Samizdat:

And Other Issues Regarding the 'Source' of Open Source Code

By Kenneth Brown

(Excerpts from soon to be released book on open source software)

> *"Torvalds and his peers who oversee popular open source projects accept contributions from any and all sources based on the merits of the code alone. They don't have the institutional resources to ensure that a programmer isn't guilty of plagiarism…"*
>
> -Wired Magazine, November 2003
> Gary Rivlin, "Linus Torvalds, Leader of the Free World"

Kenneth Brown, President, AdTI
Justin Orndorff, Research Assistant
Alexis de Tocqueville Institution
Washington, D.C.
www.adti.net
kenbrown@adti.net

May 20, 2004

Table of Contents

Section V. Tuomi's Theory of Attribution and Invention

5.1 Further Explanation
5.2 Rembrandt, Minix and Linux
5.3 The Eliadian Election of Linus Torvalds
5.4 "Matthew Affected"
5.5 Eponymy Theory and Naming an Inventor

Section VI. A Closer Look at the "Art"

6.1 On Copying a Rembrandt
6.2 Reverse Engineering
6.3 Obfuscation
6.4 Pretty Printers
6.5 Comparison Programs
6.6 The Dilemma of Residual Risk

Section VII. Sustaining Corporate Partnership

7.1 Code Moving In or Out of the Corporation
7.2 The 'Source' of Volunteers
7.3 Sponsoring Competition

Section VIII. The Collapse of Samizdat: 8 Scenarios

8.1 Pot Luck Software
8.2 Terminal Partnerships
8.3 Un-Volunteerism
8.4 Diligence
8.5 "Too Small to Sue" Will Not Last
8.6 Money and Rights
8.7 Proprietary Becomes Fall-Back Position
8.8 Inevitability

Section IX. Achieving Balance

9.1 Keeping A Free Software Model
9.2 Importance of R&D Integrity
9.3 Government Policy
9.4 Agreement Across All Arenas

The Wall Building Contest

Samizdat Paper Abstract

Two brick masons are in a contest to build a wall----

The first mason has a massive, identifiable infrastructure. He manufactures his own bricks and has an expensive labor force to build the wall. The second mason has a volunteer workforce. The second mason also produces bricks faster than his counterpart, with no identifiable infrastructure or costs. --- And in fact, the second mason ends up winning the wall-building contest.

Befuddled, the first mason remarks, "I am impressed that you were able to build the wall with little cost and volunteers. Our cost for the bricks alone was millions of dollars. Moreover, not only was the cost to produce quality bricks a great setback, but the time it took us to make quality bricks was an even greater handicap." "So, I just have to ask", the first mason says, "How were you able to do both? How were you able to produce the same quality bricks, with relatively no cost, in less time than we could?" The second mason with a wry smile responds, "I can't tell you exactly how I do it, but trust me, it just gets done." Confused with the answer, the first mason reluctantly asks, "Do you acquire these bricks by legal means?" "Of course it is by legal means", the second mason responds firmly, "in fact, many of your employees are our biggest contributors."

"Samizdat: The Source of Open Source Code", discusses the controversial production factory of "free" computer source code. While the literal meaning of Samizdat refers to a period of freedom-fighting publishers in early Russia, the term has been borrowed by programmers that engage in the practice of surreptitiously circulating and/or using software source code that belongs to other individuals or companies. Whether it is reverse engineering, employee theft, or Rembrandt-like copying, plagiarism in software programming has become the proud flag of many in the 'open source movement'.

Samizdat begins in the 70's and delves into how the Unix operating system becomes the most imitated, licensed, and stolen software in the history of computer science. Moving through the 80's during a period of Unix envy, to the 90's, Samizdat explores the arrival of the Linux program, a product whose invention is so controversial, that its origin many argue defies both popular and scientific opinion.

Samizdat predicts that the practice of questionable source code production will not only negatively impact hi-tech, but intellectual property across all industries. Samizdat concludes that it is imperative that government leadership take significant steps to correct this problem, before it causes an irreversible setback to the intellectual property economy as a whole.

Samizdat is part of a soon to be published book on operating systems and open source authored by Kenneth Brown.

Foreword

The True Meaning of Samizdat

By Cynthia Martin, Associate Professor of Russian
University of Maryland, College Park
College Park, Maryland

To understand the appropriateness of the word samizdat in the title of this paper, a brief discussion about the word's meaning and its significance in Soviet history is in order.

Russian culture has always recognized the power of the word, spoken and especially written. In contrast to a democratic tradition predicated upon the notion that protecting free speech is necessary to foster the open exchange of ideas, a monolithic world-view, be it tsarist, monarchy, or Communist totalitarianism, cannot tolerate the potential for alternative positions or systems of government gaining broad support. The written word, as the bearer of such alternative ideas, is viewed as quite powerful, and hence, it is not surprising that official control over all forms of publication has been exercised throughout Russian history, especially during the Soviet period.

State-sponsored censorship developed during the pre-1917 tsarist period, and subsequently found its full elaboration in the Soviet Union. *Samizdat* was a response to the attempt by the Russian government to control access to all publications and publication outlets. *Samizdat* referred to the practice of "self-publishing" by dissident thinkers in a variety of areas, including political thinkers, academics and scholars, scientists, and literary and artistic figures in the Soviet Union. The word itself is based on two Russian roots: "sam" meaning "self," and ‘izdat’ meaning "publish," and resonates to the Russian ear as a kind of oppositional foil to the word "*gosizdat"*. The beginning of the word "gos" is an abbreviation of the full Russian word meaning ‘government.’ Literally translated, *gosizdat* referred materials officially published by the government.

By the 1930's, the state controlled all publishing in Russia, both in terms of decisions as to what would be published, and the actual material mechanisms for publishing, including all printing presses. For a work to be published, it first had to be passed by the official Soviet censors, who prohibited anything that could be perceived as being anti-Soviet. There were no alternative outlets for works denied by the official censoring organs, and so authors would often resort to replicating their manuscripts in limited numbers themselves. The main challenge for those producing *samizdat* rested in the primitiveness of the duplication technology available to them right up to the fall of the USSR in 1991. Copying machines were virtually inaccessible to the average person, and those that did exist in the country were heavily monitored and access was limited. Most often

samizdat took the form of multiple typewritten copies of a manuscript produced using simple carbon paper. These copies would then be circulated from hand to hand.

The punishment for producing *samizdat* or even possessing such self-published literature could be harsh, resulting in prison sentences or worse. To prevent unauthorized publishing, state control of the printing apparatus was so meticulous, that over long holiday weekends, for example, publishing offices containing typewriters and other forms of copying technologies were literally locked and their doors were sealed. The particular keystrokes of all typewriters were registered with the authorities so that illegally typed works might be traced to those responsible.

One of the most famous cases of a dissident writer whose works, political and literary, were published via *samizdat* is the case of Alexander Solzhenitsyn. His personal fate is evidence of how much Soviet Russia feared the bearer of alternative ideas, and how total the attempt was to control the dissemination of texts that offered alternative views. Solzhenitsyn came to be seen as more of a threat inside Russia, where he could still spread his anti-Soviet views, than outside, and therefore he was stripped of his Soviet citizenship and expelled from Russia in February 1974.

Samizdat played a significant role during the Soviet period in fostering the open exchange of independent ideas despite official censorship and strict government control of all publishing outlets. Many dissidents and readers paid high prices for producing or even possessing works published via samizdat, and even today the symbolism of the word samizdat resonates for those who lived through the Soviet period.

I applaud the Alexis de Tocqueville Institution's inclusion of this introduction to clarify for readers the true origin and definition of *samizdat.*

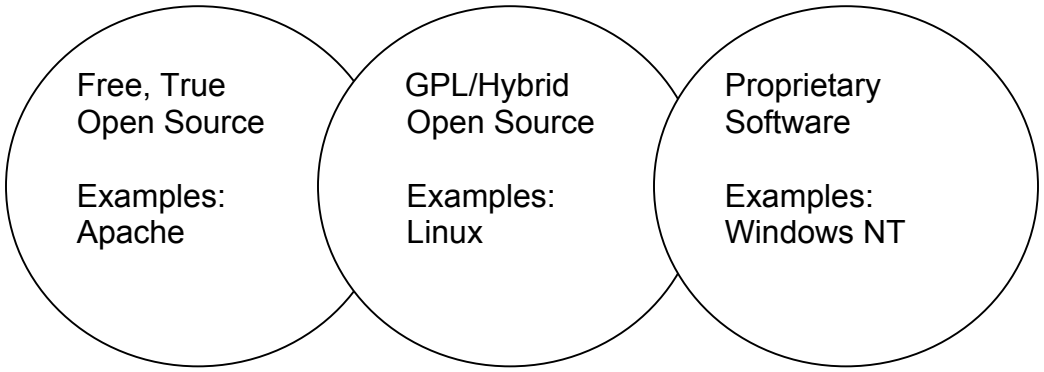
Introduction

Understanding the Hybrid Open Source Code Model

Software is a business. But ironically, free software is a business too. The free software model provides users with accompanying source code for modification or development of the original software. The business logic for providing free source code is to enable clients to modify/customize aspects of the accompanying software.

Linux and many other products are referred to as open source. But in fact they would more properly be referred to as hybrid source, products that attempt to offer the benefit of true open source, but operate in a commercial world like traditional proprietary products. For example Apache is a true open source product. In contrast, the Red Hat Linux operating system is a hybrid product. It is very important to differentiate between the two.

Differentiating Open Source, Hybrid Source and Proprietary Software



True open source is software and source code that can be used for any reason, for any use. If you get it with a license, it only requires attribution, or a copyright notice. You can modify it in any way and sell it as your own, without any additional requirements.

The second type, hybrid source, gets the lion's share of attention. It is software that is also no cost or free, but any modification to it becomes the 'equal property' of the original author and any user that is interested in it. In sum, if you use this type of free software code to develop and sell any new product, the source code

becomes open to any other user. This is done deliberately to nullify the value of what you have created. If it is open, then anyone can improve upon it, copy it or use it. Mandated openness ends an individual's ability to leverage its scarcity. Simply put, mandated openness, eliminates value because in essence everybody can have it. This model is regulated by the general public license (GPL). GPL licensed software in effect is a hybrid of open source and traditional proprietary software. It is not true open source. However, it is a product that is marketed largely for commercial use that is kept (theoretically) permanently free.

Any software that incorporates hybrid source code, becomes hybrid as well. The viral nature of hybrid source is not only a threat to true open source, but also traditional proprietary software because it unwillfully absorbs both products.[1] In contrast, with true open source and proprietary software, many hybrid proponents do not promote intellectual property rights, only to the extent that existing law can enforce the GPL, the tool that dictates all of its accompanying source code stay open and free in cost. Although introduced at a much later date, ironically, hybrid source has become the largest pool of free open source software.[2]

The original open source model, promoted by academics, scientists, and research encouraged collaboration. The original free model also encouraged government sponsorship of research because it was paid for by taxpayers, thus, belonged to the taxpayers. Anyone was free to use it and benefit from its commercial value.

The empirical success of open source is integral to the promotion of all science and technology. Likewise, perpetuating research and development is integral to the intellectual property economy. Commercial gain enables research and development to sustain itself.

However, it is unquestionable that the hybrid source code model is having a deleterious effect on both true open source, research and development, and the commercial intellectual property economy.

This paper examines how and why.

[1] Opening the Open Source Debate, AdTI, June 2002, Discusses the problems of GPL software.
[2] David Wheeler, More than a Gigabuck, Estimating GNU/Linux's Size
http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html

1.1 What is the Source of Open Source Code?

Linux and other hybrid source code products[3], commonly referred to as 'open source' software, have steadily migrated into the IT departments of both private and public institutions. As usage of the non-proprietary model of selling software and software services grows, like any other new technology, it is important to continually analyze its accompanying opportunities and consequences to best implement and shape relevant public policy.

Almost immediately, the notion of a discussion focused solely on the 'source' of open source code seems trivial. However, this topic speaks not just to the foundation of the open source platform, but to its perpetuation and long-term viability as a business model. Almost reflexively, the future and fate of open source is irreversibly linked to answers from a series of simple questions, such as where did Linux come from? Or, who owns Linux? These issues impact every developer and user consideration of open source in the immediate future.

Software is source code - and the topic of the 'source' of the code is as big as the billion dollar industry itself. Open source code is no different from any other type of intellectual property, thus the topic of open source code intersects every issue regarding the sale, use, and ownership of copyright protected intellectual property.

An issue that flies beneath the radar is the question – where does the successful Linux product come from? The origin of true open source code doesn't really matter, because a) it does not have many significant legal consequences of misuse b) it has almost no use restriction. It is definitely free—commercial products such as Linux are entirely different.

We know where traditional commercial proprietary source code comes from. We also know who its original owners are. However, we don't really know what the origin of the bulk of hybrid source code is. We don't know much about this pool of software, other than what we are told. The assumption is, there is no cause to ask---For example, we know that Linux is a free public domain product, given to us by its inventor Linus Torvalds. But not many people ask where did it come from?-----Is it a dumb question to ask, "what is the origin, the 'source' of this pool of source code?"

Is there a reason to focus a discussion just on the 'source' of open source code as opposed to the 'source' of proprietary code? The answer is no. However, while society (particularly the courts, government, inventors, the user community, etc.) has become well acquainted with the rules of proprietary source code for almost forty years, the topic of open source code is still a very new discussion.

[3] Find hybrid source code vs. open source code definition in paper introduction.

Interestingly enough, understanding the 'source' of open source code not only provides a better overview of the open source model, but also provides us with tools to resolving conflict and implementing policy for its use.

Another reason to focus specifically on the origin of open source code is because too often, discussion about the topic tends to stir heated, emotional exchanges, occasionally moving parties on both sides to questionable conduct. Some critics are even unlucky enough to receive widespread excoriation in public forums. Nevertheless, this backlash only increases the necessity to continue exploring these issues more thoroughly. Furthermore, fear, uncertainty, and doubt (FUD) will not better technology, nor open source software. Thus, it is better to delve further into the facts than to suppress them. As is often mentioned in open source discussions, "obscurity is not necessarily security."

1.2 Science, Law, Business, Public Policy & Open Source

Finally, there is a natural explanation why the open source topic usually evokes strong counter-opinions. As unassuming as it seems, open source is a topic that overlays four expansive arenas of major study at the same time: science and engineering, law, business, and public policy. Each of these major topics has sub-topics whose definitions change from arena to arena (see Diagram 1). Open source challenges thinking that in many respects are as old as the arenas themselves. It is rare to find any one person that is an expert in all four fields and understands the nuances of each arena, thus, during open source debates, we often see a communications breakdown reminiscent of the Tower of Babel[4].

While a great many topics intersect these four paradigms, open source is uniquely different because 1) open source is still a very new consideration within each arena, 2) there is a very limited amount of resolution/precedent that each arena agrees upon, 3) there are members with unlike thinking in each arena that side with opposing thinking, i.e. lawyers that disagree with the legal community but agree with the science community, or policy makers that disagree with government practice and side with the private sector. This fourth factor makes the debate especially volatile.

In addition, we find that some ideas are more compatible across arenas than others (see Diagram 2). To minimize entropy, answers to important questions are avoided, left vague, or deliberately ignored.

1.3 Reaching Agreement

In summation, the goal of a discussion exclusively on the 'source' of open source is to have a closer look at a topic of conflict that has caused disturbance across

[4] Old Testament, Genesis 11:1-9 …Therefore its name was called Babel, because there the Lord confuse the language of all the earth…

each arena. After a closer examination of this topic, it should become easier to find an acceptable approach for the adoption of an open source model. In addition, a rigorous look at this topic will surprisingly reveal that many members of each sector already agree and have arrived at similar conclusions. Finding these shared opinions and common goals should be help the majority of members within each arena to come to agreement. With better agreement, all groups can come up with a way that open source can best be used to improve technology and society.

Diagram 1

Multiple Theories on Source Code

Four Expansive Arenas With Different Theories On Source Code

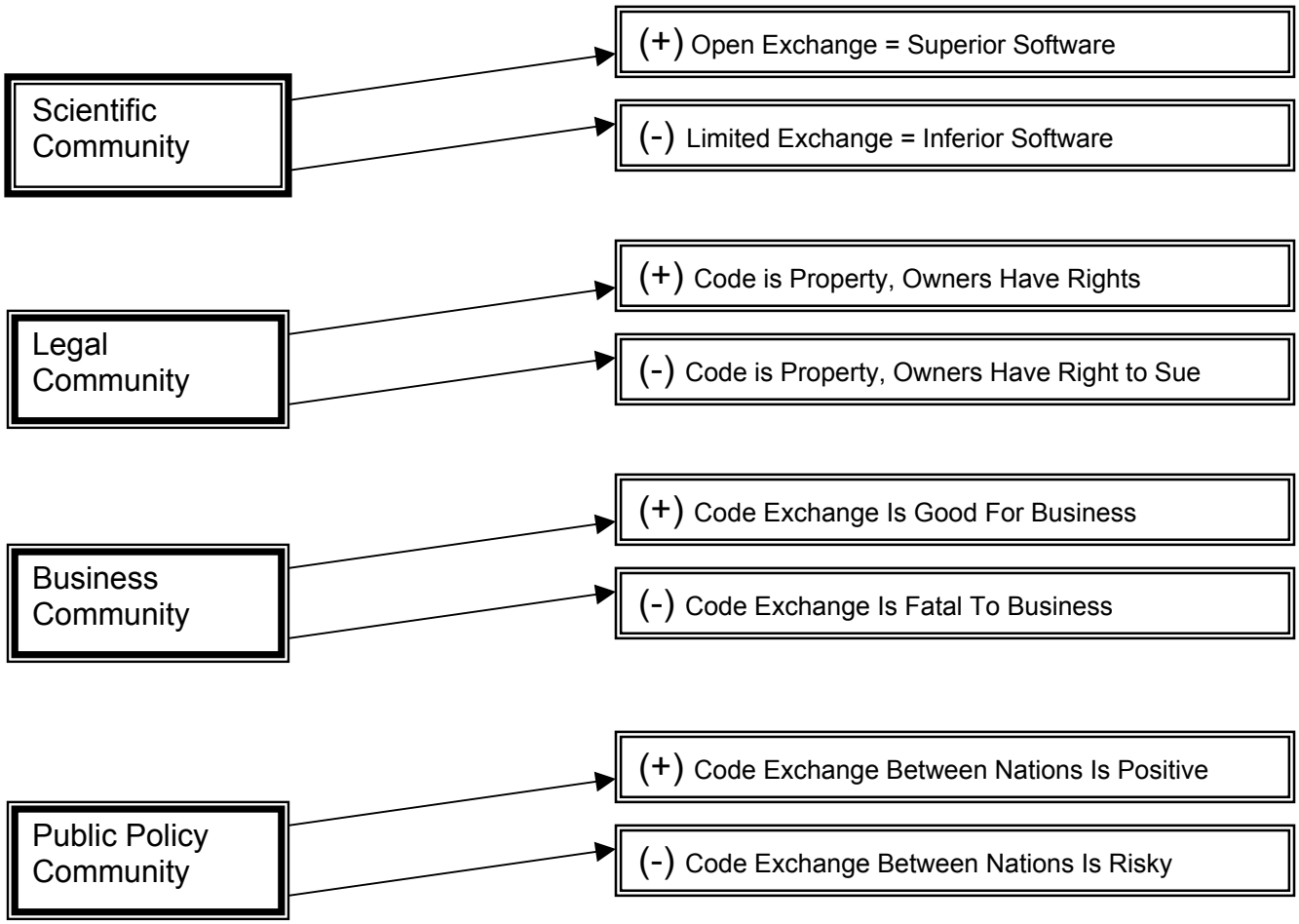
Science and Engineering	Legality and the Court System	Business Practice(s)	Government Regulation
Innovation • Permission for Use • Sanctity • Protection Exchange • Ownership • Sale • Attribution • Disclosure Secrecy • Theft • Derivation • Copy • Free • License National Security • Public Domain • Academic Use Only			

Diagram 2

Open Source & Ideology

Competing Open Source Ideologies

(+) Idea is more compatible between arenas
(-) Idea is less compatible between arenas



Section II. The Lions Incident

2.1 The Invention of Unix

Studying the history of the open source model enables us to better understand why the debate impacts different arenas. In particular, one notable event almost thirty years ago could be credited for triggering the global discussion about 'free software'.

In 1969, Dennis Ritchie and Ken Thompson invented Unix at Bell Labs. Dennis Ritchie and Ken Thompson found a little-used and obsolete PDP 7 computer, a tiny machine in the class of a Commodore 64 computer and built the first Unix kernel.[5] The operating system was named Unix, to distinguish it from the complexity that burdened MULTICS.[6]

"By early 1973, the inventor Ritchie explains, the essentials of modern C were complete. The language and compiler were strong enough to permit us to rewrite the kernel for the PDP-11 in C during the summer of that year."[7] Ronda Hauben, author of "*Unix and Computer Science*" writes, "They created a Unix kernel accompanied by a toolbox of programs that could be used by others at Bell Labs. The kernel consisted of about 11,000 lines of code. Eventually, 10,000 lines of the code were rewritten in C and thus could be transported to other computer systems."[8]

"The kernel, Ken Thompson writes, is the only Unix code that cannot be substituted by a user to his own liking. For this reason, the kernel should make as few real decisions as possible." Thompson describes creating the kernel, "What is or is not implemented in the kernel represents both a great responsibility and a great power. It is a soap-box platform on `the way things should be done.' Even so, if `the way' is too radical, no one will follow it. Every important decision was weighed carefully. Throughout, simplicity has been substituted for efficiency. Complex algorithms are used only if their complexity can be localized."[9]

Hauben adds, "just as Operating Systems people in the Bell system had come to recognize the need for portability in a computer operating system, Ritchie and Thompson and the other programming researchers at Bell Labs created the computer language C and rewrote the majority of the Unix kernel in C; and thus had made the important breakthrough in creating a computer operating system that was not machine dependent."[10]

---

[5] Hauben, Ronda. "Unix and Computer Science". *Linux Journal.* http://www.linuxjournal.com/article.php?sid-2792
[6] Ibid. MULTICS, (Multiplexed Information and Computing Service, earlier time-sharing operating system that inspired Unix. http://www.mit.edu:8001/afs/net/user/srz/www/multics.html
[7] Ritchie, Dennis. "The Development of the C Language," ACM, presented at Second History of Programming Languages conference, Cambridge, Mass, April 1993, p.9.
[8] Hauben, Ronda. "On the Evolution of Unix…". http://www.dei.isep.ipp.pt/docs/unix.html
[9] Thompson, Ken. "UNIX Implementation", The Bell System Technical Journal, Vol. 57, No. 6, July-August 1978, p.1931
[10] Hauben, Ronda. "Unix and Computer Science".

According to Bell Labs, “After three decades of use, the Unix computer operating system from Bell Labs is still regarded as one of the most powerful, versatile, and flexible operating systems (OS) in the IT world. Its popularity is due to many factors, including its ability to run a wide variety of machines, from micros to supercomputers, and its portability – all of which led to its adoption by many manufacturers…Its development and evolution led to a new philosophy of computing, and it has been a never-ending source of both challenges and joy to programmers around the world.”[11]

[11] The Creation of the Unix Operating System. 2002. Bell Labs. 16 Apr. 2004 <http://www.bell-labs.com/history/Unix>.

Diagram 3

Unix Co-Inventor – Dennis Ritchie

Short Dennis Ritchie Narrative Posted on Home Page
http://www.cs.bell-labs.com/who/dmr/

"When I joined in 1967, Bell Labs was a corporation jointly owned by American Telephone and Telegraph Company and its subsidiary Western Electric. Its official name was Bell Telephone Laboratories, Incorporated. Soon after, Ken Thompson, together with me and others, first started work on Unix. Also soon after, AT&T, which still owned most of the Bell System, updated its logo (I doubt the events were related). The new logo just updated the image; corporate structure remained the same. The material published by us during the period up to 1984 used this Bell logo and the name "Bell Laboratories." In 1984, AT&T, under a negotiated consent decree, divested the local telephone companies it had owned and in the process gave up the Bell logo and the Bell name except in connection with Bell Laboratories. Bell Telephone Laboratories Inc. was dissolved as a corporation and became an integrated unit of AT&T. We lost the Wehrmacht helmet and gained the Deathstar, and now identified ourselves as working at "AT&T Bell Laboratories." In 1996, AT&T (this time voluntarily) spun off its systems and technology organizations into Lucent Technologies, while AT&T kept the services business. Bell Labs stayed mostly with Lucent, though some of our colleagues helped form a new AT&T labs, much as some of us went to Bellcore in 1984. The new corporate logo usually includes the line "Bell Labs Innovations." Bell Labs remains a remarkably good place to do work that has enduring impact over the long run, no matter what the company, the courts, and PR types decide should be our name and logo on a given day or year."[12]

[12] Dennis Ritchie Home Page. Mar. 2002. Apr. 27, 2004. <http://www.cs.bell-labs.com/who/dmr/>.

Diagram 4

Unix Co-Inventor - Ken Thompson

Short Narrative Posted at www.sciencedaily.com/encyclopedia/ken_thompson and Wikipedia

"Kenneth Thompson (born 1943) is an US computer scientist, notable for his influence on UNIX. He was born in New Orleans, Louisiana, USA. He received in Bachelor's degree and Master's degree, both in electrical engineering, from UC Berkeley. In 1969, while at Bell Labs, Thompson and Dennis Ritchie were the principal creators of the Unix operating system. Thompson also wrote the B programming language, a precursor to Dennis Ritchie's "C", one of the world's most commonly used programming languages. Later, while still at Bell Labs, he and Rob Pike were the principal creators of the Plan 9 operating system. During this work, he also created the UTF-8 character encoding for use on the Plan 9 operating system. He also wrote programs for generating the complete enumeration of chess endings, for all 4, 5, and currently 6-piece endings. Using these, a chess-playing computer program can play perfectly once a position stored in them is reached. Thompson and Ritchie jointly received the Turing Award in 1983 "for their development of generic operating systems theory and specifically for the implementation of the UNIX operating system". Thompson's style of programming has influenced others, notably in the terseness of his expressions and a preference for clear statements. Thompson retired from Bell Labs on December 1, 2000."[13]

13 Ken Thompson - Wikipedia. July 27, 2001. Apr. 28, 2004.

2.2 Samizdat

To promote the study and promulgation of Unix, Bell Labs made Unix available to academic institutions at a very small charge. ATT had actually been rather forthcoming with licenses to Unix for academic use.[14] One university, the University of New South Wales, in Australia, had a faculty member, Professor John Lions of the Department of Computer Science at the University of New South Wales in Australia, that was able to acquire a copy of research Unix Edition 5 (with the tape and manuals) for $150 for the University in December 1974, including tape and manuals.[15]

Professor Lions decided to use Unix to teach his students operating systems architecture at the university. Since Unix was written in C, it was much easier for a student to trace the flow of the code algorithmically. The university was fortunate enough to have a source code license to Unix.[16] John Lions took the further step of deciding that it would be helpful to his students if he printed out the code and included almost line-by-line detailed explanation of the code as you read through the entire program. "Writing these", he recounts, "was a real learning exercise for me. By slowly and methodically surveying the whole kernel, I came to understand things that others had overlooked."[17]

Lions introduced his notes in the form of a notebook to ATT. Recalling Lions presenting his notes, Ritchie commented, "We in the research group reacted with great pleasure to Lions' book; it was very well done. Indeed the early Unix Support Group (that became USL etc.) were pleased as well, and in fact invited Lions for a couple of stays with them to help annotate more documentation. Lions also visited us later in the research group and did some annotation on early Plan 9.[18]

But Professor Lions had the interest of sharing his notes with other parties besides ATT and his students. "In keeping true to the Unix community spirit of helping each other, Lions wrote a letter to Mel Ferentz, Lou Katz and others from Usenix and offered to make copies of his notes available to others. After some negotiation with Western Electric[19] over the patent licensing, he distributed the notes titled, "A Commentary on the Unix Operating System" to others with Unix licenses on the conditions that Western Electric had set out."[20] According to Nick Moffitt, "If you had taken Lions' class at the time, you would have bought two

[14] Unix version 6 (circa 1976) was free for universities and version 7 cost $100 (government labs and commercial entities had to pay $21,000). This was partly due to a consent decree that forbade ATT from selling software commercially. Source: Moffitt, Nick. "Nick Moffitt's $7 History of Unix". http://www.crackmonkey.org/unix.html
[15] Hauben, Ronda. "On the Early History and Impact of UNIX Tools to Build the Tools for a New Millennium". http://www.people.has.harvard.edu/~lib113/reference/unix/unix3.html
[16] Ibid.
[17] Hauben, Ronda. "UNIX and Computer Science".
[18] Ritchie, Dennis. Personal interview, April 4, 2004.
[19] Western Electric, controlled Unix and Bell Labs, Division of ATT before 1984 Breakup.
[20] Hauben, Ronda. "UNIX and Computer Science".

books (one red and one orange), they were the Source code and Commentary on Unix Level 6. The class became quite popular."[21]

The exact history of what happened afterwards is unclear, but within one year of the preparation of the Lions commentary, ATT, concerned about its trade secret protection and intellectual property rights, changed its mind and decided that it wanted to end the distribution of the manuals. Dennis Ritchie comments, "What was decided from the licensing point of view (after 6th edition) was that teaching whole classes from the source might be worrisome, and I think he was asked not to continue doing this. The original book was never officially published except within UNSW for his course; however it was reprinted by ATT for internal purposes and it was also was made available to ATT/WEco Unix licensees. Lions was disappointed that things changed so that he couldn't teach courses from a newer edition, but I don't recall any animus."[22]

Somehow---- (no one seems to know exactly when) copies of the commentary and code started becoming widely distributed. In addition, somehow----- the announcement that the notes were banned fostered an even greater number of illegal copies of the notes becoming illegally copied and distributed.

The exchange of the notes was described as "samizdat", a word of Russian origin that used to describe the circulation of banned books that were circulated and read underground. Raymond comments in Hackers guide, "Lions Book n. ----Source Code and Commentary on Unix Level 6", by John Lions…for years the only detailed kernel documentation available to anyone outside Bell Labs. Because Western Electric wished to maintain trade secret status on the kernel, the Lions book was never formally published…In spite of this, it soon spread by samizdat to a good many of the early Unix hackers."[23]

Just as underground copies of banned books made their way around Russia via the black market, the photocopy machine facilitated countless illegal copies of the Lions notes. The Lions incident intersected moral and political thinking as well. Distribution and ownership of illegal copies of the Lions notes created a common 'consciousness' among many owners that it was wrong to suppress source code. In solidarity, programmers were quite open about their possession of the illegally copied code. Scott Barman recounts, "…The book has lived in infamy since. Arguably, it ranks as the most photocopied book ever, as it has passed from Unix geek to Unix geek in the 70's and 80's---including me. :-)"[24]

Ironically enough, the developer community, not its critics, proudly chose the term 'samizdat'. But it is this irony that provided an early sign that the open

[21] Moffitt, Nick. "Nick Moffitt's $7 History of UNIX". http://crackmonkey.org/Unix.html
[22] Ritchie, Dennis. Personal interview. April 4, 2004.
[23] Raymond, Eric S. New Hacker's Dictionary. 3rd ed. Cambridge, MA: MIT P, 1996.
[24] Barman, Scott A. "Re: SCO Forum98 Announcements (fwd)". Online posting. August 24, 1998. Mid-Atlantic Linux. April 15, 2004. http://boudicca.tux.org/mhonarc/ma-linux/

source community would not be tied to the private sector's traditional rules and restrictions regarding intellectual property or trade secret protection.

2.3 A New Politic

Professor Lions quickly became a cult figure in the developer community because, by default, he symbolized the 'new' opposition to obscuring source code. Even to this day, the late Professor Lions is a celebrated figure. In Australia, The Australia Unix and Open Systems Users Group[25] (www.auug.org/awards/lions) has a yearly John Lions Award. In addition, the University of New South Wales established the John Lions Chair in Operating Systems.[26] Ironically, his notoriety in part is due to the infamy of illegally copied books whose contents he didn't have the right to distribute.

While there are some accounts of this event that are murkier than others, it is solidly untrue that Unix source code was ever relinquished or given away for free by ATT. David Bloch an attorney with McDermott, Will & Emery discussing the question hypothetically comments, "…The source code is still copyrighted, regardless of whether it is "published for all to see." In fact, copyright's general purpose is precisely to ensure that artistic or other creative works are "published for all to see" without robbing the creator of his ability to profit from the work. Disclosure of copyrighted work has absolutely no bearing on liability for copyright infringement. Use of copyrighted code without permission is copyright infringement, regardless of whether the professor published or disseminated it. Copyrights aren't like patents or trade secrets---you can't dedicate a copyrighted work to the public merely by publishing it. But if the professor published the source code without permission, he's an infringer."[27]

In fact, ATT did insist on 'for academic use only' licenses that spelled out the terms of the use of its Unix code. However, there is a very spotty history of the Lions incident. For example, there is frequent mention that because Professor Lions was never restricted by a disclosure agreement, the code was not illegally copied. However, a disclosure agreement is irrelevant to the larger issue. As long as ATT retained the rights to Unix, it didn't make a difference whether Lions released it with or without a non-disclosure agreement. The code was never free, and the university was licensed the code with the specific purpose of academic use. Bloch continues, "I don't think a university professor has the absolute right to publish his findings, regardless of the countervailing interests of the persons or entities funding his research. This is not an issue of academic freedom. If the professor's miraculous drug or idea was developed at company expense, he ought to not have the discretion to reveal the idea on his own (thus

[25] John Lions Chair in Operating Systems. 2001. University of New South Wales. 15 Apr. 2004 <www.do.cse.unsw.edu.au/industry/JohnLions/chair.phtml>.
[26] The John Lions Award For Research Work in Open Systems. 2002. Australian UNIX and Open Systems Users Group. 15 Apr. 2004 <www.auug.org.au/awards/lions>.
[27] David Bloch interview with AdTI, April 9, 2004. Bloch was NOT asked about the Lions incident specifically, only legal questions about scenario.

potentially ruining the idea's commercial potential, which is presumably why the company gave the professor research dollars in the first place).[28]

This event is important because the Lions incident creates a discussion about the juxtaposition of open source for academic use only vs. open source for both academic and commercial use. Both programmers and academicians became disgruntled that they were charged for licenses for rights to the Unix code. Although many users continued to pay significant sums for a license to study and/or use the Unix code, owning an illegal copy of the Lions Book became a symbol of rebellion to "paying for source code" which later becomes a central theme of the open source community.

The Lions incident effectively fuses left, center and right wing views about open source code into a 'political' movement. Twenty-six years later, the open source community's different factions still resemble the early reaction to the Lions incident (see Diagram 3).

[28] Ibid

Diagram 5

Open Source Factions

Three Perspectives Of Lions Incident Become 'Factions' of Present Open Source Movement

1. Left Faction – "The samizdat exchange of the Lions commentary was merely academic and good for computer science. There was no theft. The publication of the Lions book and the need for open source is only a manifestation of that bond that already exists among fellow academics and scientists. Open source will only continue to improve developer capability and overall software development."

2. Center Faction- "The samizdat exchange of the Lions commentary was academic and good for science. Sometimes a little theft is necessary. But there is theft everywhere and the open source community should not be singled out. Anyway, there is plenty of evidence that suggests that releasing the code made Unix better and everyone else better off. Open source is a viable, competing software development model."

3. Extremist Faction – "The samizdat exchange was outright theft but it was necessary. Western Electric brought it on themselves anyway. Companies are wrong to suppress and leverage source code for their own gain. Furthermore, proprietary development denies us the best future for technology. All code should be free. Anyone for code obscurity is fundamentally wrong. Open source is superior and will be the prevailing model for all software development."

2.4 "Gray" Thinking About Source Code

Over the years after the Lions incident, more questions than answers surface. Worse, each arena arrives at "silent disagreement" about the matter. The arenas never come to a resolution on how to approach the wide circulation of the illegally copied source code. Source code theft of this sort is not a black or white issue…this by default precipitated "gray" thinking about the incident across each sector:

Science and Academic Community

New Questions
Was the widespread illegal publication of the ATT source code a violation of academic/private sector partnership? Did the education community gain or lose by the widespread publication of the Lions book? Was it unaffected?

Gray Consensus
There is 'warm' approval in the community regarding the use and distribution of the Lions Book via samizdat. The publication of the source code did NOT hurt Unix or the private sector's ability to sell the product. Educational institutions and education should not be confined to the same rules as commercial organizations. Samizdat is wrong, but understandable, and sometimes benefits education.

Business Community

New Questions
Was there high demand for the Lions book because of its helpful analysis or because of the value of the source code?

If it was for its analysis, is the source code irrelevant?
If it was for the source code, is it logical to deduce that source code should and must remain a closely guarded secret by corporations?

Gray Consensus
Partnering with universities should include processes to minimize loss of any source code via samizdat, but in such cases, the partnership is "cost-benefit". The private sector cannot "fear" its source code being released into the public domain.

Legal Community

Gray Issues
Should/could an academic institution be held liable for this type of action? Did ATT legally lose the Unix code to the public domain? If the theft is widespread, is it unreasonable to ask a court to enforce its copyright with all users and developers? Can a property theft become too big to challenge? If source code is illegally in the public domain in a widespread fashion, should it or can it be

reclaimed with damages by a plaintiff in a court of law? Are users of the Unix code in the Lions book in violation of copyright law if ATT could prove they created software with an illegal copy of the Unix source code? Are derivative creations violations of copyright law as well?

Gray Consensus
- No significant amount of published opinion or outcry regarding the Lions incident.

Government

Gray Issues
Does the theft of source code impact society: Should government have a say in these types of matters? Should government be concerned about source code illegally ending up in the public domain? Is illegal copying of source code a public policy issue? Could a samizdat incident be a threat to society or national security?

Gray Consensus
No public policy direction, incident never significantly debated at any level of government.

In sum, the Lion's incident fused a new group – that was in favor of exchange and/or openness of not just academic source code but source code for commercial use as well. Although the incident was significant, members of the arenas did not come to any agreement of whether the perpetuation of this type of thinking was good or bad for society.

Clearly, the Western Electric lawyers saw commercial value in ending the wide availability of the Unix code, so they made the decision to end the book's circulation. However, Western Electric's inability to aggressively prosecute anyone reinforced the 'idea' that it was inconsequential for a programmer to have an innocent copy of stolen or illegally traded source code.

Section III. A Fork in Unix

3.1 10 Years of Unix Licenses (and Un-Licensees)

Unix licensees go on to create new versions of the operating system. Eric Levenez, a chief project manager with Thales in Europe, created a chronicle of these events, diagramming the history of Unix. The Levenez chart is one of the most detailed histories of Unix ever published. It provides a 'family tree' style diagram of Unix showing each divergence of Unix development across the decades. (www.levenez.com/unix)[29]. Levenez's chart starts at 1969 with the Ritchie, Thompson invention and moves through the decades to show each version of new Unix development. The Levenez chart is particularly helpful because it illustrates the forks in Unix development in a way that you are able to determine the order of derivative works from each version of Unix. For example, the chart details Unix V.6 development by Berkeley Systems Development (BSD) in 1978, SunOS development from 4.1 BSD in 1982, and Unix V.7 development of Xenix by Microsoft in 1980.

The Levenez chart however does not differentiate between products that spring from unlicensed Unix source code vs. licensed Unix code. Subsequently, the Levenez chart does not make this part of the history of Unix very clear.[30] However, the Levenez chart is extremely helpful in discerning which software continues to evolve and which remain homogenous and/or stop in its evolution.

After 1984, ATT continues to keep tight reins on the Unix code, licensing it to a wide assortment of vendors and organizations. Many companies license Unix code for both offensive as well as defensive reasons. Offensively, they licensed Unix to be competitive. Defensively, having a legitimate license ended the risk of copyright infringement. Should a commercial version of Unix or a Unix-compatible product appear without permission from ATT, it was almost certain that it would be challenged in court. The Levenez chart provides us with the history of each commercial version of Unix, almost all of which originate, from licensed Unix code, a Unix licensee, or a previous Unix licensee. Although Unix is widely circulated, licensed and unlicensed, for the next ten years, almost every Unix compatible operating system is built by a Unix licensee, or someone with considerable familiarity and expertise in Unix- except one - which we shall discuss later.

As noted earlier, the Unix kernel source code, owned by ATT, was freely exchanged in illegally photocopied books from 1977 forward without substantive prosecution of any sort. Although it is unspoken, the Lions incident creates two worlds, 'true' Unix invention from licensed code and Unix invention from unlicensed code.

[29] Levenez, Eric. "Unix History". http://www.levenez.com/unix/
[30] "My Unix chart is not a representation about copyright or patent." Eric Levenez interview with AdTI, April 26, 2004.

3.2 The Invention of Minix

By 1984, the PC is growing from a curiosity to a household word. Unix, an operating system for a client-server environment, was being studied actively for its potential use for PCs. However, a professor in the Netherlands, still angered about the Unix licensing requirements and the Lions incident, decides to build a Unix like software that he could use to teach his students about operating systems.

Vrije University is a private university located in the southern part of Amsterdam. It is a medium-sized Dutch-speaking university, with over 15,000 students. Its Department of Computer Sciences has about 700 bachelors and masters students.[31] Andrew Stuart "Andy" Tanenbaum is the head of Department of Computer Systems at Vrije University and teaches courses about computer organization and operating systems. Professor Tanenbaum is widely recognized as one of the most prolific theorists on computer science and operating systems (see diagram 5) and in 1984, begins writing a Unix-clone from scratch. There are other operating systems released with a similar history, also reportedly without any Unix source code.[32]

In 1987, in a partnership with Prentice Hall, Tanenbaum releases Minix 1.0, which stands for Mini-Unix, a product compatible with the V7 version of Unix. Tanenbaum comments, “We got Unix version 6 as soon as it was released. We were the second licensee in Holland, after the CWI. We ran V6 on our PDP-11 until V7 came along. We also taught V6 using Lions' book. When V7 came along and that (using the Lions Commentary) was forbidden, I wrote MINIX to have something to teach.”[33]

Established in 1913 by Charles Gerstenberg and Richard Ettinger, Prentice Hall is well known as one of the most successful publishers of textbooks. Recently merged with Pearson Education, Prentice Hall is now part of Pearson (NYSE: PSO), an international media conglomerate. In addition to Pearson Education, Pearson's primary operations include the Financial Times Group and the Penguin Group.[34]

In 1987, Prentice Hall is not very interested in distributing the digital form of the Minix source code and software with the book. However it agrees to do so, although publishing a book with diskettes in the back was a cumbersome process. Prentice Hall had a significant investment in the production, and wanted to increase the price of the book to recoup its costs for providing the

---

[31] “Vrije University: Welcome to the Department of Computer Science”. http://www.cs.vu.nl/welkommern.html

[32] Minix is often compared to a number of 'Unix-like' operating systems such as Idris, Coherent and Uniflex. Reportedly, these systems were written because ATTs initial licencing of Unix precluded it being sold to commercial organisations. According to Wikipedia, these operating systems are similar to Minix because reportedly they were written from scratch without access or license to ATT Unix code.
Wikipedia: http://en.wikipedia.org/wiki/Minix.

[33] Tanenbaum, Andrew. Online interview. April 12, 2004.

[34] About Prentice Hall; http://vig.prenhall.com/catalog/main_content/0,4929,-12,00.html

floppy diskettes for Minix installation within each book. Conversely, Professor Tanenbaum wanted to keep the price as low as possible to keep the book affordable for students. In a later version of the book, it was agreed that Minix would be downloadable on the Internet.[35]

Minix was hailed by thousands of programmers as a great tool for learning and, in the spirit of education, in 1987 Professor Tanenbaum set up a forum online for other parties interested in Minix to exchange, comments, ideas, etc. on how to improve the program (comp.os.minix).[36]

[35] AdTI interview with Andrew Tanenbaum. March 23, 2004.
[36] Ibid.

Diagram 6

Minix History

Minix History[37]

1987: "MINIX -- which stands for Mini-Unix"[38] 1.0 released
1990: MINIX 1.5
1992: MINIX 1.6.25
1996: MINIX 2.0
1998: MINIX 2.0.2
2000: MINIX becomes BSD-licensed[39]
2001: MINIX 2.0.3

---

[37] Minix Timeline (1961-2004). 30 Aug. 2002. 15 Apr. 2004 <http://minix-up.sourceforge.jp/timeline/TIMELINE.html>.
[38] Unger, Thomas. The Amiga Minix Page. 2003. 16 Apr. 2004 <http://home.arcor.de/kickstart/TKA/Tutorials/AmigaMINIX/aminix.html>.
[39] "The Minix license changed in April 2000, and applies retroactively to all previous Minix distributions, even though they still carry the old, more restrictive license within." Andrew Tanenbaum. Source: http://www.cs.vu.nl/pub/minix/

Diagram 7

Brief Bio on Professor Andrew 'Andy' Tanenbaum

Andrew Stuart "Andy" Tanenbaum (born 1944) is the head of Department of Computer Systems, Vrije Universiteit, Amsterdam in the Netherlands. He was born in New York City and raised in White Plains, NY. (Quote) Developer of the 'Amoeba' distributed operating system[40]. Andrew Tanenbaum received in B.A. at MIT and a Ph.d. from University of California at Berkeley. Andrew Tanenbaum is the author of Minix, and responsible for the first significant fork in Unix.

Published Works on Operating Systems

Short list of Andrew Tanenbaum's Research Papers:

P. Homburg, L. van Doorn, M. van Steen, A.S. Tanenbaum, and W. de Jonge. "An Object Model for Flexible Distributed Systems," Proc. First Annual ASCI Conference, Heijen, Netherlands, May 1995 pp. 69-78.

M. van Steen, P. Homburg, L. van Doorn, A.S. Tanenbaum, and W. de Jonge. "Towards Object-based Wide Area Distributed Systems." Proc. Fourth Int'l Workshop on Object Orientation in Operating Systems, IEEE, Lund, Sweden, August 1995, pp. 224-227.

M. van Steen, F.J. Hauck and A.S. Tanenbaum. "A Scalable Location Service for Distributed Objects." Proc. Second Annual ASCI Conference, Lommel, Belgium, June 1996, pp. 180-185.

Books Authored by Prof. Tanenbaum

'Computer Networks, First Ed.' 1986 Prentice Hall
'Operating Systems: Design and Implementation, First Ed.' 1987 Prentice Hall
'Minix 1.5 for Amiga' 1990 Prentice Hall
'Distributed Operating Systems' 1995 Prentice Hall
'Operating Systems: Design & Implementation, Second Ed.' 1997 Prentice Hall
'Structured Computer Organization, Fourth Ed.' 1999 Prentice Hall
'Modern Operating Systems, Second Ed.' 2001 Prentice Hall
'Distributed Systems: Principles and Paradigms' 2002 Prentice Hall
'Computer Networks, Fourth Ed.' 2002 Prentice Hall

[40] Andrew S. Tanenbaum - Wikipedia. 25 Feb. 2002. 15 Apr. 2004 <http://en.wikipedia.org/wiki/Andrew_Tanenbaum>.

3.3 The 'Invention' of Linux

By 1991, the Minix user forum exceeds over 40,000 participants.[41] Minix is exceptionally popular and generates more requests for modifications than Professor Tanenbaum can readily commit to. The professor wrote the program for academic use only. However, he receives frequent comments and requests about altering Minix but is not interested. Tanenbaum commented, "I was going crazy with the endless stream of new features people were sending me. I kept refusing them all because I wanted to keep MINIX small enough for my students to understand in one semester…".[42]

One participant interested in changes was a Helsinki University computer science student named Linus Torvalds. Linus posts the following message to the Minix group:

> 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 portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.[43]

Around September of 1991, a month later and approximately six months from his announced start date, Linus Torvalds releases Linux V0.01.[44]

---

[41] Tanenbaum, Andrew S. Andrew S. Tanenbaum's FAQ. 8 Apr. 2004 <http://www.cs.vu.nl/~ast/home/faq.html>.
[42] Bezroukov, Nikolai. Portraits…
[43] Torvalds, Linus B. "What would you like to see most in minix?" Online posting. 25 Aug. 1991. comp.os.minix. 10 Apr. 2004. <http://www.ibiblio.org/usenet-i/groups-html/comp.os.minix.html>.
[44] "Birth of Linux", Linus Torvalds http://www.externet.hu/linux/linux-birth.html

This announcement becomes, however, only the middle of the Linux story. The beginning is the process and development of the unprecedented software; the beginning is about how Linux came into being.

Diagram 8

Torvald's Pre-Linux Career

Brief Timeline of Linus Torvald's Pre-Linux PC Career[45]

1988	Enrolled at Helsinki University Computer Science Major
Fall 1990	Helsinki University acquires MicroVAX System, Linus "started" to learn digital Unix. Linus exposed to the Minix Community.
	Linus learns C Programming Language
Jan. 1991	Linus Buys a 386 PC
April 1991	Linus Announces that he has Started "Linux" Project On Minix Community Usenet
Sept. 17, 1991	Linux v0.01 Announced
Oct. 1991	Linux v0.02 Announced – The first "official" version of Linux

---

[45] Bezroukov, Nikolai, Portraits… Chapter 4.

Diagram 9

Linux Development Chart

Brief Linux Development Chart[46]

- Linux version 0.01 released mid September, 1991
- Linux version 0.02 released 5th October, 1991
- Linux version 0.03 released late October, 1991
- Linux version 0.10 released December, 1991
- Linux version 0.11VM+ released Christmas, 1991
- Linux version 0.12 released January, 1992
- Linux version 0.95 released March, 1992
- Linux version 1.0 released 14th March, 1994
- Linux version 1.2 released March, 1995
- Linux version 2.0 released 9th June, 1996
- Linux version 2.2 released January, 1999
- Linux version 2.4.0 released January 2001
- Linux version 2.6.0 released 17th December, 2003

---

[46] Stein, Harvey J. "Benevolent Dictatorship." Online Posting. 20 Mar. 1998. LispOS. 16 Apr. 2004 http://lists.tunes.org/mailman/listinfo/lispos and Raymond, Eric S. The Cathedral and the Bazaar. Apr. 10, 2004 <http://www.free-soft.org/literature/papers/esr/cathedral-bazaar/cathedral-bazaar-10.html>. Raymond, Eric S. Interview with AdTI,

Section IV. Did Linus Invent Linux?

4.1 Without A Doubt…

Linux is released in 1991. By 2000, there are hundreds of references to Linus Torvalds as the creator of Linux, the inventor of Linux, and the brains behind Linux. The gravity of the history is this: Unix is now firmly on two tracks, one commercial version that is proprietary and another commercial version that is open source and free. Linux becomes the most successful commercial-use, open source Unix clone; claiming to owe no license fees, royalties, copyright, original code, credits, or use permissions to either ATT/Unix or Prentice Hall/Tanenbaum and Minix. No other product accomplishes this success commercially.

However, due to the significance of this event, it is important to note that if any aspect of this story is even slightly exaggerated or untrue, there are literally hundreds of ramifications and questions. But is there any question about Linux development?

Nikolai Bezroukov, author of "Portraits of Open Source Pioneers" publishes a comprehensive, punchy narrative about the introduction of Linux. What is notable about the piece, is Bezroukov's repeated reticence about the "development" of Linux. Specifically, whether Linus invented it as a solo project, how much credit he should get, and whether Linux was actually Professor Tanenbaum's Minix program with a few bells and whistles.

While Bezroukov says complimentary things about Linux, his writing is almost contradictory, extolling the accomplishment of Linux, while at the same time, doubting whether everything has been 'vetted' about the road to Linux. His comments include:

> "Linux was and is an excellent piece of work and Linux owes a lot to Minix as a system and even more to the Minix community --- the decisive fact in the success of Linux."
>
> "I would like to stress the importance of the community – it was approximately 40 thousand that included a lot of very talented software developers (and many of them had know Unix internals much better than Linus and had already published code that enhanced Minix to make it more Posix compatible)."
>
> "I do not buy the idea that Linus Torvalds started developing the kernel for personal use and "for the fun of it"…..But, anyway, for a student with approximately just one year experience in C and a simple terminal emulator as the only (semi) completed project, it was an extremely bold move…"

Diagram 9

On the 'Invention' of Linux

What Linus Says…

"Linux has been written from scratch, and therefore does not contain any ATT or Minix code—not in the kernel, the compiler, the utilities, or the libraries."

Linus Torvalds, Linux Information Sheet (1992)

'I'm basically a very lazy person who likes to get credit for things other people actually do.'

Interview with Eric Raymond

What They Say…

"Linus Torvalds – a 21-year-old student at the University of Helsinki – wrote the kernel from scratch over a winter break in 1991 and made it available on the Internet for everyone to use."

Malobi Chodhuri "Linux: the Choice of the 'GNU' Generation" (2003)

"It all started seven years ago when Finnish computer science student Linus Torvalds grew frustrated at the lack of a decent operating system for his home computer. So he decided to write his own from scratch."

Kevin Colville, "The Linux Operating System"

"He was a skilled programmer, and came from a time when the source code had been available, so his software was very Unix-like, but written from scratch."

Thomas Brady, "Linux: What's the Big Deal?"

What Really Happened?

"Linus Torvalds, for example, didn't actually try to write Linux from scratch. Instead, he started by reusing code and ideas from Minix, a tiny Unix-like OS for 386 machines. Eventually all the Minix code went away or was completely rewritten…."

Eric Raymond, "The Cathedral and the Bazaar"

"Thus, media sources frequently make erroneous statements such as claims that the entire Linux operating system (in the popular sense) was written from scratch by Torvalds in 1991, that Torvalds directs the development of other components such as graphical interfaces, or that new releases of the kernel involve a similar degree of user-visible change to new major versions of proprietary operating systems such as Windows."

Sciencedaily.com Encyclopedia

The reason why the Bezroukov critique is particularly interesting is because it directly conflicts the account of almost all media history and accounts of Linus Torvalds that 1) Linus is the sole inventor of the Linux kernel and 2) that Linux is 100% original—specifically Linux is Not Minix...it is an independent product, not a derivative.

Linux remains an underground product for many years. To date, no other product, comes to life this way. As its significance as an event is unquestionable, the exact truth of this event is equally significant. Particularly because, this new program reportedly is created by Linus Torvalds, who reportedly had the right to give it away. The problem is though, depending on how you look at the history, Linux may not have exactly been his to give away…

4.2 A Few Problems With The Story

Problem #1 The Money Question

It would be safe to guess that billions of dollars have been invested by corporations to 1) buy the Unix source code and 2) to develop Unix operating system products and applications from 1969 forward to 1991, and from 1991 forward.

Thus, the first problem with Linux is very simple. Why are the most brilliant business minds in the history of PC technology, with hundreds of millions of dollars in capital, licensing Unix source code, if it is as simple as writing it from scratch with little help or experience? Is it possible that building a Unix operating system really only takes a few months—and, oh by the way, you don't even need the source code to do it?

Problem #2 The Access Question

> Tanenbaum: ""MINIX has been written from scratch, and therefore does not contain any AT&T code--not in the kernel, the compiler, the utilities, or the libraries. For this reason the complete source can be made available (by FTP or via the WWW)."[47]
>
> Torvalds: "Yes – it's (Linux) free of any Minix code."[48]

The fork (Unix to Minix & Minix to Linux) from licensed to unlicensed-development in Unix is one of the significant events in the history of technology. We also know that the event has a troubling inconsistency. Both authors insist that they wrote these operating system kernels from scratch. This means they wrote their systems without any Unix source code.

[47] Tanenbaum, Andrew. Minix Information Sheet. Vrije University. Apr 19, 2004 http://www.cs.vu.nl/~ast/minix.html.
[48] Torvalds, Linus B. "What would you like to see most in minix?" Online posting. Aug 25, 1991. comp.os.minix. Apr 10, 2004. http://www.ibiblio.org/usenet-i/groups-html/comp.os.minix.html.

This is significant because they both had access to the Unix source code: 1) Both Vrije University and Helsinki University were academic licensees of Unix source code at one time or another. 2) Tanenbaum taught from the Lions notes until it was banned. Many suspect that Linus also had the Lions notes.[49] It would be very difficult to prove that Linus had access to the Unix source at his campus, it also would be equally difficult to argue that he didn't. The following excerpt of an interview with Helsinki University faculty member Petri Kutvonen elevates this point:

> Hi,
>
> …I gather you mean Unix *source* license. I have to actually answer yes and no. We had actually acquired Unix source license in the beginning of 80's, a Decade before Linus' stay here but that was for the seventh edition Unix and not much of interest in the late 80's or early 90's and I doubt if Linus ever did see a single line of original Unix code.
>
> However, at that time the department of was involved in many Unix activities, we (or maybe it was the entire university) was an academic member of both major Unix consortia, Unix International and Open Software Foundation. In addition we participated in something called ASUP, Academic Support Unix Program, sponsored by AT&T and the Italian Olivetti. At that time we ranSy stem V on an AT&T 3B2 machine, Ultrix on a MicroVAX and SunOS 4.0 Numerous Suns.[50]

It is safe to argue that Tanenbaum, one of the most accomplished computer science educators in the world, with considerable understanding of the Unix architecture from teaching from the Lions notes for a significant amount of time, could build a Unix compatible kernel in three years without the Unix code. However, it is highly questionable that Linus, still just a student, with virtually no operating systems development experience,[51] (see Diagram 6) could do the same, especially in one-sixth of the time.

Problem #3 The Derivative Question

If Linus started with Minix and writes the Minix code out of Linux, Linux becomes unquestionably a derivative product, a product that is not necessarily independent intellectual property, especially without their permission. Many dispute the Raymond claim. However, the history is conflicted--- most will say it that Linus based Linux on Minix. Moreover, the described process is not invention; but by definition, reengineering.

---

[49] Bezroukov, Nikolai. Portraits…
[50] Professor Petri Kutvonen, Interview With AdtI, May 13, 2004
[51] Bezroukov, Portraits…. Chapter 4 writes, "He owned a Sinclair QL, knew Assembler language and enjoyed playing computer games in the late eighties, with Prince of Persia as a special favorite."

Deeming the Linux kernel to be a derivative work has a number of legal implications. But the bottom line is this, if a case could be made that Linux started from Minix, arguably not a free source code product until 2000[52], for almost ten years, Linux development was unsanctioned Minix development. (see Diagram 10).

This point resonated with Tanenbaum years before Linux. The following is a posting by Tanenbaum:

> Tanenbaum Usergroup Posting about Minix 1.5
>
> "Once more, to avoid any confusion, MINIX is not public domain or shareware. It is copyrighted software. All the files in this directory, including the demo disks are also fully protected by copyright. The copyright owners hereby grant permission to copy and use the MINIX demo disks and other files in this directory only for the purpose of evaluating MINIX to see if it is suitable for your needs."[53]
>
> Andy Tanenbaum (ast@cs.vu.nl)

Jason Kipnis a partner at Weil, Gotshal & Manges comments on the issue on the definition derivative production: "Typically, something is assumed to be a derivative work if it is derived from another work. Pretty basic…The question is when something is so far removed from the original work that it CEASES to be a derivative work. That is a question of degree. And there is no simple solution." On the implications of derivative works, Kipnis continues, "Whether the owner of the original work has rights to the derivative work is open to negotiation between the parties. Certainly, the owner continues to own its rights in the original work, even if it has somehow been incorporated into a derivative work. Commonly, the owner of the original work will receive a non-exclusive license to the derivative work, but I've also seen arrangements in which the owner of the original work also owns all rights to any derivatives."[54]

4.3 Is Anyone Looking?

It is unclear whether ATT or Prentice are paying attention to Linux development or take it seriously, but between 1991 and 2000, while Unix and Minix are restricted for commercial use, a new program Linux is ready for commercial use without any obligation whatsoever to either party. Undoubtedly, ATT nor Prentice Hall figured that Linux would be very relevant commercially. It is also presumable that neither ATT or Prentice Hall ever asked any questions about

---

[52] http://minix1.hampshire.edu/faq/mxlicense.html. "MINIX is now available under the BSD license modified: Apr 9 2000, Better late than never. I finally got permission from Prentice Hall to change the MINIX license to the BSD license. The lawyers sort of sat on this for two years…"

[53] Tanenbaum, Andrew. Read Me. Vrije University. Apr. 25, 2004. <http://www.cs.vu.nl/pub/minix/old/demo/READ_ME>.

[54] Kipnis, Jason. Interview with AdTI,

how exactly the 'free' operating system with so many similarities to Unix and Minix came to be in August 1991.

Diagram 10

Early Minix License Language

Posting on comp.os.minix

Copyright (c) 1987,1997, Prentice Hall
All rights reserved.[55]

Redistribution and use of the MINIX operating system in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Prentice Hall nor the names of the software authors or contributors may be used to endorse or promote products derived from this software without specific prior written permission.

11. LEGAL STATUS OF MINIX[56]

Although MINIX is supplied with the complete source code, it is copyrighted software. It is not public domain. It is also not like GNU. However, the copyright owner, Prentice-Hall has granted permission to bona fide universities to copy the software for use in courses and in university research projects. It is also permitted for MINIX owners to change the software to suit their needs and to distribute diff listings containing their changes freely. The shrink-wrap license that comes with MINIX states that you may legally make two backup copies of the software. Prentice-Hall is being much less strict than other software vendors. Please do not abuse this. Companies that wish to embed MINIX in commercial systems or sell MINIX-based products should call (212) 753-7753 to discuss licensing terms. [57]

[55] Comp.os.minix
[56] Minix Information Sheet, last changed, June 14, 1995, http://www.faqs.org/faqs/minix-info/
[57] Minix Copyright. Apr. 25, 2004. <http://www.cs.vu.nl/pub/minix/LICENSE>.

4.4 Brooks' Law and Linux

Journals and media celebrate Linus as an inventor, who wrote Linux from scratch. With the lack of challenges to this claim, objective individuals are still left to wonder: is it possible that the 21 year-old could have written Linux from scratch? More questionably, is it possible that Linus and a group of volunteers could have produced the debugged, working Linux operating kernel (Linux V0.12) by January 5, 2002 from scratch?[58]

This discussion would be incomplete without a brief overview of software engineering, and the physical production of software. An overview of this topic also provides us with a very helpful perspective on the software industry, the success of Unix, and the conspicuous nature of the Linux creation.

Building a stable, widely accepted operating system is 1) the result of exceptional talent, 2) the result of an exceptional amount of time. As we have seen, both the open source community and the proprietary community rallied around existing Unix technology; technology that was tried, true and tested. Before there was Windows, almost every software company in the country was building applications for Unix standard operating systems or releasing a new version of the Unix operating system. Understanding this anecdotally is helpful, but scientific explanation provides valuable insight.

Frederick Brooks, Jr. was a manager of the OS/360 operating system project at IBM during the early 1960s. After personal experience and further study, in 1975, Frederick Brooks, Jr. published, "The Mythical Man Month", a book about software engineering, specifically managing large, complex software projects.

Although the first edition of "The Mythical Man Month" was released during the time period of early Unix and PC development, most insist that although almost thirty years later, Brooks' theories remain extremely accurate about software development. His widely accepted analysis on the amount of time required to produce quality, useful source code from scratch is especially relevant to our discussion.

Ray Duncan, writes in a book review about the Brooks treatise, "Assume that such a program might take one very smart, highly-motivated, expert programmer approximately a year to design, code, debug, and document- in other words- 12 man-months. Imagine that market pressures are such that we want to get the program finished in a month, rather than a year. What is the solution? You might say, "get 12 experienced coders, divide up the work, let them all flog away for one month, and the problem will be solved. It's still 12 man-months, right?" Alas, time cannot be warped so easily. Dr. Brooks observed, while he was managing

[58] "Actually, 0.12 was out January 5th, and contained major corrections. It was in fact a very stable kernel: it worked on a lot of new hardware, and there was no need for patches for a long time. 0.12 was also the kernel that "made it": that's when Linux started to spread a lot faster. Earlier kernel releases were very much only for hackers: 0.12 actually worked quite well." Torvalds, Birth of Linux

the development of Operating System/360 (OS/360) in the early 1960's, that man-months are not -- so to speak -- factorable, associative, or commutative. 1 programmer x 12 months does not equal 12 programmers x 1 month. The performance of programming teams, in other words, does not "scale" in a linear fashion any more than the performance of multi-processor computer systems. He found, in fact, that when you throw additional programmers at a project that is late, you are only likely to make it *more late*." [59]

In another revealing book review, Frank Chance condenses the Brooks treatise on software production by providing the following analysis: "Remember, these words were written in 1975. Do they still apply today? Consider Windows NT 5.0. First scheduled to appear in 1997, it slipped to early 1998, to late 1998, then to 1999 (whence it was renamed Windows 2000). Here are some public estimates: 5,000 programmers = 35,000,000 lines of code

Clearly, NT 5.0 qualifies as a large-system programming project. And just as clearly, Brooks' tar pit is as prevalent today as in 1975! Let's carry on with the NT 5.0 example. Assume the worst case, that all 35,000,000 lines are new code. It's reasonable to assume that the development started in roughly 1994. So we have:

5,000 programmers x 5 years = 25,000 programmer years
35,000,000 lines of code / 25,000 programmer-years = 1,400 lines per programmer/year.

If you are a programmer, or have ever taken a programming class, this number (1,400 lines per year) seems amazingly low. Most of us have hacked together a bit of code that approaches one thousand lines in just a day or two. How could it possibly take a Microsoft programmer an entire year to complete 1,400 lines? Two possibilities spring to mind:

Microsoft hired 5,000 incompetent programmers to develop NT 5.0 or
it's a lot harder to write a large-scale programming systems product than to hack together a single program. Brooks would argue that the latter answer is the correct one. He starts by defining terms:

(1) A program: An individual program is the result of our two-day programming binge. It is ready to run by itself, on the machine where we wrote the code. If we add documentation, generalize the code, write test cases, and make the code maintainable by a general programming audience, we have:

(2) A programming product: Alternatively, if we take our program and completely define its interfaces according to a predefined specification, and test its interaction with a large number of other components, we have:

[59] Duncan, Ray. ERCB: The Mythical Man-Month. 26 May 1996. Electric Review of Computer Books. 15 Apr. 2004 <http://www.ercb.com/feature/feature.0001.html>.

(3) A programming system component: And if we do both (add documentation, generalize the code, write test cases, make the code maintainable, define its interface, test its interactions), we have:

(4) A programming systems product component
Brooks uses a 3x rule of thumb for the work required in taking each of these steps:

(2) = 3 times the effort of (1)
(3) = 3 times the effort of (1)
(4) = 9 times the effort of (1)

Or, in other words, development of a stand-alone program requires only 1/9th the effort required to develop a component in a programming system. Returning to our Microsoft example, if we apply this 9x factor to the 1,400 lines per programmer-year productivity measurement, we get 12,600 lines per programmer-year (e.g. if we took each of those programmers and set them to work in isolation, hacking away on a single program).

In another measure of just how little has changed since 1975, Brooks quotes an estimate of 1,000 lines per programmer-year. If the 1,400 lines per programmer-year quoted above is accurate, it represents a productivity gain of just 1.75% per year for the 20 years between 1975 and 1995. This result confirms another of Brooks' hypotheses -- that productivity of programmers is relatively constant, no matter the language used for development."[60]

Since its release in 1975, Brooks' analysis has been widely debated, but has managed to survive the test of time. Industry acceptance of the Brooks model is does not exist because of convenience, but because it has been useful over the decades. Considering the very first question about the invention of Linux, why so many companies with competent programmers chose to license Unix from ATT rather than write one, after consideration of Brooks explanation, it becomes very apparent that building a stable operating system would require too many man/months and years.

Brooks differentiates between creating a mere "individual program" that can be designed and written in isolation vs. building a "programming systems product component", i.e. an operating system, which Brooks demonstrates takes 9x the effort. Brooks' analysis also explains why accomplished individuals such as Professor Andrew Tanenbaum took three years to develop Minix.

[60] Chance, Frank. Book Reviews: The Mythical Man-Month by Frederick Brooks Jr.. 15 Apr. 2004 <http://www.fabtime.com/man-month.shtml>. Reprinted with permission.

4.5 Raymond, Brooks, and Linux

Almost tellingly, Eric Raymond, author of the 'Cathedral and the Bazaar', recognized by many as an open source extremist, insists Brooks' Law is obsolete. Raymond's treatise argues that a community of programmers can contribute enough effort, quality time and code to produce a stable operating system. Raymond writes in his essay, "if Brooks' Law were the whole picture, Linux would be impossible."[61] Indeed, even with a shortage of facts, history, and witnesses, Brooks' analysis suggests that the invention of the Linux kernel within six months was highly improbable. (See Diagram 11). As we see in this chart, looking at the wide disparity in Brooks law, the exact truth about the Linux development project is not only anecdotally questionable, but mathematically questionable.

The widely accepted Brooks model not only helps us understand the scarcity of operating systems, but why one of the largest investments in building software is the cost of labor. Meanwhile, Tanenbaum, an exceptional programmer, meets the debated range of Brooks theory, producing 1,500 to 4,000 lines of code per year. While the inexperienced programmer, Torvalds, announces a release of a stable kernel for an operating system within months, at a programming rate of 24,000 lines per year. With this significant disparity, it is critical that for Raymond to position his widely accepted theories in his treatise Cathedral and the Bazaar, he must vehemently disagree with the computer scientist Albert Brooks.

4.6 The Problem with the Credits Files

Linux has an intriguing set of credits files[62]; logs of the original contributors to Linux and Linux applications (see Diagram 9). These files however present significant attribution issues as well:

Problem #1 Ownership

From the Linux V0.01 release in 1991 through March 1994, there are no documented credits files for any of the code in the software. Thus, without accurate attribution records, much of the early kernel's ownership could be challenged. The code could have been contributed without permission for distribution. It is questionable whether Linus could prove or disprove who the developers were during any period in Linux development because many of the files have expired email addresses and incomplete information.
(See Diagram 8 for an example.)[63]

---

[61] Brooks Law and Open Source: The More the Merrier?, http://www-106.ibm.com/developerworks/linux/library/os-merrier.html. <http://www.free-soft.org/literature/papers/esr/cathedral-bazaar/cathedral-bazaar-10.html>.

[62] See Notes (Section XII)
Tuomi, Ilkka. The Evolution of the Linux Credits File. 3 Apr. 2004
<http://www.jrc.es/~tuomiil/articles/EvolutionOfTheLinuxCreditsFile.pdf>.

[63] AdTI interview with Ilkka Tuomi, March 26, 2004

Problem #2 Permissions

The credits files in from 1991 to 2000 do not credit Tanenbaum nor Prentice Hall. To this day, there is no record of any rights to use Minix code or permissions to sell a commercial derivative of Minix. Just to note, versions of early Linux do not credit Unix either.

Problem #3 Who Can Defend Linux in Court? Who Owns Linux?

It is questionable whether 1) a case challenging or defending copyright can be enforced or heard without the owner of the original work. 2) The kernel cannot defend itself in a copyright case. It must have an owner (or owners) to defend itself. 3) Conversely, if the owners of Linux are challenged for copyright infringement, who becomes liable?

Diagram 11

Brooks' Law and Operating System Development

	Developers	Lines of Code	Time Spent	Code Written/Year
Unix	Ritchie and Thompson	11,000	4 years	2,750
Minix	Tanenbaum, Vrije Faculty, Bruce Evans	12,000	3 years	4,000
Linux 0.01	Linus Torvalds (Solo?)	8-12,000[64]	6 months	16-24,000

[64] Analysis of the kernel length depends on "interpretation". Many comment the first kernel was approximately 12,000 lines. This chart demonstrates how the line count is debatable. AdTI provides the following chart after a code analysis of the kernel to present different reasoning to calculate the kernel length. Note that V 0.12, released in less than a year of the April start date, is at an even greater code production rate of over 32k lines of code/year. This is also significant because Linus claims the Jan 1992 V0.12 as the first fully operational kernel design.

OS	lines of c	lines of c and assembly	lines of c and assembly, counting blank lines
linux-0.01	7574	8933	9877
linux-0.11	10232	11453	12666
linux-0.12	14059	15420	16914
linux-0.96c	29418	29719	32943

Diagram 12

Examples of Linux Credit Files

Ananian complete credit file for Linux v.2.5.59

"C. Scott Ananian

74 E: cananian@alumni.princeton.edu
75 W: http://www.pdos.lcs.mit.edu/~cananian
76 P: 1024/85AD9EED AD C0 49 08 91 67 DF D7 FA 04 1A EE 09 E8 44 B0
77 D: Unix98 pty support.
78 D: APM update to 1.2 spec.
79 D: /devfs hacking.
80 S: 7 Kiwi Loop
81 S: Howell, NJ 07731
82 S: USA"

Schmitz incomplete credit file for Linux v2.5.59[65]

Michael Schmitz

2783 E:
2784 D: Macintosh IDE Driver

[65] Linux 2.5.59 Credits File. 15 Apr. 2004 <http://www.iglu.org.il/lxr/source/CREDITS?v=linux-2.5.59>.

4.7 IP, Money, and Other Considerations

It is also important to note that if motivated parties with the power of subpoenas, witnesses, interviews, and evidence delved deeper into the development of Unix to Minix to Linux (UML) there are a number of reasons why there could potentially be problems.

1) U.S. Code

U.S. stipulates that a copyright owner has 70 year right of enforcement to any owned copyrighted material.[66] That means these issues or any other regarding copyright, ownership or attribution could still be heard. It is very probable that Prentice Hall is not interested in pursuing this issue. Nevertheless, should they decide to, any valid legal question of rights and attribution of Minix in Linux would undoubtedly change all future Linux use and development. Because this legal question is so unclear, it is very real issue.

2) Overlap

The time frames overlap too questionably. Unix code and Minix code are governed by strict licenses by ATT (1969 – forward) and Prentice Hall (1987- 2000) during the period of active Linux development. As a different issue, although the Prentice Hall license changes to the less restrictive BSD license in 2000, this does not restrict Prentice Hall from researching whether there was a copyright or permissions violation in the earlier period.

3) Corporate Interest

Prentice Hall was focused on selling books, not on source code. Thus, their focus was not on the rights to the source code, but to the royalties from book sales derived from including Minix code.[67] Prentice Hall, however, made a decision to invest in the Minix text book with the expectation that the popularity of the Minix source code would either a) increase its book sales due to demand for the source code and Minix or b) increase its books sales due to the popularity of the source code and Minix.

But to this day, Linux, a product known virtually around the world, still does not properly credit Minix for its source code, its derivative use or its influence. Arguably, this has cost Prentice Hall considerable book sales from the years 1987 to present. In addition, it also obviously cost Prentice Hall sales between 1987 and 2000. One reason is due to the loss of customers that would have bought the Prentice Hall publication for the Minix code. Instead many decided

[66] Strasser, Mathias. "A New Paradigm in Intellectual Property Law?" Stanford Technology Law Review (2001): p.20. 19 Apr. 2004 <http://stlr.stanford.edu/STLR/Articles/01_STLR_4/article.htm>. p.20.
[67] AdTI interview with Tanenbaum, March 23, 2004

instead of buying the Tanenbaum book for the Minix code, they could get a free copy of Linux.

The following posting is an example of Linux decreasing Minix sales:

- *Subject*: Re: Minix VS Linux
- *From*: cwr@pnet01.cts.com (Will Rose)
- *Date*: 3 Feb 92 14:16:11 GMT
- *Newsgroups*: comp.os.minix
- *Organization*: People-Net [pnet01], El Cajon CA

> "Someone, either here on this newsgroup or over on alt.os.linux, made a very valid observation: the cost of a 16 MHz 386SX system is about $140 more than a comparably equipped (in terms of RAM size, display technology, hard drive space, etc.) 8088 system. Minix is $169. In economic terms, Linux wins if you have to buy Minix."[68]

Hypothetically, at $100 per book, at a loss of just 500 book sales a year, to date, Prentice Hall and Tanenbaum have lost almost $1,000,000 in revenues. This is of course only represents compensatory damages, not punitive. Arguably, Prentice Hall has lost out on tens of millions of dollars.

This formula becomes exponentially relevant considering also that:

a) If Minix credited as the primary source of Linux, the value of all Prentice Hall products would have received exceptional benefit

b) If Prentice Hall and Tanenbaum had received their due credit, it is unquestionable that this might have led to new book contracts, consulting and other new products

4) University Interest

It is arguable that Vrije University, including its foundation and its staff, have lost out on considerable notoriety that could be translated into financial loss (in terms of enrollment, grants, gifts, etc.) due to the lack of credit, permission and attribution of the Vrije Computer Sciences Department for the development of Minix that enabled Linux development.

This is not irrelevant, particularly because Helsinki University actively promotes itself as the home of the inventor of Linux.[69]

[68] Rose, Will. "Re: Minix VS Linux." Online posting. Feb. 3, 1992. comp.os.minix. Apr. 25, 2004. <http://www.ibiblio.org/usenet-i/groups-html/comp.os.minix.html>.
[69] "Linux was invented here." University of Helsinki. CS Dept. page, <http://www.cs.helsinki.fi/linux/>.

4.8 Considering Pre-emptive Argumentation

A) Linus, his fans, and the media misspoke…

Linus is not the inventor, he is the coordinator or organizer of the first Linux kernel.

Problem

Linus posts the first kernel in 1991 without any attribution or contribution from anyone else. This work has to come from somewhere. It is either his, or someone else's. If it is his work, we must still ask how a student with almost no operating systems experience can write a POSIX compliant kernel without use of any Unix or Minix source code, any experience, in such an incredibly short amout of time. If attribution belongs to someone else, we have a number of problems…including the fact that someone at any time could come along and claim it as their invention.

B) Why hasn't Prentice Hall or anyone else challenged Linus?

If the Linux kernel was truly an intellectual property problem, people would have discovered this years ago…

Problem

#1) It is irrelevant whether Prentice Hall or anyone files a legal complaint. The issue is whether they have a case at all. If someone were to substantiate the fact that the early kernel was a derivative work of Minix, the Linux kernel is in violation of explicit Minix copyright permissions from 1987 through 2000.

#2) If Prentice Hall at any time decided to argue that Linux has not properly credited Minix, they would also a have legal argument not just based on the engineering question, but on the question of attribution.

C) This issue is irrelevant. It just goes away, right?

That is the problem--- without enough facts to support either side, the origin, rights and legality of the Linux kernel remains a very serious question.

Section V. Tuomi's Theory of Attribution and Invention

5.1 Further Explanation

After examination of these and other issues, these events still present an array of inconsistencies in the history of Linux. This is because the "invention" of Linux is a discussion not just about the facts, but about the way history is recorded, especially regarding invention and technology.

Ilkka Tuomi in his book, "Networks of Innovation," demonstrates how the utility of the Internet directly influenced technological development. While Tuomi provides an exceptional amount of research on the process of innovation, interestingly enough, Tuomi dedicates an entire chapter to questioning attribution. As a purist, Tuomi insists that to truly understand innovation, we must have a correct history of invention and understand the aberrations of attribution. Discussing the relevance of history, Tuomi writes, "...This is theoretically important when we analyze the loci of innovation, but it is also has important practical implications, for example, for attribution of intellectual property rights. When we discuss the actors and agents of innovation, their importance depends on historical reconstruction."[70]

In his chapter, "Retrospection and Attribution in the History of Arpanet and the Internet", Tuomi introduces readers to the assortment of ways the history of invention is "reinvented." We will notice how Tuomi's theory aptly applies to the arrival and credit for early Linux.

5.2 Rembrandt, Minix, and Linux

> "MINIX was the base that Linus used to create Linux. He also took many ideas from MINIX, including the file system, source tree, and much more."[71]
>
> Andrew Tanenbaum

Appropriately, the first section of Tuomi's treatise discusses the question of original vs. copied work. With or without an author's blessing, copies of great works are abound. Tuomi writes, "authorship and reputation are always assigned retrospectively. At the beginning of the twentieth century, it was estimated that there were about 1,000 paintings by Rembrandt. Since then, the number has been cut down to 700, then to 630, and to 420. Ongoing research by the Rembrandt Research Project in Amsterdam will cut the number further. Many of the well-known 'Rembrandt' paintings were not painted by Rembrandt."[72]

[70] Tuomi. "Networks of Innovation", pg.
[71] Tanenbaum, Andrew. Interview with AdTI. March 8, 2004.
[72] Tuomi. "Networks of Innovation", pg. 154

Tanenbaum is a proponent of technology, particularly, operating system development. Reminiscent of Rembrandt, Tanenbaum does not challenge does not challenge Linux. Like Rembrandt himself, Tanenbaum almost encourages people to validate Linux commenting, "I think he (Linus) could have developed it in six months…I really don't have the interest to thoroughly compare the programs…"[73] To this extent, like Rembrandt, Tanenbaum's silence is a de factor endorsement of Linux. Without expert evaluation available to the public, faux Rembrandts are adored as originals. Likewise, Linux is reported as an invention, not a faux product.

It is interesting to note that Tanenbaum and Rembrandt have another trait in common--- Rembrandt actually signed work by his copiers. Some theorize he did it because he was flattered. Likewise, Tanenbaum actively defends Linus Torvalds. Even though the professor knows he used his work, everybody else knows he used his work, and everybody knows that Linus claims the work as his own, Tanenbaum actively praises Torvalds.

In a discussion about this research topic with a Dutch furniture executive, he humorously commented, "I guess it could happen…you could have people inventing new operating systems, copying Unix and calling it their own… If noone says anything, why not. I guess you could describe a faux Unix program as "Funix". Funix for faux Unix development…", he joked.

Ironically, after thorough consideration, reasonable minds have to conclude that either 1) Unix forked, without original licensed Unix code or 2) Unix never forked…and its free version is merely 'Funix'.

5.3 The Eliadian Election of Linus Torvalds

"Hi, I'm Linus Torvalds and I'm your God."[74]

If you ask 100 programmers, "Is Linux based on Minix/Unix", most will say yes. But if you asked the same 100 programmers, "Could Linus Torvalds have written the product from scratch without looking line for line at Minix or Unix source code", depending on their politics, you will get a range of non-answers.

Tuomi suggests the reinvention of history also occurs because, in a vacuum, sometimes we "look" for discoverers, inventors, even when we know it is mythical. We know it is false, but are culturally secure in a "deliverer". Tuomi writes, "Mircea Eliad argued that in traditional cultures, events become meaningful to the extent that they repeat mythical archetypes…" Tuomi

[73] Tanenbaum, Andrew. Interview with AdTI. March 8, 2004.
[74] Linux Expo Keynote Speech (Linus Ramblin'). Linux Expo '98, Raleigh, North Carolina, May 29. 1998.

continues, “As Eliad showed, recall of meaningful stories quickly fills in the missing details and puts the actors in their expected roles.”[75]

Tuomi’s theory of attribution explains why the conspicuous nature of the invention of Linux takes hold. The programming community wanted a free, non-Unix operating system and Torvalds gives them one. As Tuomi insists, an invention is well-received, particularly when society needs one.

Interviews with many of the most respected people in the IT industry manifest the Eliadean election of Linus Torvalds. While it can be expected that any individual would respect and admire Linus Torvalds, it is almost unexplainable (but for Eliadean theory) why so many accomplished individuals would rather contradict themselves, than to question whether there was anything questionable about the invention of Linux. Excerpts of this AdTI interview with widely respected programmer Fred N. van Kempen provides examples of reticent, inconsistent opinions about the origins of Linux within the open source community:[76]

> **Clarify for me what it means when someone says: I wrote an OS from scratch? I wrote a kernel from scratch?**
>
> A kernel is the core component of an operating system… it is what drives the hardware, creates and schedules processes (programs), has the driver software for all the connected devices, and so on. An operating system is that, plus the programs that come with it, the documentation, the tools to create new programs (editors, assemblers, compilers, linkers), games, and so on. As you can see, it takes a lot more work to "write an OS" from scratch, simply because this is a lot more stuff.
>
> **If you have the Unix code or Minix code, or even look at it even once, legally it is not from scratch right?**
>
> It is, just not necesarily a clean-room implementation, meaning, there could have been no way you could have copied anything over.
>
> **Eric Raymond says in Cathedral and Bazaar, "Linus Torvalds, for example, didn't actually try to write Linux from scratch. Instead, he started by reusing code and ideas from Minix, a tiny Unix-like OS for 386 machines. Eventually all the Minix code went away or was completely rewritten.." What is your reaction to this? If this is remotely true...why did PH sue you and not Linus?**
>
> He did not. Eric often doesn't exactly know what he is talking about, especially when it comes to legalese. Linus, like me, wanted Minix to use the 80386's full capabilities. Minix was an OS for the 8086/186 and 286,

[75] Tuomi, Ilkka. “Networks of Innovation”.
[76] Fred N. van Kempen interview with AdTI, April 28, 2004

not the 386. But to use the 386 to its fullest, the whole kernel had to be redone, and in ways that would make it a completely new thing, incompatible with the system as it was…So, Linus started to play with things. He DID initially use some Minix code, but then decided that was not the way he wanted things to run, so restarted again, and from scratch this time... his famous aaaabbbb kernel code (which alternated between printing 'a' and 'b' characters on the screen.) So..I assume PH let it go (if they knew about it at all..) just because that lasted only a short while.

Thanks for the clarification on Eric Raymond...but you know that was in an interview with Linus and he never denied it...

Hmm. I don't get the follow-up comment either... 'I'm basically a very lazy person who likes to get credit for things other people actually do.' Although I know he said that more often, I don't think Linus is "that way". I have met Linus many times, and even though we have very different views on things, I don't think he would knowingly take credit for something he did not do.

And some media reports that Linus says he was the sole inventor…

Well, anyone who knows anything about programming, especially the art of OS design and programming, knows one does not "invent" an OS. You grab, design and gobble together bits and pieces, and then shape them into a single entity: the OS. Linux is "just another UNIX derivative", so Linus sure as hell did not "invent" it. He wanted to create a 32-bit UNIX-like system (alike Minix for the earlier Intel chips) that was optimized for the then-almighty 386. He started with Minix as a base, then dropped that and restarted from scratch, but still using the ideas he took from elsewhere. So:

- Linus did not "invent" Linux
- Linus did "write from scratch" Linux, based on many ideas already existing out there.

Now Linus says he did not have the Unix code…He did not have the Lions book.

He probably did not. I did, and so did some of the other now-gone Old-timer kernel coders. I assume he did not, but can't be sure. I do have it.

5.4 "Matthew Affected"

In January 5, 1968, Science Magazine published "The Matthew Effect in Science" by Robert K. Merton. Merton introduces the idea that the history of

invention is often disingenuous and very hard to pinpoint because credit often moves away from true inventors to individuals with bigger reputations, magnetic personalities, or significant prior accomplishment. Tuomi writes, "Robert Merton introduced the concept of the 'Matthew Effect'[77] to describe the allocation of credit among authors of multiple discoveries or collaborators, arguing that the 'rich are likely to get richer'…."[78]

Case #1 The Matthew Effect and Tanenbaum

While Tanenbaum had tens of thousands of programmers interested in Minix, the genius and experience, he was not interested in building a super product for commercial use. The Netherlands professor also had the additional handicap of being a tad flippant and dismissive. Tanenbaum is widely recognized as a genius, not a political revolutionary. Almost without resistance, Tanenbaum is not only summarily pushed away from any credit of Linux, but was publicly put down by the young Finnish upstart:

> From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
> Subject: Re: LINUX is obsolete
> Date: 29 Jan 92 23:14:26 GMT
> Organization: University of Helsinki
>
> Tanenbaum: As most of you know, for me MINIX is a hobby, something that I do in the evening when I get bored writing books and there are no major wars, revolutions, or senate hearings being televised live on CNN. My real job is a professor and researcher in the area of operating systems.
>
> Torvalds: You use this as an excuse for the limitations of Minix? Sorry, but you loose: I've got more excuses than you have, and Linux still beats the pants of Minix in almost all areas. Not to mention the fact that most of the good code for PC Minix seems to have been written by Bruce Evans….your job is being a professor and researcher: That's one hell of a good excuse for some of the brain-damages of Minix. I can only hope (and assume) that Amoeba[79] doesn't suck like Minix does.

Case #2 The Matthew Effect and Stallman

At the time, there was one reticent but persistent programmer that did decide to challenge the attribution and credit of the Linux kernel development. As noted by Merton, the Matthew Effect argues persuasively that regardless of genius, an inventor that receives more recognition is likely to get more recognition. Conversely, an inventor that receives little credit, is likely to get even less. Albeit

[77] Merton's reference to Matthew is from the passage in the New Testament, Matthew 25:14-30 the 'Five Talents'…For to every one who has will more be given…
[78] Tuomi, Ilkka. "Networks of Innovation".
[79] Amoeba, a time-sharing system (operating system) written by Tanenbaum in 1989. More info at: www.cs.vu.nl/pub/amoeba; www.en.wikepedia.org/wiki/Amoeba

a genius and the biggest contributor technologically to early Linux, we will observe how Richard Stallman is firmly prodded out of the history books by Linus Torvalds.

The Linux kernel needed a compiler program[80]. The only way that Linus and his team could know if any of their work was functional, was to be able to run it through a compiler. Around this time, it was well known that for years, Richard Stallman, founder of the GNU Project was working on an operating system called GNU. His effort was also to write a non-Unix operating system from scratch and make it freely available. He too needed a compiler program to make it work. The compiler he built, called the GCC, was a program of significant complexity and size, reportedly over 110,000 lines of code.[81] Richard Stallman had completed the GCC compiler but had not finished the operating system.

Linus Torvalds needed a compiler program to help him with Linux and keep his loosely assembled team of programmers interested in debugging the Linux kernel. The exact terms of the partnership between Linus Benedict Torvalds and the MIT genius[82] are unclear, besides the fact that the arrangement would be that the new project consisting of GNU software and the Linux kernel would be called GNU/Linux. For some reason, this idea devolved, and Stallman's work was only attributed to the project as a supporter, but not anyone that substantively contributed to Linux.

But there is a greater irony to the Stallman story. Stallman began writing software for the GNU operating system in 1984. He completed a number of programs including the GCC compiler by 1990. But he was unable to finish an operating system. Widely recognized as one of the most vigilant and talented programmers in the world, Stallman confesses to the enormous amount of work required to build an operating system kernel, writing:

"By 1990, the GNU system was almost complete; the only major missing component was the kernel. We had decided to implement our kernel as a collection of server processes running on top of Mach. Mach is a microkernel developed at Carnegie Mellon University and then at the University of Utah; the GNU HURD is a collection of servers (or ``herd of gnus") that run on top of Mach, and do the various jobs of the Unix kernel. The start of development was delayed as we waited for Mach to be released as free software, as had been promised. One reason for choosing this design was to avoid what seemed to be the hardest part of the job: debugging a kernel program without a source-level debugger to do it with. This part of the job had been done already, in Mach, and we expected to debug the HURD servers as user programs, with GDB. But it took a long time

[80] "A **compiler** is a computer program that translates a computer program written in one computer language (called the source language, human readable format) into a program written in another computer language (called the output or the target language or binary, machine code)." Source: Wikipedia: http://en.wikipedia.org/wiki/Compiler
[81] Bezroukov, Nikolai. Portraits…
[82] In 1990, Stallman was awarded a $240,000 fellowship by the John D. and Catherine T. MacArthur Foundation, often known as a "genius grant."

to make that possible, and the multi-threaded servers that send messages to each other have turned out to be very hard to debug. Making the HURD work solidly has stretched on for many years… The GNU Hurd is not ready for production use. Fortunately, another kernel is available. In 1991, Linus Torvalds developed a Unix-compatible kernel and called it Linux…"[83]

After years of development, Richard Stallman, devout programmer, tech activist, and respected programmer does not produce a kernel. Stallman's journal provides another series of reasons to question the origin of Linux considering :

1) To save on time and effort, the GNU team confesses to using Mach, a pre-developed kernel from Carnegie Mellon University, a licensee of Unix.

2) The GNU multi-developer team could not seem to build a kernel as fast as the solo effort of Linus Torvalds.

3) Although building a free operating system was its primary intention, the GNU team builds a number of other software products more easily than they develop an operating system kernel.

The following is an excerpt from a brief AdTI interview with Richard Stallman on this subject:[84]

> **What were the problems that you faced in your personal endeavor in the 80's to develop a kernel?**
>
> There may be some confusion here, because I would not say that that "personal endeavor" was to write a kernel. The project I launched in 1984 was more than a kernel. It was to develop a complete free operating system: GNU. To achieve that goal we need many programs, including a kernel. The development of our kernel, the GNU Hurd, was a subgoal of the larger goal. GNU Hurd development started in 1990.
>
> I never wrote any code for the Hurd myself. I chose the general idea of the design, but after that, the work was done by others.
>
> **In all of the outside help that you had in creating a kernel for your GNU Project, why do you think it took so long to finally reach completion?**
>
> The GNU Hurd runs, but I would not say it has "reached completion" even now. It still needs more work. Your main question is why it took so long. I do not know for certain, but here are some of the factors that contributed.

[83] Stallman, Richard. The GNU Project. 1998. 8 Apr. 2004. <http://www.gnu.org/gnu/thegnuproject.html>.
[84] AdTI interview with Richard Stallman, May 7, 2004

- It was somewhat of a research project, and some things had to be redone once or twice base on what the developers learned.
- Debugging multithreaded asynchronous programs is hard.
- The Mach development environment we were using was not as reliable or easy to use as we expected.
- The GDB developers did not cooperate very well with Hurd development.

Why did you feel that Linus' kernel was the right fit for your situation?

I never felt that way. What happened is that other people combined GNU and Linux, and the GNU/Linux combination worked and became popular, so we started using it too.

Do you have any comments regarding the relatively quick time (reportedly six months) it took an unknown Finnish graduate student with just a semester's experience with C to create an operating system..?

He did not create an operating system. He wrote a kernel. What Linus released in 1991 was not a mature kernel, it was barely a functioning kernel. It took a couple of more years for him to arrive at a kernel with functionality comparable with the kernel of Unix.

Nonetheless, it is true he got Linux to work in an amazingly short time, much less time than the Hurd needed. My only comment on that is that he clearly a good programmer.

Case #3 The Matthew Effect and the Credits Files

Another interesting perspective on the credits files is the limited credit to members from developing countries in the Tuomi chart.[85] This can be explained away by simply suggesting that non-English speaking countries would have been slow to show interest in Linux development. However, by 2000, although it is widely known that China and India are heavy Linux developers, they both receive an insignificant amount of credit in the Linux credits files[86]. In fact, India, an English-speaking country, is non-existent, while countries such as Mexico, Brazil, and Argentina are recorded with minimal presence. Amusingly, while the Tuomi chart studies Linux credits from 1991 to 2000 from over thirty countries, according to Tuomi's study of the credits files, Finland per million inhabitants, remains the number one source of original Linux code for the ten year project.

[85] Tuomi country chart. See notes section.
[86] Linux 2.5.59 Credits File. 15 Apr. 2004 <http://www.iglu.org.il/lxr/source/CREDITS?v=linux-2.5.59>.

It is almost certain that Tuomi, a scientist and rigorous researcher, does not introduce this data to argue that there may be country bias in the credits files. However, Tuomi's point "History has a very selective memory…" could be relevant in this instance as well. After all, the Matthew Effect historically has been very effective in purging the origin of invention from developing countries for many years. We don't have any evidence that it occurred with Linux. However, it is conspicuous that an open source model, touted to uplift developing countries, does not seem to have contributions from the very countries Linux advocates are arguing they are interested in promoting.

5.5 Eponymy Theory and Naming an Inventor

> *"They put it together and called it Linux without acknowledging the work that came before," says Stallman. "I think it's unfair to call our work by someone else's name." While Stallman concedes that Torvald's contribution was essential, he estimates that the kernel represents only about 3 percent of the entire system. In contrast, the GNU project contributed about 30 percent of the code, while the remaining 67 percent was taken from other sources, he says.*[87]

Tuomi writes, "… The problem of allocating authorship is clearly seen in the phenomenon of eponymy. Eponymy associates a specific idea, phenomenon, or result with a (single) person as in Gaussian distribution, Planck's constant, Halley's comet, Rorschach test, or Tobin tax. Based on his studies on the history of statistics, Stephen Stigler proposed his own 'Stigler's Law of Eponymy'. In its simplest and strongest form it says: 'No scientific discovery is named after its original discoverer'…"

After looking at the evidence, with Tanenbaum and Stallman as "unelected" representatives of Linux, leadership moved to Linus. Linus is duplicitous about this point.[88] He waffles on whether he minds the term GNU/Linux. However, he also doesn't concede the name to Stallman. Finally, he goes as far as buying the trademark to "Linux".

While history is still unclear about Linux authorship, Tuomi's theory of habitual eponymy provides us an explanation why Linus Torvalds is hastily identified as "the" inventor, as opposed to "one inventor of many".

[87] Linux's Forgotten Man, Wired, March 5, 1999
[88] Loli-Queru, Eugenia. Interview with Linus Torvalds. Oct. 10, 2001. Apr. 15, 2004 .<http://www.osnews.com/story.php?news_id=161>.

Section VI. A Closer Look at the Art

6.1 On Copying A Rembrandt

Software is art, and Unix, unquestionably, is a Rembrandt. From the Lions incident in 1977, to questionable attribution in Linux, Unix is undoubtedly the most licensed, imitated, and stolen product in the history of software. We note that almost thirty years later, the Ritchie, Thompson invention is hailed as genius not just because of its capability, but because of the amount of time and expertise it would take anyone else to build a product as capable. As we follow Unix history, almost every entrepreneur, academic and developer has preferred to either 1) modify Unix as a licensee and/or 2) build applications for the popular operating system. Regardless of the reason or circumstances, only a handful of people have chosen to build a new operating system from scratch. So even today, we notice a plethora of "copies" and a noticeable scarcity of originals.

Nevertheless, the question of "just-building-another-operating-system" is the most curious paradox in computer history. Building an operating system is not a question of specialized material or resources, but solely a question of labor. Today especially, there are more tools to build an operating system than there were ten and twenty years ago. If software is just programming, why haven't we seen as many versions of operating systems as we see of software applications? The answer is reflexive; there are thousands of different types of software applications because building and selling mere software is not as difficult as building and selling an operating system.

Raymond argues that for Brook's Law to hold true, "Linux would be impossible". But Raymond's point is not purely argumentative because Linux does exist. So we are left to ponder, could Raymond and Brooks both be right? First, regardless of Brook's Law or Raymond, the open source community has two very real dilemmas 1) they have to produce reliable, stable code as fast as the proprietary community and 2) and they have to accomplish this task without hundreds of millions of dollars in resources.

Linux is here, but Brooks law is also well-entrenched in the industry. So we are left to ask, "is there a process that speeds the development of quality software? Is there an infinitely cheaper way to develop quality source code?

A brief overview of this issue provides a controversial, but necessary look into the processes that enable programmers to take 'short-cuts'; processes that enable programmers to hastily and inexpensively produce quality source code. This discussion becomes the final proof that the 'source' of open source code is a matter of exceptional gravity.

6.2 Reverse Engineering

The capability and use of reverse engineering processes have been well documented for decades. "While a good deal of code decompilation is completely aboveboard, the fact is that a good decompiler is one of the essential tools of software piracy."[89] In addition to widespread knowledge of the practice, there has been a substantial amount of case law that has substantiated the capability of decompiling techniques to glean information, processes and source code from compiled software (binary or machine code) as the courts have ruled that there is both legal and illegal reverse engineering.[90]

With just a minimum amount of investigation, it becomes clear that the open source community openly and proudly engages in reverse engineering to produce competing products. Their reasons for doing this include: achieving competitiveness and interoperability. A Google search of 'open source + reverse engineer' produces over 1000 hits of products, articles, opinion, commentary, etc. by open source community activists embracing reverse engineering. (See Diagram 10)

Reverse engineering is real in the open source community and clearly an inexpensive tool to speed the development of "new" software.

6.3 Obfuscation Software

Obfuscation is an issue that makes the reverse engineering debate even more interesting. Today, open source advocates vehemently argue:

1) Source code theft and copying would end in an open source world.
If all software were open, everyone could see everyone's code. Any suspicion of software theft would be swiftly answered. Comparing the code line for line would produce uncertain doubt about whether the code was original, or pilfered.
2) Open proprietary code would easily reveal source code theft. (note Miller in Diagram 10). However, as it turns out, obfuscation techniques make both of these statements quite untrue. For decades, software programmers have written software that disguises, reconfigures and obfuscates code. Ironically, software

[89] Travis, Greg. How to lock down your Java code (or open up someone else's). 1 May 2001. 19 Apr. 2004. <http://www-106.ibm.com/developerworks/java/library/j-obfus/>.
[90] For example, last year in the reverse engineering case of Bowers vs. Baystate, the Supreme Court recently decided not to hear the accused software company's appeal after a lower court awarded the plaintiff and inventor Harold L. Bowers $5.27 million. http://www.infoworld.com/article/03/06/26/HNreverseengineering_1.html. Conversely, in March 2004, in CCA vs. Bunner, the DVD Copy Control Association lost to Andrew Bunner, a programmer that reverse engineered its DeCSS encryption code and posted it on the net. http://news.zdnet.co.uk/business/legal/0,39020651,39147906,00.htm

Diagram 13

Reverse Engineering and Open Source

Example #1 Awards for Reverse Engineering

Builder.com: What has been your biggest challenge?
SAM: Obtaining specifications for codecs and file formats. A lot of reverse engineering was involved and some of this reverse engineering is at the very border of the law in some parts of the world, and our biggest challenge may come from individuals or entities trying to stop us (using perverted legal systems, using the DMCA, the EUCD, etc.).[91]

Open Source Awards 2004: VideoLAN, by Sean Michael Kerner

Example #2 Books on Reverse Engineering

"…Furthermore, interoperability issues with closed-source proprietary systems are just plain annoying, and something needs to be done to educate more open source developers as to how to implement this functionality in their software...Reverse engineering as this book will discuss it is simply the act of figuring out what software that you have no source code for does in a particular feature or function to the degree that you can either modify this code, or reproduce it in another independent work….Why reverse engineer? Answer: Because you can…"[92]

From abstract of book, "Introduction to Reverse Engineering" by Mike Perry and Nasko Oskov.

Example #3 Stolen Windows Source Code

"… Windows 2000 and Windows NT4 source code has been leaked on the internet...there are a number of positive developments that can occur because of this:

Search MS source code for evidence of theft from open source projects;
Search MS source code for evidence of antitrust violations ("Windows isn't finished until * won't run"); Reverse engineer MS source code for greater compatibility with all sorts of open source projects; Reverse engineer MS source code for Windows emulators on Linux such as WINE or Lindows; Prove that the browser/media player/etc. can be removed from the operating system with little harm done;
Reverse engineer MS source code to crack DRM schemes;
Play with source code just because Bill Gates won't like it;[93]

From "Good News - MS Windows Source Code Leaked", by Ernest Miller.

---

[91] VideoLAN Project, the. Interview with Sean Michael Kerner. Builder.com. 15 Jan. 2004. 14 Apr. 2004 <http://builder.com.com/5100-6375-5136135.html?tag=search

[92] Oskov, Nasko, and Mike Perry. Introduction to Reverse Engineering Software. 19 Apr. 2004 <http://www.acm.uiuc.edu/sigmil/RevEng/>.

[93] Miller, Ernest. Good News - MS Windows Source Code Leaked. 13 Feb. 2004. 16 Apr. 2004 <http://www.corante.com/importance/archives/001877.html>.

designed to "protect" proprietary authors from having their code stolen in reverse engineering attempts can be used to keep open code cloaked. To keep source code completely hidden from users in a reverse engineering attempt, obfuscation code throws in lines, characters, etc. that completely disguise, seemingly readable code. (see Diagram 11) Although, the PC runs the program without a problem, the obfuscation technique cloaks the appearance of the code to the human eye, almost eliminating the ability to effectively discern anything about the product's design as well as its internal source code.

Ironically, while the open source movement actively embraces controversial processes such as reverse engineering, it insists that all source code stay un-obfuscated. For example, the source code section of the opensource.org website reads, "The program must include source code, and must allow distribution in source code as well as compiled form…The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed."[94]

Meanwhile, a Google search of "obfuscation software" produces dozens of references to popular software to limit the ability of reverse engineering. The dilemma is real. If reverse engineering technology is being used to questionably produce code, obfuscation software inevitably will be used to convince authors that their code was NOT indeed stolen or reverse-engineered.

The future is uncertain particularly because there is no trust on either side, as seen with the exceptional amount of source code/reverse engineering theft cases in the courts today. Thus, as long as obfuscation software becomes more and more advanced, opening source code will not deter, nor end feared source code theft.

In a ZD News story, reporter Paul Tyma, makes a similar point writing, "… In this context, it can be argued that obfuscation is stronger than encryption. Certainly data with strong encryption is practically impossible to decrypt, however, it also cannot be executed in that form and if the key is obtained, it is now in complete view. Obfuscation is a one-way lousy transformation that destroys the structure that reverse-engineering programs look for. The information that existed prior to obfuscation does not equivalently exist after obfuscation. Unlike one-way hashing, there is not a one-to-one relation between unobfuscated and obfuscated code. If possible, a brute force attack could find many (and under the right circumstances, infinite) possible correct "original versions" of the code. There isn't enough information to be sure which original version would be the right one. Given that even the tiniest manufactured logic error in heuristically-created code could crash an application, brute forcing to undo obfuscation isn't particularly viable."[95]

---

[94] Perens, Bruce. The Open Source Definition. June 1997. 19 Apr. 2004. http://www.opensource.org/docs/definition.php
[95] Tyma, Paul. Encryption, Hashing and Obfuscation. 8 Apr. 2003. 17 Apr. 2004

Diagram 14

Obfuscated Code

Disassembled intermediate code before control-flow obfuscation:

```
// Code Snippet copyright 2000, Microsoft Corp, from WordCount.cs
// sample app
public int CompareTo(Object o) {
  int n = occurrences - ((WordOccurrence)o).occurrences;
  if (n == 0) {
    n = String.Compare(word, ((WordOccurrence)o).word;
  }
  return (n);
}
```

Same code after control-flow obfuscation:

```
public virtual int a(object A_0) {
  int local0;
  int local1;

  local0 = this.a - (c) A_0.a;
  if (local0 != 0)
          goto i0;
      goto i1;
      while (true) {
          return local1;
          i0: local1 = local0;
      }
      i1: local0 = System.String.Compare(this.b, (c) A_0.b);
      goto i0;
}
```

Author Binstock writes, “as can be seen, a bogus test is inserted, then a goto is performed. At the goto destination, the original statement (in obfuscated form) is executed, then another goto statement returns control to the original branch in the logic flow. Notice the unexecuted and just misleading while() loop. In this small snippet, after close comparison with the original, it's possible to figure out what's real and what's not. However, on a large routine without the benefit of the source code, these misdirecting interpositions create a hugely time-consuming effort”.

Reprinted from destination.net newsletter[96]

[96] Binstock, Andrew. Obfuscation: Cloaking Your Code From Prying Eyes. 6 Mar. 2003. 19 Apr. 2004 <http://www.devx.com/SummitDays/Article/11351>.

6.4 Pretty Printers

A pretty printer is a reformatting software. Depending on the application, a pretty printer enables a programmer to significantly change the appearance and technical layout of a program. Pretty printers are widely used to cut down on work, for debugging and also for presentations, hence the name “pretty printer”. The process quickly eliminates the need for hours of recoding very quickly. For example, a pretty printer can convert HTML to XML or a pretty printer can convert a difficult to read program into a format that the programmer is more comfortable with.

With regards to copying a Rembrandt, the pretty printer is helpful because it can be used to disclose information about the program that is not easily apparent. A programmer can spend half their time just reading a program. A pretty printer quickly cuts down on the time spent trying to extract vital information about a program. Unlike obfuscator programs, pretty printers are useful in revealing pertinent information about the architecture of software. Also, with a pretty printer, you don’t need to ‘copy’ or rewrite code per se.

Reverse engineering and pretty printing often become closely related operations as well. To understand a program, it is essential to be able to read it. A decompiled program is often quite illegible, even if the programmer is familiar with the language it is written in. After the code is decompiled, it can be pushed through a pretty printer. A Google search for Pretty Printer will produce over a one thousand references to software that enable programmers to accomplish this task.

Even with the code, source code can appear exceptionally hard to read. The pretty printer software allows a programmer see a program as the PC sees it. A pretty printer doesn’t just print the original source code, instead it reconstructs what the source might have been from an internal representation that is actually used to run the software. This way, the programmer can see “hidden” variables available within a program. Pretty printers enable a programmer to transform any ‘foreign’ code or style to something that they can understand.

Reverse engineering is particularly difficult because it tends to produce code with discernable file structure, etc. Pretty printer programs are designed to put code in any format a programmer desires. Thus, a million lines of reverse engineered code becomes much more manageable and easy to decipher with a pretty printer program. Second, the fascinating thing about a pretty printer is that the most difficult piece of evidence to remove is often the physical appearance, data structures, and organization of the program. Specifically, if you copy a program and change the code, it still closely resembles the pilfered because the formats and structure are too close. However, a pretty printer helps to solve this problem.

6.5 Comparision Programs

These products are also widely available. Comparison programs provide a wide variety of comparison data about two programs. Comparison programs produce text data, line data, and mathematical information about the similarity of programs. After the art is copied, comparison programs easily double-check the new copy against its subject. This eliminates the need to compare a file manually. Comparison programs will compare files on byte-by-byte basis or compare them for structural similarities, etc.

The most obvious mistake in copyright infringement is to compare a program line for line a find a duplicate. This problem was discussed earlier by both Professor Tanenbaum and Linus. Each insisting that there is not a line of Unix in Minix or a line of Minix in Linux. (see p.20 for quotes on the matter) After the copy is made, a comparison program is the last, but vital check a programmer needs to make sure that there isn't stolen code line-for-line in another program.

This is important, because a comparison program can only be effective to check copying line for line. However, it is very limited its ability to determining whether product is derived from another.

However a programmer that would insist that their program was not identical in any format or line of code, has by default admitted to using a comparison program of some sort—this also means they it admit to having the code.

Diagram 15

Popular Formatting Software

Name	Reference	Company
Reverse Engineering Software		
CPPX	www.swag.uwaterloo.ca/~cppx/	SWAG[97]
CxRef	www.gedanken-demon.co.uk/cxref/	Andrew Bishop (FSF)
Headway Review	www.headwaysoftw.com/html	Headway Software
Obfuscation Software		
JCloak	www.sys-con.com	Force 5 Software
Jalopy	www.jalopy.sf.net	Marco Hunsicker
Stunnix Perl-Obfus	www.stunnix.com	Stunnix
Pretty Printers		
Grind	www.cs.bgu.ac.il	*Mayer Goldberg
Jindent	www.jindent.com	Software and Solutions
JRefactory	www.jrefactory.sf.net	*Chris Sequin
Comparison Software		
Beyond Compare	www.scootersoftware.com	Scooter Software
Active File Compare	www.formulasoft.com	Formula Software
Compare and Merge	www.compareandmerge.com	All Your Software

*

[97] Software Architecture Group (SWAG)

6.7 The Dilemma of Residual Risk

Whether any of these and other processes are employed for either defensive or offensive purposes, conspicuous duplication remains a real concern. As countless variations of formatting software become more readily available, the overall risk continues that persons who infringe copyrights will inevitably be caught. It can be expected that companies with significant interests in keeping their software code proprietary will continue to pour millions of dollars in developing processes to catch copyright infringement. Thus, while formatting tools are helpful, they are not foolproof and like the almost perfect Rembrandts, phonies will soon be discovered.

Roedy Green, encouraging source code owners not to worry about theft, lists over a dozen ways that would-be copiers can be slowed down considerably or caught. He writes, "Throw in red herring code that looks like some complicated security system but which is actually gibberish. Let the pirate waste hours discovering that. Riddle your code with a wide variety of anti-tampering devices. Camouflage them in as many ways as you can think of. The simpler the tampering test, the more it looks like ordinary code. Don't immediately react to a tampering detection."[98]

[98] Green, Roedy. Obfuscator - Java Glossary. 1996. 19 Apr. 2004. <http://www.mindprod.com/jgloss/obfuscator.html>.

Section VII. Sustaining Corporate Partnership

Greetings,[99]

I'm currently doing research into corporate contributors of open source software, such as Linux. According to a recent Linux Credits File, one of your employees, a Mr. Hannu Savolainen, is a contributor. Is Mr. Savolainen still an employee with your company?

Yes

Does your company have any policies regarding contributions to public open source projects?

Yes we contribute GPL'ed software - OSS drivers in kernel and XMMS (http://www.xmms.org) are our products that we have developed and gpl'ed.

We have a policy of releasing software under GPL only if it has no value commercially, ie, if a product cannot drive revenue from being closed source, we just release it as open source.

Is there any difference in company policy regarding work-time contributions versus free-time contributions?

We do have a policy that all software developed by our employees Whether during work or during free time is subject to productization/commercialization first and if there is no commercial value then we release it as free software. 4Front is an employee owned company and each employee has a stake in the company and therefore looks out for the company's interest first.

7.1 Code Moving In or Out of the Corporation

It is commonly known that most open source projects are supported by a vast pool of programmers that are full-time workers for corporations. In fact, when you look at the credits files, a significant number of the email addresses belong to corporate interests, many of whom sell closed, proprietary technology products.[100] The contradictory nature of this relationship we see in the interview with Dev Mazumber, having policies that "hope" employees do not contribute profitable code.

[99] Interview with AdTI, Dev Mazumdar, President, 4Front Technologies, May 19, 2004

[100] Examples include Monalisa Agrawal of Nortel Networks, Tim Alpaert of Toyota Motors Europe, and Matt Domsch of Dell. Source: Linux v2.5.59 Credits File.

Should this phenomena be of any concern to corporations? Surprisingly, corporations don't seem to agree. Charles Mills, a due diligence consultant comments, "As a software due diligence practitioner, I'm more concerned with "leakage" from open source to company product. If I am evaluating your company, and it looks like one of your employees "donated" some of your code to Linux, that's a problem, but it's a small problem…But if your programmer has appropriated open source code and incorporated it into your proprietary product, then potentially of all of your subsequent sales have violated the FSF's (Free Software Foundation) copyrights and/or license terms, and the value of your key asset --- your software --- may be substantially impaired. That's a problem."[101]

In disagreement, Henry Jones of Intersect Technology Consulting comments, "I know and work with plenty of companies that –do- permit such OSS (open source software) participation during working hours…"Smart companies allow talent to work on non-company projects (charity, civic, etc.). Smart companies are now developing robust OSS strategies processes, and staffing….Nobody's laughing at Richard Stallman any more."[102]

While the companies and their advisors may disagree, the magnitude of the issue still comes back to the questionability of the 'source' of source code. In either direction, there are two primary concerns about open source moving in/out of the corporation: 1) As Mills describes, a lack of corporate diligence at any point by an employee could corrupt closed source with protected open source. 2) Companies that decide to use code in the public domain are later subject to suit by a company that claims ownership of it.

Mills continues about the risk, "In my due diligence work, I am not seeing companies embracing the concept of employees working on OSS projects. I am seeing significant fear of OSS "contamination."[103] If your product and Linux have source code in common, do you have the resources to fight over which came first?"

7.2 The 'Source' of Volunteers

Software programming tools cost money, in some cases tens of thousands of dollars. In addition, many of the volunteers need the facilities and time to produce quality, competitive open source code. Undoubtedly, the contributors have years of training, expertise and resources. Thus, it can be expected that the natural place to find solutions to the need for free software is moonlighting programmers.

[101] Mills, Charles. Interview with AdTI. Apr. 14, 2004.
[102] Jones, Henry. Interview with AdTI. Apr. 14, 2004.
[103] Mills, Charles. Interview with AdTI. Apr. 14, 2004.

The labor force behind open source becomes a final topic regarding the 'source' of open source code. Open source uniquely depends upon both vendor and volunteer supplied source code. Mary Merrill of Merrill Associates wrote, "To gain a better understanding of the today's self-focused motivations, consider the Open Source community as an intriguing model of volunteer engagement. Open Source refers to a community of volunteer systems developers and programmers from all over the world who are committed to developing and delivering non-proprietary (free) software... This Open Source community may have much to teach us about creating models for engaging a new generation of volunteers... The Open Source movement is an amazing example of the volunteer structure - or lack of structure - and the incentives that are attracting and engaging a new generation of creative, innovative people."[104]

It is questionable, as outsourcing continues to seriously impact jobs in the IT industry sector, whether corporations can continue to produce "free" software. Outsourcing impacts that future of the model at a number of levels including:

1) The increase in competition shrinking "free" time of programmers on staff
2) Decreased interest in programmers to divert time to community projects

7.3 Sponsoring Competition

The consideration with the most gravity is whether companies can contribute and/or support a model that may put itself out of business. Open source continues to traverse every common software application in the sector. Will corporations' view of open source change?

In a story on Oracle and open source, David Banks comments, "…But now, Oracle and other database suppliers face a growing threat from below: "open source" databases, which give customers a free or low-cost alternative to commercial products. While the impact has been small so far, some analysts expect open-source software to eventually turn databases into a low-cost commodity, just as the open-source Linux operating system is posing a threat to Microsoft Corp.'s Windows franchise."[105]

The Oracle open source partnerships have been one of the most visible in the last few years. Ironically, Oracle's determination to partner with Linux to compete as a bundle with Microsoft and its database partners is convincing its customers that open source is not just a valid operating system product, but a database one too. Banks writes, "One user of an open-source database is Cox Communications Inc. The Atlanta-based cable-TV operator is using the software to monitor the performance of more than 1.5 million cable modems providing customers with high-speed Internet access…For Cox's system, the price for

[104] Merrill, Mary V. Being a Gatekeeper. Apr. 2004. 19 Apr. 2004. <http://www.merrillassociates.net/topicofthemonth.php>.
[105] Bank, David. 'Open Source' Database Poses Threat to Oracle. 9 July 2003. 19 Apr. 2004 <http://webreprints.djreprints.com/785490482991.html>.

licensing Oracle's system would have totaled about $300,000, not including a service contract. But even after Cox upgraded to the commercial version of the MySQL database, the company's licensing costs were under $1,000. Mr. Cotner also pays $12,000 a year for support services."

Shankland of CNET writes this year, "Open-source databases are in the experimentation phase of the market but will move to widespread acceptance by 2006," the AMR Research study said. AMR surveyed 140 information technology managers in December and released results this month."[106]

Again, however, the speed to market of stable, cheap, open source alternatives remains a conspicuous issue. Examples include, the open source MP3 format development. Fraunhofer, owner of the MP3 patents commented openly, "We doubt very much that they are not using Fraunhofer and Thomson intellectual property," Linde said. "We think it is likely they are infringing."[107] Questionable infringement follows the speed to market issues. Whether it is enterprise software of popular consumer products, open source products actively migrate into the marketplace. This recent posting is another example:

"Freely copy iTunes Music Store files"

PlayFair is an open source app that strips Apple's DRM out of iTunes Music Store singles, allowing you to freely copy the music you pay for. It takes one of the iTMS Protected AAC Audio Files, decodes it using a key obtained from your iPod or Microsoft Windows system and then writes the new, decoded version to disk as a regular AAC Audio File. It then optionally copies the metadata tags that describe the song, including the cover art, to the new file. [108]

[106] Shankland, Stephen. Study: Open Source Databases Going Mainstream. 8 Mar. 2004. 19 Apr. 2004 <http://zdnet.com.com/2100-1104_2-5171543.html>.
[107] Borland, John. Open-source MP3 project continues after parent's demise. 11 Dec. 2000. 19 Apr. 2004. <http://news.com.com/2100-1023-249710.html?legacy=cnet>.
[108] Doctorow, Cory. Freely Copy iTunes Music Store Files. 4 Apr. 2004. 19 Apr. 2004 <http://boingboing.net/2004/04/04/freely_copy_itunes_m.html>.

Section VIII. The Collapse of Samizdat: 8 Scenarios

The 'source' of open source code is a topic with a number of ramifications. Like any other debate, some concerns will prove tenuous and others will have serious repercussions. The following are the eight most vulnerable aspects to the model, the developer community, and the user community.

It is certain that the model (as we know it) is not sustainable if any one of these vulnerabilities reaches a critical mass.

8.1 Pot Luck Software

No entity on earth can guarantee 'what' source code is in open source. There is no way to know, particularly, when we cannot identify who all of its contributors are. Credits files are arbitrary. Furthermore, we can never know whether code is stolen, accidently stolen, accidently distributed, etc. While the motivations seem pure, too many issues regarding the 'source' of open source code remain conspicuous.

8.2 Terminal Partnerships

The promotion of open source depends on corporate sponsorship. Paradoxically, every dollar of advertising and promotion corporations such as IBM and Oracle contribute to increasing customer interest in 'Free Platforms' respectively will cost these companies lucrative accounts.

This is a terminal relationship.

8.3 Un-Volunteerism

The pool of volunteer professional help faces three fates:

1) Whether outsourcing concerns will end programmer interest in 'free' work
2) Whether litigation concerns will end policies encouraging OSS participation
3) Whether competition concerns will end policies encouraging OSS participation

8.4 Diligence

To date, there has not been a risk/analysis study on open source. Inevitably however, this risk/analysis assessment will dismiss any notion that public domain software is 'free'. In addition, because the cost-effectiveness of implementing source code inspection/diligence programs have not been studied, it is uncertain whether public domain software can be cost-effectively 'cleaned' or 'sanitized'.

The point is—when the legal costs for indemnification, insurance and legal research begin to hike the price for the use of open source, it is unquestionable that total cost of ownership issues will slowly end the commercial viability of hybrid software.

8.5 Too Small To Sue Will Not Last

The best defense of the open source model has been the cost/benefit of suing small entities. This defense is specious: if the model is successful, it will face the same litigation challenges any other entity faces; if the model is unsuccessful, we must conclude it does not have long-term viability.

8.6 Money and Rights

We can expect that an unlucky entity will make a considerable profit with public domain software, only to find out that he must legally share those profits with another owner… But that is an entity. We may see an individual programmer, a very small entity raise this concern first.

Either way, small or large, a damaging IP violation will be difficult to walk away from...

8.7 Proprietary Becomes Fall-Back Position

To defend themselves, all open source organizations are slowly becoming more bureaucratic and more closed--more like proprietary software companies.

While customers are free to alter source, the only code that is guaranteed by vendors is the code that they supply. This is logical. Nevertheless, any significant legal event will slowly push all vendors to take this course. We can expect to see devolution of 'freeness' in place of closed environments to 1) preserve competitive advantage and 2) exercise legal discretion.

The gradual movement that we are already observing is moving away from the 'free' exchange is a telltale sign that inevitably, the widespread practice will inevitably look exactly like the proprietary model we have today.

8.8 Inevitability

Perhaps the most obvious problems is that we do not know whether the foundation of open source as we know it will survive a legal challenge.

No lawyer wins every case, thus, we can only expect that a lawyer will inevitably lose an open source case. At that point, any serious decision against the model will force it to significantly change from what it is today.

Section IX. Achieving Balance

9.1 Keeping A Free Software Model

Discussing the 'source' of open source code provides us with a litany of obvious conflicts between the science, law, business, and government arenas.
For the open source model to survive, it will be necessary for all four arenas to come to a more constructive relationship than there is today. One of the best ways to achieve this is with public policy that separates commercial software from academia.

Open source originated from the academic and science community as non-commercial research and should remain as such. To keep open source free of attribution, ownership and legal problems, we need to develop a policy that rigidly keeps software, particularly for use by colleges and universities, free for anyone to use. Meanwhile, commercial software models should exist to guarantee users and developers clean source code and liability-free software.

9.2 The Importance of R&D Integrity

After an abbreviated look at questionable cooption of intellectual property linked to the academic community specifically: The Lions Book and the University of New South Wales, Minix and Vrije University, Linux and the University of Helsinki, It becomes clear why it is inane for corporations to continue funding R&D programs at colleges when it could end up producing products that either compete with them, provide no financial remuneration, or that blindside their commercial interests.

In the interest of longevity, it is in the best interest of both parties that academia and its R&D sponsors come to agreement whether source code (or any intellectual property) for that matter will be used for academic purposes, or will go out into the private sector. Whether it is on a college campus or in the private sector, corporate interests cannot fund the cooption of their intellectual property. Attribution might just be a wink and a nod within academic circles, but among users, the courts, and ultimately the government, attribution easily becomes an issue of paramount importance.

It is interesting to note that Richard Stallman was on staff for MIT when he started the GNU commercial software project. Stallman decided to quit working for the university to eliminate conflicts of the ownership of GNU software.[109] Nevertheless, MIT's lenient facility and resource arrangement with Richard Stallman makes it a default financial sponsor of the GNU program. This is only to say that any corporate interest that is funding the research and development of

[109] Stallman, Richard. Richard Stallman's Personal Page. 1996. 15 Apr. 2004 <http://www.stallman.org/>.

better proprietary software could also be subsequently funding the free software project on the campus, a project that pledges to release source code out to the public.

9.3 Government Policy

Corporate interests cannot fund truly free software because their interests are tied to the promotion of their business. Corporations are not philanthropists. However, the government is in a better position. It could fund the creation of free software programs at universities that everyone could develop and own for academic or commercial interests. Professor Tanenbaum for example should have an opportunity to pursue sizable grants for operating systems development at Vrije University. A professor of his caliber would easily turn these R&D dollars into jobs and economic stimulus. However, restricted R&D for proprietary or hybrid source projects would keep the professor limited in his ability to work with students on projects such as Minix.

It is in the best interest for the federal government to take the lead on funding a bigger open source project at universities. The commercial open source model is 1) depreciating the value of U.S. proprietary software 2) depreciating the value of U.S. investment in the IT industry 3) diminishing the returns of the IT industry which is in turn send U.S. jobs overseas to make up for losses. 4) funding the devolution of the U.S. intellectual property rights economy.

The U.S. federal government is the biggest sponsor of research and development in the world. Colleges and universities receive billions of dollars in grants for science and technology research.[110] In addition, as of the beginning of 2002, the U.S. government owned over 27,700 patents.[111] Invention spawned from government sponsored research is the fruit of the entrusted investment of billions in taxpayer monies. This return on investment equals jobs, a vibrant economy and a nation that can continue to keep pace with technological development and innovation around the world.

Worse, the U.S. government sends billions of dollars in grants to universities. It is inane for the government to use taxpayer dollars to support the deterioration of its present and future investment in intellectual property. The federal government cannot allow the migration of taxpayer-funded technology into the public domain without a return on its investment.

Government concerns about open source software are not restricted to national security. The U.S. and the U.S. government have an exceptional stake in the future of a vibrant, sustainable intellectual property environment.

---

[110] Federal Research and Development Overview. Comp. R D. Burck. Nov. 14, 2001. University of Texas. May 5, 2004. <http://www.utsystem.edu/News/FederalRDoverview.PDF>.

[111] United States Trademark and Patent Office. TAF Profile Report: U.S. (Federal) Government Patenting. Apr. 2002. Apr. 2004. <http://www.uspto.gov/web/offices/ac/ido/oeip/taf/us_gov.pdf>.

Thus:

1) The government should support R&D at universities with open source projects that produce research that all parties can use. This includes developers and commercial interests. However, taxpayer dollars cannot support open source projects that are tied to commercial open source models that compete with the private sector.

2) Universities and colleges that receive government grants should not be able use taxpayer dollars to generate source code that is restrictive. Both individuals and business should be able subsequently to develop free software and protect it as its own intellectual property

To be clear, the hybrid open source model encourages conspicuous development and 'sources' of source code, because it competes with commercial proprietary software models. In addition, the present hybrid open source model depends upon commercial interests to 'volunteer' manpower, resources, etc. to perpetuate a model that may never produce any returns or cannibalize its existing or new customers. This is short term drag on the U.S. economy and a long-term drag on U.S. innovation. For a U.S. corporation to contribute to free software models to any extent, diminishes their overall inventory of intellectual property. Especially in the IT world where billions of dollars in value in U.S. corporations is tied directly to its intellectual property inventory, companies need to increase there ownership of prime IP, not decrease it.

Finally, U.S. corporations, especially in today's economy, would only benefit by more research and development assistance. The hybrid commercial open source model will inevitably become one of the biggest corporate investment mistakes in history. A government policy that promotes 'free and clear' open source development at universities and colleges would discourage U.S. corporations from continuing to fund a failing model. Everyday a company uses the hybrid open source model, they are increasing the exposure to financial liability.

9.4 Agreement Across All Arenas

There are a number of advantages to this model.

1.) Improve Current Relationships

Government funded open source would increase the opportunities for universities and corporations to capitalize on intellectual property. With more research and development available, corporations would have a greater interest in partnering with universities. In addition, we would end problems with academic development of software. It would be truly free for all to use. Professors and students would not need to worry about license, copyright, etc. and other issues.

They would be “free” to invent, which is the true intention of research and development programs, especially within academic environments.

2.) Support Purist Approach to Innovation

Government supported open source would end debates regarding attribution, ownership and copyright. In addition, people could contribute their time for academic purposes or commercial purposes without complicated contracts. Companies and individuals would be free to develop new products with a need to co-opt copyright restrictions.

3.) Increase competitiveness of U.S. IT Industry

Corporations spend billions of dollars on research and development. While this cost is necessary, the more support it could from the academic community the better. While this research would be available to their competitors, it would still be fruitful for the IT industry to have a more robust universe for testing and improving upon new ideas.

4.) Legal Community and Licensing

As we have explored, without certainty, the commercial hybrid open source model will become a financial and litigious black hole. Lawyers could better protect their client’s interests with a clearer delineation between free and proprietary software.

Section X. Policy Recommendations

It is in the best interest of all parties if the origin and history of invention is properly recorded. If we fail to attribute invention properly, we do the additional injustice of not appreciating its incredible contribution to society. We devalue invention when we decide not to protect it, or worse attribute its beginnings to non-inventors. The Unix inventors, Dennis Ritchie and Kenneth Thompson are hailed as geniuses for developing probably the most influential software in the history of science and technology. But if we are not careful, it becomes easy to devalue their contribution. Subsequently, if we are not careful, we devalue all invention.

Appreciating the true origin of invention is not just a matter of intellectual property protection, it provides us with understanding to know what resources, talent, and environment is needed for our society to continue to advance technologically.

Thus, understanding invention is in the best interest of its inventors, invention and our community. To support this concern, as policy, the federal government should:

1. Work vigorously to create a true 'free source' code capability program at universities and colleges. This program should go to promote true open source projects, not hybrid source projects like the GPL and Linux. The federal government should support a $5 billion budget over ten years to produce a true 'free source code' program in partnership with the IT industry, and other governments interested in promoting increased computers science research and development. This effort would be a benefit to academia, the private sector, and the IT economy.

2. Actively study the taxpayer return on investment (TROI) from government funded government research and development at colleges and universities.

3. Increase the United States Patent and Trademark Office budget to property support the anticipated growth in intellectual property filings by the public as result of the 'open source' program at colleges and universities.

4. Increase financial incentives for corporations to participate in an open source program at colleges and universities.

Section XI. Bibliography

1. Active File Comparison. Created 2001. Accessed Apr. 27, 2004. <http://www.formulasoft.com/afc.html>.

2. Amoeba distributed operating system - Wikipedia. Jan. 25, 2004. Apr. 15, 2004 <http://en.wikepedia.org/wiki/Amoeba>.

3. Amoeba WWW Home Page. Comp. Andrew Tanenbaum. Apr. 1998. Vrije University. Apr. 15, 2004 <http://www.cs.vu.nl/pub/amoeba/>.

4. Andrew S. Tanenbaum - Wikipedia. Feb. 25, 2002. Apr. 15, 2004 <http://en.wikipedia.org/wiki/Andrew_Tanenbaum>.

5. Bank, David. 'Open Source' Database Poses Threat to Oracle. July 9, 2003. Apr. 19, 2004. <http://webreprints.djreprints.com/785490482991.html>.

6. Barman, Scott A. "Re: SCO Forum98 Announcements (fwd)." Online Posting. Aug. 24, 1998. Mid-Atlantic Linux. Apr. 15, 2004 <http://boudicca.tux.org/mhonarc/ma-linux/>.

7. Beyond Compare Feature List. Created 2004. Accessed Apr. 27, 2004. <http://www.scootersoftware.com/moreinfo.php?c=features>.

8. Bezroukov, Nikolai. Portraits of Open Source Pioneers. Mar. 27, 2004 <http://www.softpanorama.org/People/index.shtml>.

9. Binstock, Andrew. Obfuscation: Cloaking Your Code From Prying Eyes. Mar. 6, 2003. Apr. 19, 2004 <http://www.devx.com/SummitDays/Article/11351>.

10. Birth of Linux. Apr. 2, 1995. Apr. 16, 2004 <http://www.externet.hu/linux/linux-birth.html>.

11. Borland, John. Open-source MP3 project continues after parent's demise. Dec. 11, 2000. Apr. 19, 2004. <http://news.com.com/2100-1023-249710.html?legacy=cnet>.

12. Bot, Kees & Tanenbaum, Andrew. Minix Copyright Notice. 25 Apr. 2004 <http://www.pdos.lcs.mit.edu/ld/MINIX-README.html>.

13. Chance, Frank. Book Reviews: The Mythical Man-Month by Frederick Brooks Jr.. 15 Apr. 2004 <http://www.fabtime.com/man-month.shtml>.

14. Compare and Merge. Created 2002. Accessed Apr. 27, 2004. <http://www.compareandmerge.com/>.

15. Compiler - Wikipedia. Oct. 26, 2001. Apr. 21, 2004. <http://en.wikipedia.org/wiki/Compiler>.

16. CPPX. Created June 14, 2001. Accessed Apr. 27, 2004 <http://swag.uwaterloo.ca/~cppx/>.

17. Dennis Ritchie Home Page. Mar. 2002. Apr. 27, 2004. <http://www.cs.bell-labs.com/who/dmr/>.

18. Doctorow, Cory. Freely Copy iTunes Music Store Files. Apr. 4, 2004. Apr. 19, 2004 <http://boingboing.net/2004/04/04/freely_copy_itunes_m.html>.

19. Duncan, Ray. ERCB: The Mythical Man-Month. May 26, 1996. Electric Review of Computer Books. Apr. 15, 2004. <http://www.ercb.com/feature/feature.0001.html>.

20. Federal Research and Development Overview. Comp. R D. Burck. Nov. 14, 2001. University of Texas. May 5, 2004.<http://www.utsystem.edu/News/FederalRDoverview.PDF>.

21. Green, Roedy. Obfuscator - Java Glossary. 1996. Apr. 19, 2004 <http://www.mindprod.com/jgloss/obfuscator.html>.

22. Grind. Accessed Apr. 27, 2004 <http://www.cs.bgu.ac.il/~gmayer/grind/>.

23. Gross, Grant. Contract Case Could Hurt Reverse Engineering. June 26, 2003. Apr. 19, 2004. <http://www.infoworld.com/article/03/06/26/HNreverseengineering_1.html>.

24. Hansen, Evan. US Court: Reverse Engineering is 'Presumptively Legal'. Mar. 1, 2004. Apr. 19, 2004 <http://news.zdnet.co.uk/business/legal/0,39020651,39147906,00.htm>.

25. Hauben, Michael, and Ronda Hauben. Netizens: On the History and Impact of Usenet and the Internet. Los Alamitos, CA: Wiley-IEEE Computer Society P, 1997.

26. Hauben, Ronda. Home Page. Aug. 1, 1994. Apr. 15, 2004. <http://www.linuxjournal.com/article.php?sid=2792>.

27. Hauben, Ronda. History of UNIX: On the Evolution of Unix and the Automation of Telephone Support Operations (ie. of Computer Automation). Apr. 15, 2004. <http://www.dei.isep.ipp.pt/docs/unix-Part_I.html>.

28. Jalopy. Created Dec. 31, 2003. Accessed Apr. 27, 2004. <http://jalopy.sourceforge.net/>.

29. JCloak. Accessed Apr. 27, 2004. <http://www.sys-con.com/java/wbg/CurrentSearch_Detail.cfm?ID=529>.

30. Jindent. Accessed Apr. 27, 2004 <http://www.jindent.com/>.

31. John Lions Chair in Operating Systems. 2001. University of New South Wales. Apr. 15, 2004. <www.do.cse.unsw.edu.au/industry/JohnLions/chair.phtml>.

32. Jones, Henry. Interview with AdTI. Apr. 14, 2004.

33. JRefactory. Accessed Apr. 27, 2004. <http://jrefactory.sourceforge.net/csrefactory.html>.

34. Ken Thompson - Wikipedia. July 27, 2001. Apr. 28, 2004. <http://en.wikipedia.org/wiki/Ken_Thompson>.

35. Kutvonen, Petri. Linux. University of Helsinki. Apr. 19, 2004 <http://www.cs.helsinki.fi/linux/>.

36. Levenez, Eric. Unix History. Apr. 15, 2004 <http://www.levenez.com/unix/>.

37. Levenez, Eric. Interview with AdTI. Apr. 26, 2004.

38. Linux 2.5.59 Credits File. Apr. 15, 2004 <http://www.iglu.org.il/lxr/source/CREDITS?v=linux-2.5.59>.

39. Loli-Queru, Eugenia. Interview with Linus Torvalds. Oct. 10, 2001. Apr. 15, 2004 <http://www.osnews.com/story.php?news_id=161>.

40. Merrill, Mary V. Being a Gatekeeper. Apr. 2004. Apr. 19, 2004 <http://www.merrillassociates.net/topicofthemonth.php>.

41. Miller, Ernest. Good News - MS Windows Source Code Leaked. Feb. 13, 2004. Apr. 16, 2004. <http://www.corante.com/importance/archives/001877.html>.

42. Mills, Charles. Interview with AdTI. Apr. 14, 2004.

43. Mills, Charles. Interview with AdTI. Apr. 15, 2004.

44. Minix Copyright. 25 Apr. 2004. <http://www.cs.vu.nl/pub/minix/LICENSE>.

45. Minix Timeline (1961-2004). Aug. 30, 2002. Apr. 15, 2004 <http://minix-up.sourceforge.jp/timeline/TIMELINE.html>.

46. Minix - Wikipedia. Sept. 18, 2001. Apr. 15, 2004 <http://en.wikipedia.org/wiki/Minix>.

47. Moffitt, Nick. Nick Moffitt's $7 History of Unix. 2000. Apr. 15, 2004 <http://www.crackmonkey.org/unix.html>.

48. Oskov, Nasko, and Mike Perry. Introduction to Reverse Engineering Software. Apr. 19, 2004 <http://www.acm.uiuc.edu/sigmil/RevEng/>.

49. Perens, Bruce. The Open Source Definition. June 1997. Apr. 19, 2004 <http://www.opensource.org/docs/definition.php>.

50. Products Center. Accessed Apr. 27, 2004. <http://www.headwaysoft.com/html/prodcntr.htm>.

51. Raymond, Eric S. The Cathedral and the Bazaar. Apr. 10, 2004 <http://www.free-soft.org/literature/papers/esr/cathedral-bazaar/cathedral-ba zaar-10.html>.

52. Raymond, Eric S. New Hacker's Dictionary. 3rd ed. Cambridge, MA: MIT P, 1996.

53. Ritchie, Dennis. The Development of the C Language. Second History of Programming Languages Conference. Cambridge, MA. Apr. 1993.

54. Ritchie, Dennis. Interview with AdTI. Apr. 4, 2004.

55. Rose, Will. "Re: Minix VS Linux." Online posting. Feb. 3, 1992. comp.os.minix. Apr. 25, 2004. <http://www.ibiblio.org/usenet-i/groups-html/comp.os.minix.html>.

56. Shankland, Stephen. Study: Open Source Databases Going Mainstream. Mar. 8, 2004. Apr. 19, 2004 <http://zdnet.com.com/2100-1104_2-5171543.html>.

57. Stallman, Richard. The GNU Project. 1998. Apr. 8, 2004. <http://www.gnu.org/gnu/thegnuproject.html>.

58. Stallman, Richard. Richard Stallman's Personal Page. 1996. Apr. 15, 2004. <http://www.stallman.org/>.

59. Stein, Harvey J. "Benevolent Dictatorship." Online Posting. Mar. 20, 1998. LispOS. Apr. 16, 2004 <http://lists.tunes.org/mailman/listinfo/lispos>.

60. Strasser, Mathias. "A New Paradigm in Intellectual Property Law?" Stanford Technology Law Review (2001): 1-73. Apr. 19, 2004. <http://stlr.stanford.edu/STLR/Articles/01_STLR_4/article.htm>.

61. Stunnix Perl-Obfus. Created 2002. Accessed Apr. 27, 2004. <http://www.stunnix.com/prod/po/overview.shtml>.

62. Tanenbaum, Andrew. Andrew S. Tanenbaum's FAQ. Apr. 8, 2004. <http://www.cs.vu.nl/~ast/home/faq.html>.

63. Tanenbaum, Andrew. Minix Information Sheet. Vrije University. Apr. 19, 2004. <http://www.cs.vu.nl/~ast/minix.html>.

64. Tanenbaum, Andrew. Minix. 2000. Vrije University. Apr. 15, 2004. <http://www.cs.vu.nl/pub/minix/>.

65. Tanenbaum, Andrew. Online interview. Apr. 12, 2004.

66. Tanenbaum, Andrew. Personal interview. Mar. 23, 2004.

67. Tanenbaum, Andrew. Read Me. Vrije University. Apr. 25, 2004 <http://www.cs.vu.nl/pub/minix/old/demo/READ_ME>.

68. The Creation of the Unix Operating System. 2002. Bell Labs. Apr. 16, 2004. <http://www.bell-labs.com/history/unix>.

69. The Cxref Homepage. Created Mar. 2, 2004. Accessed Apr. 27, 2004. <http://www.gedanken.demon.co.uk/cxref/>.

70. The Holy Bible: Revised Standard Version. New York: Thomas Nelson & Sons, 1953. 10-11.

71. The John Lions Award For Research Work in Open Systems. 2002. Australian UNIX and Open Systems Users Group. Apr. 15, 2004 <www.auug.org.au/awards/lions>.

72. Torvalds, Linus B. "What would you like to see most in minix?" Online posting. Aug. 25, 1991. comp.os.minix. Apr. 10, 2004. <http://www.ibiblio.org/usenet-i/groups-html/comp.os.minix.html>.

73. Travis, Greg. How to lock down your Java code (or open up someone else's). May 1, 2001. Apr. 19, 2004. <http://www-106.ibm.com/developerworks/java/library/j-obfus/>.

74. Tuomi, Ilkka. Networks of Innovation: Change and Meaning in the Age of the Internet. NYC, NY: Oxford P, 2003.

75. Tuomi, Ilkka. Evolution of the Linux Credits File. Apr. 1, 2004 <http://www.jrc.es/~tuomiil/articles/EvolutionOfTheLinuxCreditsFile.pdf>.

76. Tyma, Paul. Encryption, Hashing and Obfuscation. Apr. 8, 2003. Apr. 17, 2004.

77. Unger, Thomas. The Amiga Minix Page. 2003. Apr. 16, 2004 <http://home.arcor.de/kickstart/TKA/Tutorials/AmigaMINIX/aminix.html>.

78. "UNIX Implementation." The Bell System Technical Journal 57. (1978)

79. United States Trademark and Patent Office. TAF Profile Report: U.S. (Federal) Government Patenting. Apr. 2002. Apr. 2004. <http://www.uspto.gov/web/offices/ac/ido/oeip/taf/us_gov.pdf>.

80. VideoLAN Project, the. Interview with Sean Michael Kerner. Builder.com. Jan. 15, 2004. Apr. 14, 2004 <http://builder.com.com/5100-6375-5136135.html?tag=search

81. Welcome to the Department of Computer Science. Vrije University. Apr. 15, 2004 <http://www.cs.vu.nl/welkom-en.html>.

Section XII. Notes

Note I:

Andrew Tanenbaum Papers:

M. van Steen, F.J. Hauck and A.S. Tanenbaum. "A Model for Worldwide Tracking of Distributed Objects." Proc. TINA '96 Conference, Heidelberg, Germany, September 1996, pp. 203-212.

P. Homburg, M. van Steen, and A.S. Tanenbaum. "An Architecture for a Wide Area Distributed System." Proc. Seventh ACM SIGOPS European Workshop, Connemara, Ireland, September 1996, pp. 75-82.

P. Homburg, M. van Steen, and A.S. Tanenbaum. "Communication in GLOBE: An Object-Based Worldwide Operating System." Proc. Fifth International Workshop on Object Orientation in Operating Systems, Seattle, Washington, October 1996, pp. 43-47.

M. van Steen, P. Homburg, and A.S. Tanenbaum. "The Architectural Design of Globe: A Wide-Area Distributed System." Technical Report IR-422, March 1997.

M. van Steen, F.J. Hauck, P. Homburg, and A.S. Tanenbaum. "Locating Objects in Wide-Area Systems." IEEE Communications Magazine, January 1998, pp. 104-109.

M. van Steen, A.S. Tanenbaum, I. Kuz, and H.J. Sips. "A Scalable Middleware Solution for Advanced Wide-Area Web Services." Proc. Middleware '98, The Lake District, UK, Sept. 1998.

A. Bakker, M. van Steen, and A.S. Tanenbaum "Replicated Invocation in Wide-Area Systems." Proc. Eighth ACM SIGOPS European Workshop, Sintra, Portugal, Sept. 1998.

M. van Steen, F.J. Hauck, G. Ballintijn, A.S. Tanenbaum. "Algorithmic Design of the Globe Wide-Area Location Service." The Computer Journal 41(5):297-310, 1998.
M. van Steen, P. Homburg, and A.S. Tanenbaum. "Globe: A Wide-Area Distributed System." IEEE Concurrency, January-March, 1999, pp. 70-78.

G. Ballintijn, M. van Steen, A.S. Tanenbaum. "Exploiting Location Awareness for Scalable Location-Independent Object IDs." Proc. Fifth Annual ASCI Conf., Heijen, The Netherlands, June 1999, pp. 321-328.

Ballintijn, M. van Steen, A.S. Tanenbaum. "Simple Crash Recovery in a Wide-area Location Service." Proc. 12th Int'l. Conf. on Parallel and Distributed Computing Systems, Fort Lauderdale, Florida, August 1999, pp. 87-93.

I. Kuz, P. Verkaik, I. van der Wijk, M. van Steen, A.S. Tanenbaum. "Beyond HTTP: An Implementation of the Web in Globe." Technical Report IR-465, November 1999.

G. Ballintijn, M. van Steen, A.S. Tanenbaum. "Exploiting Location Awareness for Scalable Location-Independent Object IDs." Technical Report IR-459, January 1999.

G. Ballintijn, P. Verkaik, E. Amade, M. van Steen, A.S. Tanenbaum. "A Scalable Implementation for Human-Friendly URIs." Technical Report IR-466, November 1999.

A. Bakker, M. van Steen, and A.S. Tanenbaum. "From Remote Objects to Physically Distributed Objects." Proc. 7th IEEE Workshop on Future Trends of Distributed Computing Systems, Cape Town, South Africa, December 1999, pp. 47-52.

J. Leiwo, C. Haenle, P. Homburg, C. Gamage, A.S. Tanenbaum. "A Security design for a wide-area distributed system." Proc. Second International Conference Information Security and Cryptology (ICISC'99), Seoul, South Korea, December 1999. In LNCS 1787, pp. 236-256.

M. Jansen, E. Klaver, P. Verkaik, M. van Steen, A.S. Tanenbaum. "Encapsulating Distribution in Remote Objects." Technical Report IR-476, August 2000.

G. Ballintijn, M. van Steen, A.S. Tanenbaum. "Characterizing Internet Performance to Support Wide-area Application Development." Operating Systems Review, 34(4):41-47, October 2000.

A. Baggio, G. Ballintijn, M. van Steen, A.S. Tanenbaum. "Efficient Tracking of Mobile Objects in Globe." Technical Report IR-481, November 2000.

G. Pierre, I. Kuz, M. van Steen, A.S. Tanenbaum. "Differentiated Strategies for Replicating Web Documents". Computer Communications 24(2):232-240, February 2001.

A. Bakker, M. van Steen, A.S. Tanenbaum. "A Distribution Network for Free Software." Technical Report IR-485, February 2001.

M. Jansen, E. Klaver, P. Verkaik, M. van Steen, A.S. Tanenbaum. "Encapsulating Distribution in Remote Objects." Information and Software Technology 43(6):353-363, May 2001.

G. Ballintijn, M. van Steen, A.S. Tanenbaum. "Scalable User-Friendly Resource Names". IEEE Internet Computing, vol. 5(5), 2001, pp. 20-27.

A. Bakker, M. van Steen, A.S. Tanenbaum. "A Law-Abiding Peer-to-Peer Network for Free-Software Distribution." Proc. IEEE Int'l Symp. on Network Computing and Applications, Cambridge, MA, February 2002.

B. Popescu, M. van Steen, A.S. Tanenbaum. "A Security Architecture for Object-Based Distributed Systems." Technical Report IR-492, February 2002.

G. Pierre, M. van Steen, A.S. Tanenbaum. "Dynamically Selecting Optimal Distribution Strategies for Web Documents." IEEE Transactions on Computers, vol. 51 (6), June 2002.

B.C. Popescu, C. Gamage, A.S. Tanenbaum. "Access Control, Reverse Access Control and Replication Control in a World Wide Distributed System." Proc. 6th IFIP Communications and Multimedia Security Conference, Portoroz, Slovenia, October 2002.

B.C. Popescu, M. van Steen, A.S. Tanenbaum. "A Security Architecture for Object-Based Distributed Systems." Proc. 18th Annual Computer Security Applications Conference, Las Vegas, NV, December 2002.

A. Bakker, M. van Steen, A.S. Tanenbaum "A Wide-Area Distribution Network for Free Software." Technical Report IR-CS-002, January 2003.

Bogdan C. Popescu, Bruno Crispo, Andrew S. Tanenbaum, Maas Zeeman "Expressing Security Policies for Distributed Objects Applications" Proc. 11th Cambridge International Workshop on Security Protocols, to appear (LNCS series), April 2003.

A. Bakker, I. Kuz, M. van Steen, A.S. Tanenbaum, P. Verkaik. "Design and Implementation of the Globe Middleware." Technical Report IR-CS-003, June 2003.

B. Popescu, B. Crispo, A.S. Tanenbaum. "Symmetric Key Authentication Services Revisited." Technical Report IR-CS-005, September 2003.

B.C. Popescu, B. Crispo, A.S. Tanenbaum. "Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System" Proc. 12th Cambridge International Workshop on Security Protocols, April 2004.

Note II:

Additional Samples of the Linux Credits File:

Version 2.5.59 (released January 2003)

This is at least a partial credits-file of people that have
contributed to the linux project. It is sorted by name, and
formatted in a format that allows for easy grepping and
beautification by scripts. The fields are: name (N), email (E),
web-address (W), PGP key ID and fingerprint (P), description (D)
and snail-mail address (S).
Thanks,

Linus

N: Matti Aarnio
E: mea@utu.fi
D: LILO for AHA1542, modularized several of drivers/net/,
D: dynamic SLIP devices, dynamic /proc/net/, true size /proc/ksyms,
D: and other hacks..
D: Documenting various parts of network subsystem (kernel side)

N: Werner Almesberger
E: werner.almesberger@lrc.di.epfl.ch
D: dosfs, LILO, some fd features, various other hacks here and there
S: Ecole Polytechnique Federale de Lausanne
S: DI-LRC
S: INR (Ecublens)
S: CH-1015 Lausanne
S: Switzerland

N: H. Peter Anvin
E: hpa@zytor.com
W: http://www.zytor.com/~hpa/
P: 2047/2A960705 BA 03 D3 2C 14 A8 A8 BD 1E DF FE 69 EE 35 BD 74
D: Author of the SYSLINUX boot loader, maintainer of the linux.* news
D: hierarchy and the Linux Device List; various kernel hacks
S: 4390 Albany Dr. #46
S: San Jose, California 95129
S: USA

Note III:

Tuomi Chart: lu

	1994	1994	1994	1995	1995	1995	1996	1996	1996	1997	1998	1999	1999	1999	2000	2000	2000	2001	2001	2001	2002
country\ version	100	110	1123	120	130	136	210	200	201	2144	21108	220	230	239	2214	2351	240	246	249	2417	2525
Argentina	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Australia	3	3	3	7	7	8	11	11	11	11	16	17	17	17	17	20	21	21	20	20	20
Austria	0	0	0	0	0	0	1	1	1	1	1	3	3	3	3	4	4	4	4	4	4
Belgium	1	1	1	1	1	1	2	2	2	2	3	4	4	4	4	4	4	4	4	4	4
Brazil	0	0	0	0	0	0	0	0	0	0	0	1	1	2	2	4	6	7	10	10	10
Bulgaria	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
Canada	2	2	2	8	8	8	12	11	11	12	12	13	13	14	14	17	17	19	19	20	23
Croatia	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
Czech Republic	0	0	0	0	0	0	1	1	1	2	6	7	7	8	8	9	10	10	10	10	10
Denmark	1	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	4	4	4	4	4
Finland	5	5	6	6	6	6	7	7	7	6	6	6	6	6	6	8	9	9	9	9	9
France	1	1	1	2	3	3	7	6	6	6	8	8	9	10	9	11	11	11	12	12	12
Germany	15	16	17	25	25	25	38	38	38	41	45	49	51	51	51	58	60	65	66	67	69
Hong Kong	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Hungary	0	0	0	0	0	0	0	0	0	1	1	1	2	2	2	2	2	2	2	2	3
Ireland	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2
Italy	0	0	0	2	2	2	2	2	2	2	4	4	4	4	4	5	5	5	5	5	5
Japan	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Luxembourg	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
Mexico	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Netherlands	8	8	8	9	9	9	14	15	15	15	16	16	16	17	17	18	18	19	18	18	18
New Zealand	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
Norway	0	0	0	0	0	0	2	2	2	2	1	1	2	2	2	4	4	4	4	4	4
Poland	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	2	2	2	2	2
Portugal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
Romania	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	2	2
Russia	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
South Africa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
Spain	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
Sweden	1	1	1	1	1	1	1	1	1	2	2	3	3	4	4	6	8	8	8	9	9

Switzerland	0	0	0	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Taiwan	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
UK	5	5	5	8	8	8	12	11	11	14	18	21	21	21	22	25	26	30	32	32	32
Ukraine	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
USA	36	36	39	54	54	54	74	71	71	78	92	96	97	99	100	121	139	146	148	149	154
Unknown	1	1	1	1	1	1	1	1	1	2	2	2	2	4	2	4	5	5	6	6	6
Total	80	81	86	128	129	130	196	190	190	209	245	269	275	287	284	341	375	395	403	408	418
db lookups	13	14	13	23	24	24	27	27	27	40	34	41	46	42	58	48	67	73	71	72	77

Note V:
Levenez Unix Timeline: