Business Finance; American Management Association; New York, New York

Foster, Ed; January 1998; *1997 Best Technical Support Award*; InfoWorld Media Group Inc.; retrieved from the Internet on 4/20/2003 from: http://www.infoworld.com/cgi-bin/displayTC.pl?/97poy.supp.htm

Gillen, Al, and Kusnetsky, Dan; March 2001; *Linux: A Journey into the Enterprise*; IDC; retrieved from the Internet: http://www.nebula.nl/downloads/Linux_in_the_Enterprise.pdf

Greene, Jay; March 3, 2003; *Pecked by Penguins*; BusinessWeek; pp 84-86; The McGraw-Hill Companies, Inc.; New York, NY

Gross, Grant; January 3, 2003; *Linux TCO Edge: Lower labor costs*; TechRepublic; CNET Networks, Inc; retrieved from the Internet on 3/24/2003 from: http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2907876,00.html

Hahn, Robert W., editor; 2002; *Government Policy toward Open Source Software*; AEI-Brookings Joint Center for Regulatory Studies, Washington, D.C; retrieved from the Internet on 9/1/2003 from: http://aei-brookings.org/admin/pdffiles/phpJ6.pdf

Hasan, Ragib; July 24, 2002, last revision; *History of Linux*; retrieved from the Internet on 4/1/2003 from: http://ragib.hypermart.net/linux/

Hawkins, Richard E.; November 2002; *The Economics of Free and Open Source Software*; Pennsylvania State University; retrieved from the Internet on 4/20/2003 from: http://www.personal.psu.edu/faculty/r/e/reh18/research/01.sce/Open_Source.conference.pdf

Hecker, Frank; June 2000; Setting Up Shop:The Business of Open-Source Software retrieved from the Internet: http://www.hecker.org/writings/setting-up-shop.html

Helfert, Erich A.; 1994; *Techniques of Financial Analysis*, 8th ed.; Irwin Professional Publishing; New York, New York.

Herbst, Anthony F.; 1982; *Capital Budgeting: Theory, Quantitative Methods and Applications*; Harper and Row Publishers; New York, New York

Higson, C. J.; 1986*; Business Finance*; Butterworth and Company, Ltd.; London, United Kingdom

Jones, A. Russell; August 13, 2003; *Linux vs. Windows: Choice vs. Usability*; DevX, Jupitermedia Corporation; retrieved from the Internet on 8/17/2003 from: http://www.devx.com/opensource/Article/16969

Kenwood, Carolyn A.; July 2001; *A Business Case Study of Open Source Software*; The Mitre Corporation; Bedford MA.; retrieved from the Internet on 4/16/2003 from: http://www.mitre.org/support/papers/tech_papers_01/kenwood_software/kenwood_software.pdf

Kim, Eugene Eric; April 2003; *An Introduction to Open Source Communities*; Blue Oxen Associates LLC; retrieved from the Internet on 5/20/2003 from:
http://www.blueoxen.org/research/00007/

Krishnamurthy, Sandeep; January 2003; *Open-Source Software*; University of Washington; retrieved from the Internet on 3/20/2003 from:
http://students.washington.edu/borg/case2.htm

Kuwabara, Ko; March 2000; *Linux: A Bazaar at the Edge of Chaos*; First Monday, volume 5, number 3; retrieved from the Internet on 4/20/2003 from:
http: http://firstmonday.org/issues/issue5_3/kuwabara/index.html

Lakhani, Karim R. and von Hippel, Eric; June 2002; *How Open Source Software Works: "free" user-to-user assistance*; MIT Sloan School of Management; retrieved from the Internet on 9/20/2003 from:
http://opensource.mit.edu/papers/lakhanivonhippelusersupport.pdf

Lancashire, David; December 2001; *The Fading Altruism of Open Source Development*; First Monday, volume 6, number 12; retrieved from the Internet on 4/20/2003 from: http://firstmonday.org/issues/issue6_12/lancashire/index.html

Larson, Kermit D. and Miller, Paul B. W.; 1993; *Fundamental Accounting Principles*; 13th edition; Richard D. Irwin, Inc.; Homewood, IL.

Lerner, Josh and Tirole, Jean; December 2000; *The Simple Economics of Open Source*; Harvard Business School; retrieved from the Internet on 4/21/2003 from:
http://opensource.mit.edu/papers/Josh%20Lerner%20and%20Jean%20Triole%20-%20The%20Simple%20Economics%20of%20Open%20Source.pdf

Lumby, Stephen; 1988; *Investment Appraisal & Financing Decisions*, 3rd Ed.; Von Nostrand Reinhold Company, Ltd.; Berkshire, England

Margulius, Dave; August 29, 2003; *The Real Cost Of Switching To Linux*; InfoWorld; InfoWorld Media Group; retrieved from the Internet on 10/16/2003 from:
http://www.infoworld.com/pdf/special_report/Linux_feature_2003.pdf

Mayor, Tracy; May 1, 2000; *Value Made Visible*; CIO Magazine; CXO Media Inc; retrieved from the Internet on 10/19/2003 from:
http://www.cio.com/archive/050100/method.html

McCullagh, Declan and Zarate, Robert; June. 12, 2002; *U.S. Gov't Still Penguin Shy*; Wired News; Lycos, Inc; retrieved from the Internet on 3/10/2003 from:
http://www.wired.com/news/linux/0,1411,53005,00.html)

Meals, Carole; 2002; *Business Case Analysis*; Mitretek Systems, Inc.; retrieved from the Internet on 10/02/2003 from:
http://www.mitretek.org/pubs/Sigma_Pubs_spring02/chap4.pdf

Miller, Robin; April 25, 2003; *Why Do Programmers Write Open Source Software?*; NewsForge, Open Source Development Network, Inc; retrieved from the Internet on 5/2/2003 from: http://newsforge.com/article.pl?sid=03/04/19/2128256

Mock, Edward J. and Shuckett, Donald Hart; 1973; Decision Strategies in Financial Management; American Management Association; New York, New York

Morrow, Vincent; 1991; *Handbook of Financial Analysis for Corporate Managers*; Prentice-Hall, Inc.; Inglewood, New Jersey

Norton, Michael J. and Story, Derrick; July 7, 2000; *Mac OS Terms and Definitions*; O'Reilly & Associates, Inc; retrieved from the Internet on 4/16/2003 from:
http://www.oreillynet.com/pub/a/network/2000/07/13/magazine/carbonese.html

Nuvolari, Alessandro; January 2001; *Open Source Software Development: Some Historical Perspectives*; Eindhoven University of Technology; retrieved from the Internet on 4/21/2003 from: http://opensource.mit.edu/papers/nuvolari.pdf

Panko, Raymond R.; Spring 1998; *What We Know About Spreadsheet Errors*; Journal of End User Computing, volume 10, number 2, pp. 15-21. A lengthier version in HTML format is available from: http://panko.cba.hawaii.edu/ssr/Mypapers/whatknow.htm

Paul Festa; August 29, 2001; *Governments Push Open-Source Software*; CNET Networks, Inc; retrieved from the Internet: http://news.com.com/2100-1001-272299.html

Peeling, Nic and Satchell, Julian; October 2001; *Analysis of the Impact of Open Source Software*; QinetiQ, ltd; retrieved from the Internet on 3/10/2003 from:
http://www.govtalk.gov.uk/documents/QinetiQ_OSS_rep.pdf

Prasad, Ganesh; July 26, 2001; *Yes, you can do Business with Open Source*, ZDNet India, CNET Networks; retrieved from the Internet on 4/20/2003 from:
http://www.zdnetindia.com/techzone/linux/stories/32999.html

Ragsdale, Cliff T.; 1998; *Spreadsheet Modeling and Decision Analysis*; South-Western College Publishing; Cincinnati, Ohio

Raymond, Eric S.; January, 2001; *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*; O'Reilly & Associates; Sebastopol, CA

Raymond, Eric S.; June 1999; *Shut Up and Show Them the Code*; retrieved from the Internet from: http://www.catb.org/%7Eesr/writings/shut-up-and-show-them.html

Rudwick, Bernard H.; 1969; *Systems Analysis for Effective Planning: Principles and Cases*; John Wiley & Sons, Inc.; New York, New York

Schmidt[1], Marty J.; 2003; *The IT Business Case: Keys to Accuracy and Credibility*; Solution Matrix, Ltd.; retrieved from the Internet on 10/3/2003 from: http://www.solutionmatrix.com/IT_Business_Case.pdf

Schmidt[2], Marty J.; April 2003; *Business Case Essentials: A Guide to Structure and Content*; Solution Matrix, Ltd.; retrieved from the Internet on 10/3/2003 from: http://www.solutionmatrix.com/Business_Case_Essentials.pdf

Schmidt[3], Marty J.; 2003; *What's a Business Case: And Other Frequently Asked Questions*; Solution Matrix, Ltd.; retrieved from the Internet on 10/3/2003 from: http://www.solutionmatrix.com/Whats_a_Business_Case.pdf

Schweik, Charles M. and Semenov, Andrei; January 2003; *The Institutional Design of Open Source Programming: Implications for Addressing Complex Public Policy and Management Problems*; First Monday, volume 8, number 1; retrieved from the Internet: http://firstmonday.org/issues/issue8_1/schweik/index.html

Scott, Brendan; July 15, 2002; *Why Free Software's Long Run TCO Must be Lower*; retrieved from the Internet on 11/7/2003 from: http://www.members.optushome.com.au/brendanscott/papers/freesoftwaretco150702.html

Stallman[1], Richard; March 21, 2003, last revision; *The GNU Project*; retrieved from the Internet on 3/24/2003 from: http://www.fsf.org/gnu/thegnuproject.html

Stallman[2], Richard; no date; *Selling Free Software*; Free Software Foundation, Inc.; Boston, MA; retrieved from the Internet on 3/21/2003 from: http://www.fsf.org/philosophy/selling.html

Stevens, Jr., G. T.; 1983; *The Economic Analysis of Capital Investments for Managers & Engineers*; Reston Publishing Company, Inc; Reston, Virginia

Stevenson, Cooper; February 2003; *Blueprint for Linux in the Enterprise*; MetaSource Technologies; retrieved from the Internet on 3/20/2003 from: http://www.toumaforester.com/download/linux_eprise.pdf

Stoltz, Mitch; 1999; *The Case for Government Promotion of Open Source Software*; The Tides Center / NetAction; retrieved from the Internet on 4/20/2003 from: http://www.netaction.org/opensrc/oss-report.html

Taft, Darryl K. and Petersen, Scot; March 24, 2003; *Open-Source Movement Gains Ground on Microsoft*; eWeek; pp 1,14; Ziff-Davis Media

Turnbull, Stephen J.; January 24, 2003, last revision; *XEmacs vs. GNU Emacs*; retrieved from the Internet on 4/10/2003 from:
http://www.xemacs.org/About/XEmacsVsGNUemacs.html

Varghese, Sam; March 2003; *Swedish Study says TCO for Open Source Software is Lower*; The Age Company Ltd; retrieved from the Internet on 5/2/2003 from:
http://www.theage.com.au/articles/2003/03/11/1047144950542.html

Viega, John; September 1999; *Open source software: Will it make me secure?*; Reliable Software Technologies; retrieved from the Internet from:
http://www-106.ibm.com/developerworks/security/library/s-oss-security.html#h0

Walkenbach, John; 2004; *Excel 2003 Formulas*; Wiley Publishing, Inc.; Indianapolis, In.

West, Joel; December, 2002; *How Open is Open enough*; San Jose State University; retrieved from the Internet on 4/21/2003 from:
http://opensource.mit.edu/papers/rp-west.pdf

Wheeler, David A.; February 26, 2003, last revision; *Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!*; dwheeler@dwheeler.com; retrieved from the Internet on 3/10/2003 from: http://www.dwheeler.com/oss_fs_why.html

Whitlock, Natalie; March 1, 2001; *The Security Implications of Open Source Software*; Casaflora Communications; retrieved from the Internet from:
http://www-106.ibm.com/developerworks/linux/library/l-oss.html

Wilcox, Joe; May 10, 2002; *Microsoft Program Meets Some Resistance*; CNet news.com; retrieved from the Internet on 4/12/2003 from:
http://news.com.com/2100-1001-908773.html

Wysong, Thomas G.; October 2000; *Introduction to Open Source and Free Software* (version 1.1); retrieved from the Internet on 4/12/2003 from:
http://technodemocracy.org/people/tgw/docs/ossfs.html
(as of 8/30/2003, this link no longer exists)