

# AFTER THE SOFTWARE WARS



Draft: 1.025, 07/28/2009  
Words: 94669, Pages: 298

KEITH CURTIS

Copyright © 2009 by Keith Curtis

I am making this book available as a free digital download. However, I would prefer that you not give copies to other people, and instead tell them to download it from [lulu.com](http://lulu.com). This allows me to ensure all readers get the latest version, and to keep track of the number of copies out there. Your copy is yours, of course.

If you enjoyed a free version of this book, and you want to send me a donation as thanks for my two years of labor, that would be appreciated! You could purchase a paper copy, or go to [keithcu.com](http://keithcu.com) and click on the PayPal button. Any amount donated over \$5 will be given to worthy efforts in free software. If you'd like to contribute money towards a particular area of free software, but don't know how, I can help!

Several people advised me to use a Creative Commons license, but I did not see anything that looked like this.

In general, I tried to get permission for the use of other's information. However, I have over 100 images and it was hard to figure out where to get permission for some of them. For those, I will claim permission under fair use ;-) I am happy to remove any content if any owner objects.

Keith Curtis  
[keithcu@gmail.com](mailto:keithcu@gmail.com)  
twitter: @keithccurtis

ISBN 978-0-578-01189-9

# CONTENTS

|   |     |
|---|-----|
| Free Software Battle.....                         | 1   |
| Free Software Army.....                           | 3   |
| iBio.....   | 5   |
| Glossary.....                                     | 9   |
| Wikipedia.....                                    | 10  |
| Linux.....  | 16  |
| Distributed Development.....                      | 20  |
| Linux Kernel Superiority.....                     | 23  |
| The Feature Race.....                             | 34  |
| Linux is Inexorably Winning.....                  | 37  |
| Charging for an OS.....                           | 38  |
| Free Software Only Costs PCs.....                 | 41  |
| A Free Operating System.....                      | 42  |
| Linux Distributions.....                          | 48  |
| AI and Google.....                                | 52  |
| Deep Blue has been Deep-Sixed.....                | 52  |
| DARPA Grand Challenge.....                        | 53  |
| Software and the Singularity.....                 | 58  |
| Google.....                                       | 60  |
| Conclusion.....                                   | 69  |
| Free Software.....                                | 70  |
| Software as a Science.....                        | 72  |
| Definition of Free Software.....                  | 74  |
| Copyleft and Capitalism.....                      | 75  |
| Is Copyleft a Requirement for Free Software?..... | 76  |
| Why write free software?.....                     | 78  |
| Should all Ideas be Free?.....                    | 89  |
| Pride of Ownership.....                           | 90  |
| Where Does Vision Fit In?.....                    | 91  |
| Governments and Free Software.....                | 92  |
| Should all Software be GPL?.....                  | 94  |
| Microsoft's Responses to Free Software.....       | 95  |
| Just a Stab.....                                  | 97  |
| Patents & Copyright.....                          | 99  |
| Software is math.....                             | 103 |
| Software is big.....                              | 105 |
| Software is a fast-moving industry.....           | 106 |
| Copyright provides sufficient protection.....     | 106 |

|                                      |     |
|--------------------------------------|-----|
| Conclusion.....                      | 107 |
| Biotechnology Patents .....          | 108 |
| Openness in Health Care.....         | 111 |
| The Scope of Copyright.....          | 113 |
| Length of Copyright.....             | 113 |
| Fair Use.....                        | 115 |
| Digital Rights Management (DRM)..... | 116 |
| Music versus Drivers.....            | 120 |
| The OS Battle.....                   | 122 |
| IBM.....                             | 123 |
| Red Hat.....                         | 125 |
| Novell.....                          | 127 |
| Debian.....                          | 128 |
| Ubuntu.....                          | 132 |
| One Linux Distro?.....               | 140 |
| Apple.....                           | 142 |
| Windows Vista.....                   | 154 |
| Tools.....                           | 158 |
| Brief History of Programming.....    | 159 |
| Lisp and Garbage Collection.....     | 164 |
| Reliability.....                     | 167 |
| Portability.....                     | 175 |
| Efficiency.....                      | 178 |
| Maintainability.....                 | 182 |
| Functionality and Usability.....     | 184 |
| Conclusion.....                      | 185 |
| The Java Mess.....                   | 187 |
| Sun locked up the code.....          | 189 |
| Sun obsessed over specs.....         | 191 |
| Sun locked up the design.....        | 193 |
| Sun fragmented Java.....             | 194 |
| Sun sued Microsoft.....              | 195 |
| Java as GPL from Day 0.....          | 195 |
| Pouring Java down the drain.....     | 197 |
| Let's Start Today.....               | 199 |
| Challenges for Free Software.....    | 203 |
| More Free Software.....              | 204 |
| Cash Donations.....                  | 205 |
| Devices.....                         | 207 |
| Reverse Engineering.....             | 209 |
| PC Hardware.....                     | 210 |
| Fix the F'ing Hardware Bugs!.....    | 212 |



|  |     |
|--|-----|
| Metrics.....                               | 213 |
| Volunteers Leading Volunteers.....         | 214 |
| Must PC vendors ship Linux?.....           | 215 |
| The Desktop.....                           | 216 |
| Approachability.....                       | 217 |
| Monoculture.....                           | 219 |
| Linux Dev Tools.....                       | 222 |
| Backward Compatibility.....                | 223 |
| Standards & Web.....                       | 225 |
| Digital Images.....                        | 226 |
| Digital Audio.....                         | 226 |
| The Next-Gen DVD Mess.....                 | 227 |
| MS's Support of Standards.....             | 229 |
| OpenDocument Format (ODF).....             | 231 |
| Web.....                                   | 237 |
| Da Future.....                             | 243 |
| Phase II of Bill Gates' Career.....        | 243 |
| Space, or How Man Got His Groove Back..... | 246 |
| The Space Elevator.....                    | 251 |
| 21st Century Renaissance.....              | 263 |
| Warning Signs From the Future.....         | 265 |
| Afterword.....                             | 267 |
| US v. Microsoft.....                       | 267 |
| Microsoft as a GPL Software Company.....   | 269 |
| The Outside World.....                     | 272 |
| How to try Linux.....                      | 289 |
| Dedication.....                            | 290 |
| Acknowledgments.....                       | 290 |

# FREE SOFTWARE BATTLE

Some people think much faster computers are required for Artificial Intelligence, as well as new ideas. My own opinion is that the computers of 30 years ago were fast enough if only we knew how to program them.

—John McCarthy, computer scientist, 2004



*This IBM 305 RAMAC Computer, introduced in 1956, was the first computer containing a (5 MB) hard drive on 24 huge spinning platters. Today you can get 1000 times more memory in something the size of your thumb.*

Given the technology that's already available, we should have cars that drive us around, in absolute safety, while we lounge in the back and sip champagne. All we need is a video camera on the roof, plugged into a PC, right? We have all the necessary *hardware*, and have had it for years, but don't yet have robot-driven cars because we don't have the *software*. This book explains how we can build better software and all get our own high-tech chauffeur.

The key to faster technological progress is the more widespread use of free software. Free versus proprietary (or non-free) software is similar to the divide between science and alchemy. Before science, there was alchemy, where people guarded their ideas because they wanted to corner the market on the means to convert lead into gold. The downside of this “strategy” is that everyone would have to learn for themselves that drinking mercury is a [bad idea](#).<sup>1</sup> The end of the Dark Ages arrived when man started to share advancements in math and science for others to use and improve upon. In fact, one way to look at history is to divide it between periods of progress and stagnation.

Computers are an advancement comparable to the invention of movable type. While computers and the Internet have already changed many aspects of our lives, we still live in the dark ages of computing because proprietary software is still the dominant model. One might say that the richest alchemist who ever lived is my former boss, Bill Gates. (Oracle founder Larry Ellison, and Google co-founders Sergey Brin and Larry Page are close behind.)

This book will discuss free software, but the question of whether scientific research and resources of information such as libraries should be free was answered long ago. In fact, the first (privately funded) library in America was created by Ben Franklin in 1731, 45 years before the nation itself was founded. The library's motto was the Latin phrase: “To support the common good is divine.” Ben Franklin understood that sharing knowledge has no downside.

Human knowledge increasingly exists in digital form, so building new and better models requires the improvement of software. People can only share ideas when they also share the software to display and modify them. It is the expanded use of free software that will allow a greater ability for people to work together and increase the pace of progress. The case studies examined in this book demonstrate that a system where anyone can edit, share, and review the

---

<sup>1</sup> The digital version of this book has a number of hyperlinked words that take you to references, like this video of writer Cory Doctorow at a Red Hat Summit.

body of work will lead not just to something that works, but eventually to the best that the world can achieve! Better cooperation among our scientists will lead to, robot-driven cars, pervasive robotics, artificial intelligence, and much faster progress in biology, all of which rely heavily on software.

A later chapter will describe the software freedoms in more detail, and the motivations for programmers to use and write free software, but it is important to clarify here that free software generally means that the source code is made available to its users. Microsoft's Internet Explorer is not free because it requires a Windows license, but more importantly, you cannot download the source code to learn how it works.

Today, proprietary software is considered more valuable than free software because its owners charge for a black box, but that thinking is exactly backwards. Proprietary software is less valuable because you cannot learn how it works, let alone improve it. It cannot make *you* better, and you cannot make *it* better. It is true that not everyone will exercise the right to read and change their software, just as not everyone exercises their right to their freedom of the press, but that doesn't make the freedom any less valuable!

## Free Software Army

Glenn Reynolds, in his book *Army of Davids*, talks about how armies, like bloggers in pajamas, are changing journalism and other aspects of our lives. This book will focus on the free software army, created by Richard Stallman in 1985. The rank and file of this army consists of loosely-knit programmers, who live in many countries, speak different mother tongues, and either work for competing companies, or volunteer their time, to place their fingerprint on the world's software knowledgebase.

Sourceforge.net, the largest free software repository, has 1,900,000 registered developers today. Even if we divide that number by 50, because many work part-time, we are still left with an army of 38,000, three times bigger than the development teams of Google and Microsoft combined. And SourceForge is just one free software community; most of the bigger teams use their own servers to manage and organize the development process.

The most important piece of free software is the Linux (pronounced *Lin-ex*) operating system, named after its founder Linus Torvalds, who started coding it in college. While Linux is generally not used on desktops today, it and other free software run on 60% of all websites, an increasing number of cellphones, and 75% of the world's top 500 fastest supercomputers:



*IBM's Blue Gene (pun intended) supercomputer runs a lightweight Linux on each compute node, and a complete Linux on its management nodes.*

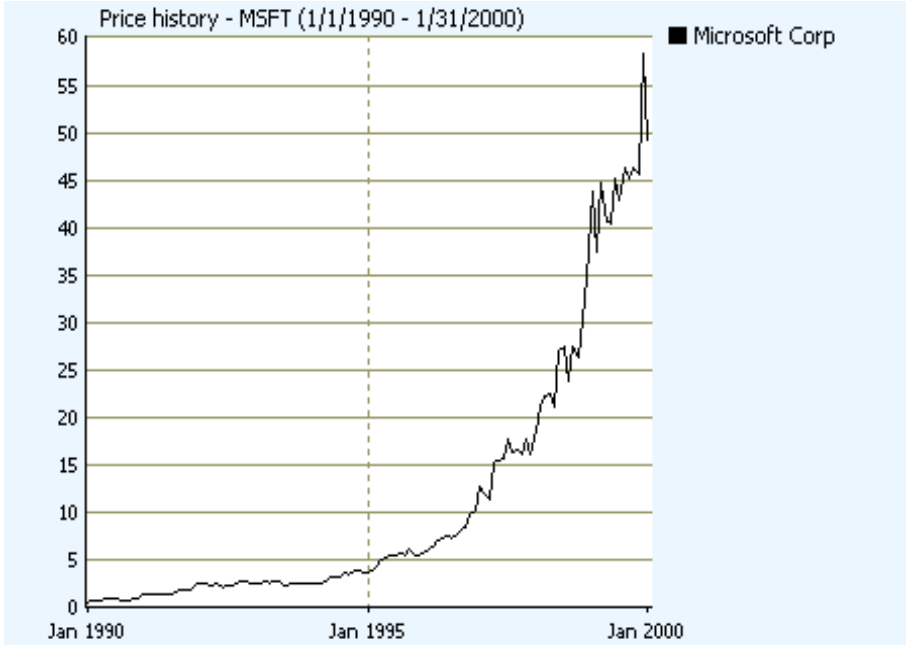
For its part, Microsoft has fiercely fought against Linux and the trend towards free software by pretending it is just another proprietary competitor. With \$28 billion in cash, dominant market share in Windows, Office and Internet Explorer, and an army of thousands of experienced programmers, Microsoft is a focused and enduring competitor.

Microsoft is the largest proprietary software company, but others have adopted its philosophy of hoarding all knowledge, no matter how irrelevant to their bottom line or useful to others. Google, the dominant player in Internet search, relies heavily on free software and considers it an important part of their success, but they are very secretive and protect nearly all the software *they* produce. They are a black hole of free software: innovation enters but never leaves.

This is all perfectly legal and ethical, and the free market gives everyone an unfettered right to innovate in any way, create any license agreement, and charge anything for a product. But free software is not just a competitor, it is a different way of creating software.



The free software community has long threatened to take over the world. Evangelist Eric Raymond once growled to a Microsoft VIP that he was their “worst nightmare.” That was in the mid-1990s, when Microsoft stock price was doing this:



*Microsoft stock price, 1990 – 2000*

A friend installed Linux in the mid-90s but he gave up because his Backspace key didn't work. Free software has come a long way since then, reaching technical critical mass, if not market dominance. This book will discuss the remaining technical challenges preventing world domination, but its inertia and ignorance are its biggest obstacles.

While this book presents a vision of the future, I believe we could have had these advancements decades ago. Free software's paradoxical success should also cause us to question other assumptions about copyright, patents, and other topics that this book will address.

## iBio

I first met Bill Gates at the age of twenty. He stood in the yard of his Washington lake-front home, Diet Coke in hand, a tastefully small ketchup stain on his shirt, which no one had the courage to point out, and answered our questions, in-turn, like a savant. As a

college summer intern, I had planned for a potential encounter and I approached him with questions that interested me but which would be arcane to non-computer mortals.<sup>2</sup>

His answers demonstrated that he was one of the top software experts on the planet and convinced me that I would be wise to start off my career at Microsoft.



*Writing software is a craft, like carpentry. While you can read books on programming languages and software algorithms, you can't learn the countless details of a craft from a book. You must work with experts on real-world problems. Before free software, you had to join a company like Microsoft.*

I joined Microsoft in 1993 when it was hitting its stride. It had recently released Windows 3.1 and Windows NT, setting itself on the path of more than a decade of dominance in the PC operating system market, and the many other markets that flow from it. I worked as a programmer for 11 years in a variety of different groups — on databases, Windows, Office, MSN, mobility, and research.

One day it just hit me — I should quit. There were no big reasons, only a lot of little ones. I had just launched v1 of the client and server side of the Microsoft Spot watch, and while it contained

---

<sup>2</sup> I asked him about the performance of Microsoft Exchange's database storage engine as compared to the one inside Microsoft SQL Server, and about NetWare's newly-announced clustering technology called SST Level 3.

sophisticated technologies, I didn't really believe it would take off in the marketplace. I had gained lots of knowledge yet only understood the Microsoft world. I was making decent money, but had no time to enjoy it. Though my boss was happy with me, I was losing motivation to just keep doing the same thing I had been doing for over a decade. When I looked around the company I saw a lot of ancient codebases and unprofitable ventures.

Like many of my fellow employees, I was only vaguely familiar with free software when I left and randomly decided to check out this thing called Linux. At Microsoft, I got all the software I wanted for free, and I always thought free software would be behind proprietary software. For 15 years I had made it a priority to learn about many aspects of Microsoft technologies, and my office contained rows of books on everything from *Undocumented Windows* to *Inside SQL Server*. When running Windows I felt as comfortable as Neo in the Matrix, without the bullets and leather, so while I was willing to look around, I was half-forcing myself and didn't want this little experiment to mess up my main computing environment.

Every technical decision was big for me: which version of Linux should I try? Should I get an extra machine or can I try dual-boot? Can I really trust it to live on the same hard drive as Windows? I got some tips and assurance from a Microsoft employee who had recently tried Linux, and with that, and the help of Google, I proceeded with the installation of Red Hat's Fedora Core 3.

While I came to not be all that thrilled with Fedora itself, I was floored merely by the installation process. It contained a graphical installer that ran all the way to completion, it resized my NTFS partition — which I considered a minor miracle, setup dual boot, and actually did boot, and let me surf the Web. I didn't have a clue what to do next, but the mere fact that this all *worked* told me more about the potential of Linux than anything I had read so far. You cannot, by accident, build an airplane that actually flies.

Over time, what impressed me the most about Linux was the power of it all. It came with tons of applications: Firefox, OpenOffice, GIMP, Audacity, Mono, MySQL, and many more for me to discover. The UI was simple, responsive, polished and customizable. Installing the Apache web server took just a few seconds and gave me access to a vast world of PHP. Installing the WordPress blog took me 15 minutes the first time, but I knew when I became more profi-

cient at things, I could do it in one. I came to understand that beyond its poorly debugged device drivers, a Windows computer is a sad joke. By mid-2005, I was in love with computers again!

I've spent three years in diligent research on the key subjects of this book, talking to hundreds of programmers, attending many conferences, and reading source code, magazines, websites and books. This book isn't really about the death of Microsoft as much as it is about the Microsoft proprietary development model that has pervaded or even infected computing. It is certainly not meant to be a bitter take on Microsoft's future although I believe they are toast. I loved working there, learned an enormous amount, and enjoyed the privilege of working alongside many brilliant minds. Like many things in life, it was fun while it lasted.

# GLOSSARY

**Bit:** A piece of information that can hold 2 values: 1 and 0. Bits are grouped into bytes of 8, characters of 2 bytes (Unicode), 4-byte numbers and pictures with lots.<sup>1</sup>

**Digitize:** Process of converting something into 1s and 0s. Once something is in a digital format, it can be infinitely manipulated by a computer.

**Software:** General term used to describe a collection of computer programs, procedures and documentation that perform tasks on a computer.

**Function:** The basic building block of software is a function, which is a discrete piece of code which accomplishes a task:

```
int SquareNumber (int n)
{
    return n * n;
}
```

**Machine language:** At the lowest level, software is a bunch of bits that represent an ordered sequence of processor-specific instructions to change the state of the computer.

**High-level language:** A programming language that looks more like English.

**Compiler:** Software that (typically) converts a high-level language into a machine language.

**Kernel:** The lowest level of an operating system that initializes and manages hardware.

**Hardware:** Physical interconnections and devices required to store and run software.

**Processor:** Hardware that executes the programmer's instructions.

**Hard drive:** Spinning magnetic platters where bits are stored even after the computer is turned off.

**Memory:** Hardware which provides fast access to bits of code and data for the processor. A processor can only manipulate data after it has loaded them into memory from the hard drive or network.

**URL (Uniform Resource Locator):** The textual location of a webpage, picture, etc. on the Internet. You can hand a URL to any computer in the world that “understands the Internet” and it would return the same thing. (It might notice that you prefer a version of the page in your language.) An e-mail address is also a URL. The only thing *everything* on the Internet has is a URL.

---

1 Like a number of places in this book, some of this text was taken from Wikipedia.



# WIKIPEDIA

A good friend of mine teaches High School in Bed-Stuy, Brooklyn – pretty much “the hood.” Try to imagine this classroom; it involves a lot of true stereotypes. But what does NOT fit the stereotype is that he started a class wiki, and has all his students contribute to it. Instead of a total mess, instead of abuse, graffiti and sludge, it's raised the level of ALL the students. It's a peer environment: once it becomes cool to do it right, to *be* right, abuse and problems dry up almost completely.

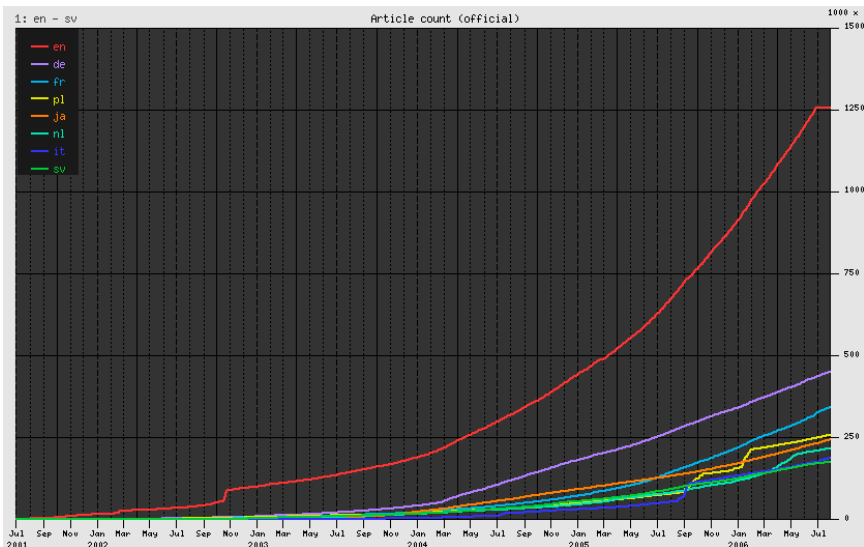
—Slashdot.org commentator

My school blocks Wikipedia entirely. When asked why, the answer is “anybody can edit it.” As opposed to the rest of the Internet which is chock-full of nothing but the highest quality, peer-reviewed content, written universally by the finest experts, hand selected from across the world?

—Slashdot.org commentator

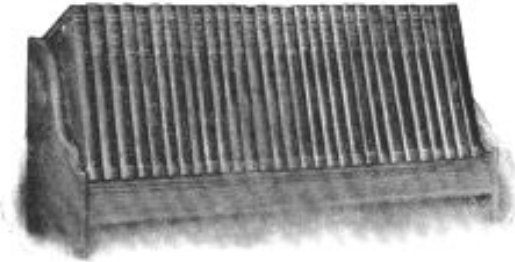
One of the great movements in my lifetime among educated people is the need to commit themselves to action. Most people are not satisfied with giving money; we also feel we need to work. That is why there is an enormous surge in the number of unpaid staff, volunteers. The needs are not going to go away. Business is not going to take up the slack, and government cannot.

—Peter Drucker, father of modern management



*Graph of the number of entries in the Wikipedias of various languages. This exponential growth is a confirmation of Metcalfe's law: the more users of Wikipedia, the better it gets, so more use it.*

# WHEN IN DOUBT—"LOOK IT UP" IN The *Encyclopaedia Britannica*



(New 11th Edition) issued 1910-11 by the  
CAMBRIDGE UNIVERSITY PRESS (England)

## The Sum of Human Knowledge

*29 volumes, 28,150 pages,  
44,000,000 words of text.  
Printed on thin, but strong  
opaque India paper, each  
volume but one inch in  
thickness.*

THE BOOK TO ASK QUESTIONS OF

FOR READING OR FOR STUDY

*Encyclopedia Britannica ad from 1913*

Compared to a paper encyclopedia, a digital edition has significant advantages. The biggest is cost, as printing and shipping a 50,000-page document represents an enormous expense in the production of an encyclopedia. The digital realm has other significant advantages: the content can be constantly updated and multimedia features can be incorporated. Why read about the phases of a 4-stroke internal combustion engine when you can watch one in action?

In the mid-1990s, Microsoft created Encarta, the first CD-ROM based digital encyclopedia. CDs were a natural evolution for Microsoft because it was shipping its ever-growing software on an increasingly large number of floppy disks. (Windows NT 3.1, released in 1993, required 22 floppies. CDs quickly became more cost-effective, as they hold 500 times more data, and are more reliable and faster, and Microsoft played an important role in introducing CD-ROM drives as a standard feature of computers.)

While CDs hold more data than floppies and are an important technological advancement, this development was soon eclipsed by the arrival of the web. Users could connect to a constantly-updated encyclopedia of unlimited size from any computer without installing it first.

Unfortunately for Microsoft, the Encarta team was slow in adopting the Internet because they felt some of the richness of its encyclopedia was lost on the web. However, with pictures, animations

and text, even the early web was good enough and had substantial advantages over a CD-ROM version. In the Internet realm, you only need one Wikipedia, albeit running on hundreds of servers, for the entire world; you don't even need to worry about the cost to "make a copy" of an encyclopedia.

However, the biggest mistake the Encarta team made was not realizing that the Internet could introduce feedback loops. The users of an Internet encyclopedia can also become enhancers of it. If I have a question about what I've read, or I think I've found a problem, I can post a question or fix the problem and report what I've accomplished.

We will discuss later if the ability for anyone to edit, enhance or add data will hurt quality, but it is important to remember that it was the creation of the Internet that allows people in all the corners of the world to work together and learn from each other; a completely new capability for man.

For its faults, Wikipedia became larger than the Encyclopedia Britannica in just 2.5 years. The database now contains more than 15 times as many articles, and is already the best compendium of human knowledge ever created. No corporation invested millions of dollars in engineering or marketing either; it happened seemingly on its own. Even if some of those articles are fluff about Star Trek characters, many are not: Wikipedia's article on carbon nanotubes and many other scientific topics is more detailed and more up to date than Encyclopedia Britannica's.

Wikipedia is one of the 10 most popular websites on the Internet, receiving 450 times the traffic of Encyclopedia Britannica, and with an article collection that continues to grow at an exponential rate. As Wikipedia has advanced, it has also added a multimedia collection, a dictionary, a compendium of quotes, textbooks, and a news aggregator — and they are just getting started.

In some ways, access to a search engine might seem to obviate the need for an encyclopedia. But while search engines provide a keyword index to the Internet, they do not replace the importance of an encyclopedia: a comprehensive, coherent, neutral, compendium of human knowledge.

Imagine that you wanted to research a topic like nuclear power. Where would you go to get an unbiased opinion: the government? Greenpeace? CNN? Some schools have banned Wikipedia, but sec-

ondary sources have long been disallowed. Even so, the article text and links to primary sources can be a useful place to start research on a topic.

While Wikipedia is a powerful resource, what is more amazing is that it is built with the same surplus intellectual energy that others spend on crossword puzzles or Sudoku. Wikipedia provides an additional outlet for people's energy, and something much greater than any one person, or even one company, could accomplish.

A key element of Wikipedia's success is that its founders created a community that people enjoyed working in, and this enjoyment factor brought in even more people. This is a hard thing to do, and it starts with an inspirational vision.

There is no monster multinational corporation behind Wikipedia. There was no CEO who pounded the table and said he wanted to create the biggest encyclopedia ever. Its annual budget is \$5,000,000, most of that goes to funding hardware, bandwidth and the salary of the very tiny six-person operations team that keeps the few hundred servers running. Maybe you haven't edited Wikipedia yet, but millions of other registered members, and unregistered users, have created and improved it over the years. I've made a few fixes to a few places — it is very easy!

Some may wonder about the susceptibility to inaccuracies and vandalism of something as widely collaborative as Wikipedia. Fortunately, this digital graffiti does not threaten to spoil things for two important reasons: accountability and pride. The good thing about these shared efforts is that they provide a way to appreciate the importance of one's fellow man.

Every change made to the encyclopedia is permanently logged and publicly recorded in a version control mechanism similar to that used in software; in fact, no changes are irreversible. Unlike graffiti, which can take hours to clean up, dumping unwanted changes or banning users takes mere seconds which is a great discouragement.

Ultimately, part of believing in the viability of a free encyclopedia requires belief in the fundamental goodness of mankind. One must trust that the amount of people in the world who gain satisfaction from making a positive contribution to a product far outnumbers those who enjoy a few seconds of perverted pride in temporary defacement. Moreover, with millions of registered users, there is a virtual guarantee that problems will get noticed.

The obvious vandalism is easily caught and removed, but there are more subtle forms of vandalism that are much harder to catch. In fact, who is to say whether or not any edit is accurate?

Wikipedia has insulated its product from inaccuracies by implementing three content policies:

1. **No original research:** Articles should reference published, reliable sources. The threshold for “reliable” is debatable, but in practice, this is not a significant obstacle.
2. **Neutral point of view:** An article should fairly and, without bias, represent all significant views that have been published by reliable sources.
3. **Verifiability:** The threshold for inclusion in Wikipedia is verifiability. Verifiable means that a reader should be able to determine whether material added to Wikipedia has already been published by a reliable source.

*That the community accepts these concepts is a key to Wikipedia's success.*

By making these policies an integral part of the culture, Wikipedia created something not necessarily perfectly accurate, but from reputable, verifiable resources which makes it good enough that other people decided it is worth reading and contributing to.

There have been objective studies that have demonstrated that Wikipedia is high quality, comparable to Encyclopedia Britannica. In general, its greatest challenge is in political articles where emotions run high and facts are disputed. Some scientists say Global Warming provides an imminent danger to humanity, while others say it is a hoax, and Wikipedia cannot resolve this contradiction between published, reliable sources.

Even for the cynics who believe that the vandals may still win, consider that since its creation in January 2001, Wikipedia has remained as much an encyclopedia as a self-organizing technological and social experiment. As Wikipedia evolves, tools are being created to detect and remove vandalism, and tag articles that don't conform to style guidelines. Sometimes articles have various warnings about how it is a work in progress, which is a useful warning. Every page also has a discussion page where issues are debated before the content itself is updated.

In short, Wikipedia is an evolving relationship between people and their software. For example, should anonymous users be allowed to make edits? Many believe they should not because anonymity decreases accountability. This is an ongoing discussion.



Wikipedia is free to read, and a study suggested that it could generate up to \$100 million per year in advertising revenue. One day, they might choose to, and could use this money in any number of ways: from acquiring proprietary content like maps, legal documents, and document templates, and making them free, to hiring employees to charge ahead in areas underfunded by the community.

Eric Raymond, in his book *The Cathedral and the Bazaar*, analogizes the free software development model to a bazaar – a disorganized conglomeration of input and ideas. That's an unsatisfactory image, however, because it suggests something primitive and disorganized. Cathedrals took hundreds of years to build, but in less than 10 years, Wikipedia has produced a larger, and more comprehensive product than its much older competitors. It is better to think of this free software product as an already very polished cathedral in its early years of development.

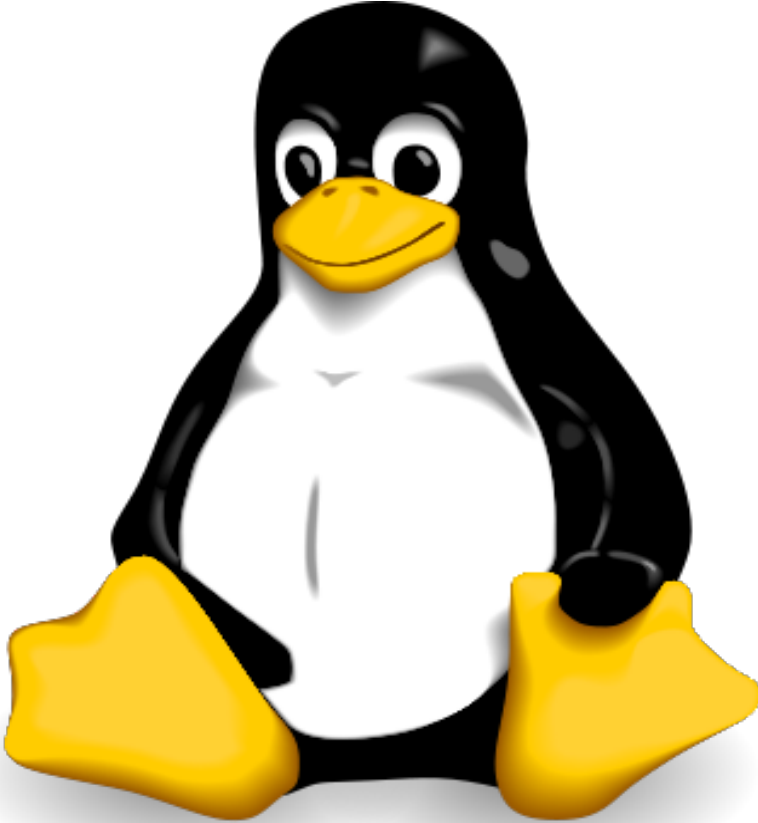
What else can independent, highly co-operative free software communities build? The answer is an infinite number of things. Specifically, an entire software stack that is as free as Wikipedia and uses zero Microsoft software.

The software used to run Wikipedia is an afterthought to many, but it is a significant piece of technology. While Wikipedia and its software won't make a large dent in Microsoft's profits, the Linux kernel is a mortal threat.

# LINUX

Really, I'm not out to destroy Microsoft. That will just be a completely unintentional side effect.

—Linus Torvalds, 2003



*The Linux mascot, Tux, created by Larry Ewing*

**T**he kernel of an operating system (OS) is the central nervous system of a computer. It is the first piece of software that the computer executes, and it manages and mediates access to the hardware. Every piece of hardware needs a corresponding kernel device driver, and you need *all* of your drivers working before you can run *any* of your software. The kernel is the center of gravity of a software community, and the battle between free software and Windows is at its lowest level a battle between the Linux and Win-

dows kernels. Microsoft has said that it has bet the company on Windows, and this is not an understatement! If the Windows kernel loses to Linux, then Windows, and Microsoft, is also lost.<sup>1</sup>

The Linux kernel is not popular on desktops yet, but it is widely used on servers and embedded devices because it supports thousands of devices and is reliable, clean, and fast. Those qualities are even more impressive when you consider its size: printing out the Linux kernel's 8,000,000 lines of code would create a stack of paper 30 feet tall! The Linux kernel represents 4,000 man-years of engineering and 80 different companies, and 3,000 programmers have contributed to Linux over just the last couple of years.

That 30-foot stack of code is just the basic kernel. If you include a media player, web browser, word processor, etc., the amount of free software on a computer running Linux might be 10 times the kernel, requiring 40,000 man-years and a printout as tall as a 30-story building.

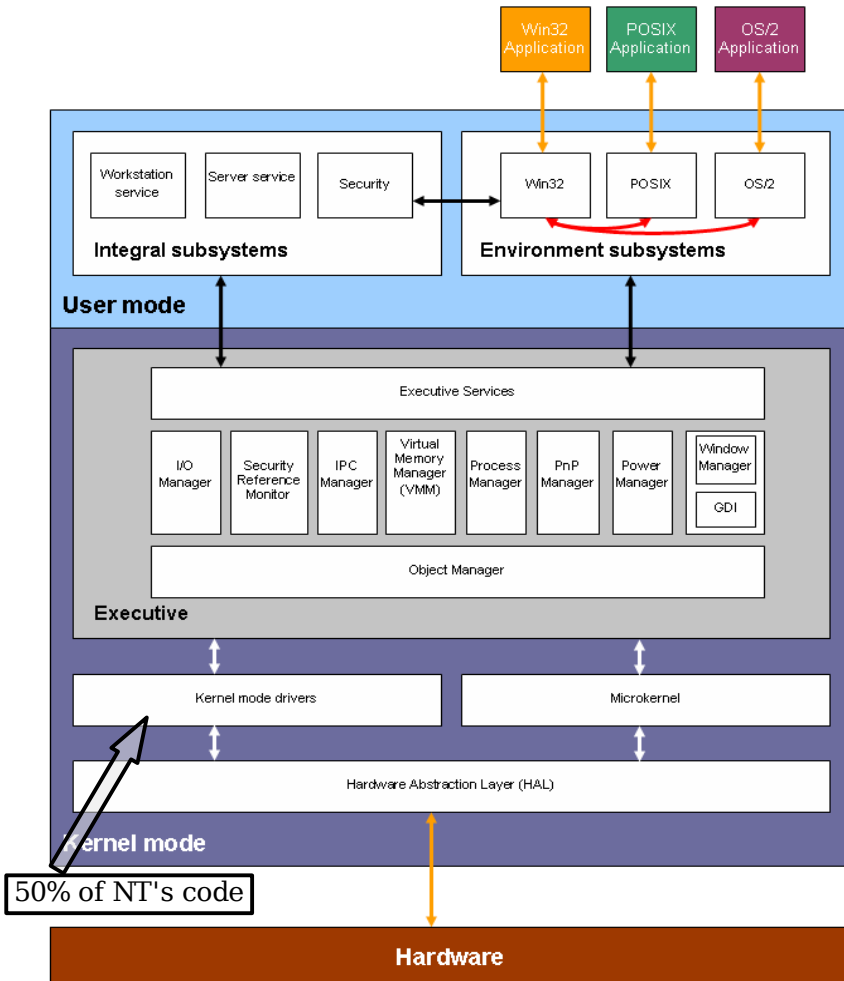
This 40 man-millennia even ignores the work of users reporting bugs, writing documentation, creating artwork, translating strings, and performing other non-coding tasks. The resulting Linux-based free software stack is an effort that is comparable in complexity to the Space Shuttle. We can argue about whether there are any motivations to write free software, but we can't argue it already exists!

One of the primary reasons I joined Microsoft was I believed their Windows NT (New Technology) kernel, which is still alive in Windows Vista today, was going to dominate the brains of computers, and eventually even robots. One of Bill Gates' greatest coups was recognizing that the original Microsoft DOS kernel, the source of most of its profits, and which became the Windows 9x kernel, was not a noteworthy engineering effort. In 1988, Gates recruited David Cutler from Digital Equipment Corporation, a veteran of ten operating systems, to design the product and lead the team to build the Windows NT kernel, that was released as I joined in 1993.

---

1 While cloud computing, the movement of increasing number of applications and services provided over the Internet, is one of the hot topics of today, it is unrelated to the Windows vs. Macintosh vs. Linux war that is going on. Even in a future where applications like word-processing are done over the Internet, you still need a kernel, a web browser, a media player, and so forth.

The kernel Cutler and his team developed looks like this:



*Windows NT kernel architecture block diagram. Cutler had a Windows 95 doormat outside his office; you were encouraged to wipe your feet thoroughly before entering.*

Unfortunately for Microsoft, the original kernel lived on through Windows 95, Windows 98, and into Windows Me. (Microsoft also had Windows CE, a small kernel for embedded devices. Microsoft had three separate kernels for most of my tenure, whereas the same Linux kernel is used on small and big devices.)

Windows has become somewhat popular for servers and devices, but it never achieved the dominance it did on desktop PCs. Perhaps the biggest reason is that its code wasn't available for others to extend and improve upon. The Linux kernel took off because there

are people all over the world, from Sony to Cray, who tweaked it to get it to run on their hardware. If Windows NT had been free from the beginning, there would have been no reason to create Linux. However, now that there is the free and powerful Linux kernel, there is no longer any reason but inertia to use a proprietary kernel.

There are a number of reasons for the superiority of the Linux kernel. But first, I want to describe the software development process. When you understand how the Linux kernel is built, its technical achievements are both more impressive and completely logical.



# Distributed Development

In Linux we reject lots of code, and that's the only way to create a quality kernel. It's a bit like evolutionary selection: breathtakingly wasteful and incredibly efficient at the same time.

—Ingo Molnar, Linux kernel developer



[Diego Novillo](#)

[Red Hat Canada](#)

[GCC - An Architectural Overview, Current Status and Future Directions](#)



[Ram Pai](#)

[IBM Corporation](#)

[Shared-Subtree Concept, Implementation and Applications in Linux](#)



[Venkatesh Pallipadi](#)

[Intel Corp](#)

[ondemand 2.0: A "New and Improved" Dynamic CPU Frequency Governor](#)



[ChanJu Park](#)

[Samsung Electornocs Co.](#)

[Linux Bootup Time Reduction for Digital Still Camera](#)



[Nick Piggin](#)

[SUSE Labs, Novell](#)

[A Lockless Pagecache in Linux - Introduction, Progress, Performance](#)



[Ian Pratt](#)

[University of Cambridge Computer Laboratory](#)

[Xen 3.1 and the Art of Virtualization](#)



[Tony Reix](#)

[Bull SAS](#)

[NFSv4 Test Project.](#)

[Jose R Santos](#)

[IBM](#)

[Improving the Approach to Linux Performance Analysis](#)



[Joel H Schopp](#)

[Resizing Memory With Balloons and Hotplug](#)



[Martin Schwidefsky](#)

[IBM](#)

[Collaborative Memory Management in Hosted Linux Systems](#)



[Suresh Siddha](#)

[Intel Corp](#)

[Effective Load Balancing Using Processor Performance States](#)



[Kay Sievers](#)

[Dynamic Device Handling on the Modern Desktop](#)

*A portion of the speaker list for the 2006 Linux Kernel Symposium the author attended. Linux kernel development is a distributed effort, which greatly enhances its perspective.*

Every 20<sup>th</sup> century management book I've read assumes that team members work in the same building and speak the same language.

Microsoft's corporate culture was based on the theory that software development was a collaborative effort that should be centralized so that people could work together. As a result, most Microsofties, especially the programmers, were based in Redmond because that was where all the other engineers were located.

Microsoft had a very open development model inside the company: developers would periodically switch teams, collaborate freely together on unreleased code, and join e-mail discussion groups with engineers in similar product roles. These resources of collaboration are some of the many which are unavailable to those outside.

The Internet, which was born when Microsoft was a mature company, has changed countless aspects of our lives, including the way software is developed. Without the Internet, free software could never exist because the developers wouldn't be able to work together. (Microsoft uses the Internet to do more development in other places, although it is still primarily in Redmond.) Microsoft grew up before the birth of the web, and thus has yet to fully embrace this distributed process. For example, Microsoft's bug databases aren't available on the Internet. Microsoft isn't taking the maximum benefit from the knowledge gained by its users because it doesn't have as many feedback loops.

Linux has achieved enormous gains against Windows, even while, from a 20<sup>th</sup> century manager's perspective, organizing the Linux kernel is a worst-case example for building a productive, prosperous organization. Where are the team-building exercises? The three-year planning retreats? It should be amazing that Linux releases anything at all, let alone dominates the supercomputer business.

Ingo Molnar's quote above appears contradictory at first glance, but it is not. Linux receives lots of different ideas; many get rejected, but what remains incorporates the best of all the ideas. The Internet allows you to quickly evolve towards optimal solutions with feedback loops of discussions, tests, and bug reports. (Tests are very important because they give you objective numbers. If you want to see if the new disk cache code is faster, you can compile the kernel, which is a very disk-intensive task, and time the result.)

In the free software movement, the battles aren't between empires, but rather between engineers fighting over technical details — testosterone-laden VIPs are irrelevant. The Linux kernel community has taken the idea of a meritocracy to the next level. All changes to the official Linux kernel must go through Linus, and his Lieutenant, Andrew Morton, and then the relevant subsystem main-

tainer — but first, the proposed change has to go through everyone else! All Linux changes are posted to a mailing list where anyone can comment and give opinions. Linus wrote:

The contributors for any given project are self-selected. Someone pointed out that contributions are received not from a random sample, but from people who are interested enough to use the software, learn about how it works, attempt to find solutions to problems they encounter, and actually produce an apparently reasonable fix. Anyone who passes all these filters is highly likely to have something useful to contribute.

Linus' primary job is to provide technical expertise. He once said his job was to keep out bad code, and that might be enough. Let the varied users of Linux take it to new places, while he will make sure no one is screwing up the existing code along the way.

Linus was asked whether the unpolished nature of a large group of programmers, with disparate backgrounds, created a dreary situation that made him want to go back and work in private, and he said:

I actually *like* arguing (sometimes a bit too much), so the occasional flame-fest really does nothing but get me pumped up.

At the same time, I'm actually pretty good at just "letting it go", once I've argued enough and am bored with the argument. Part of that is also having to occasionally just admit that you were wrong, and have the ability to send out a "mea culpa" e-mail just saying so.

I tend to care much more about improving the general development model than about the details of some particular subsystem. So that tends to make it easier for me to "let go." I'll state my opinions, but even if I'm convinced I'm right, if I'm not actually willing to write the code, in the end I'll happily be overridden by the people who *do* write the code.

This is obviously very much a matter of personality.

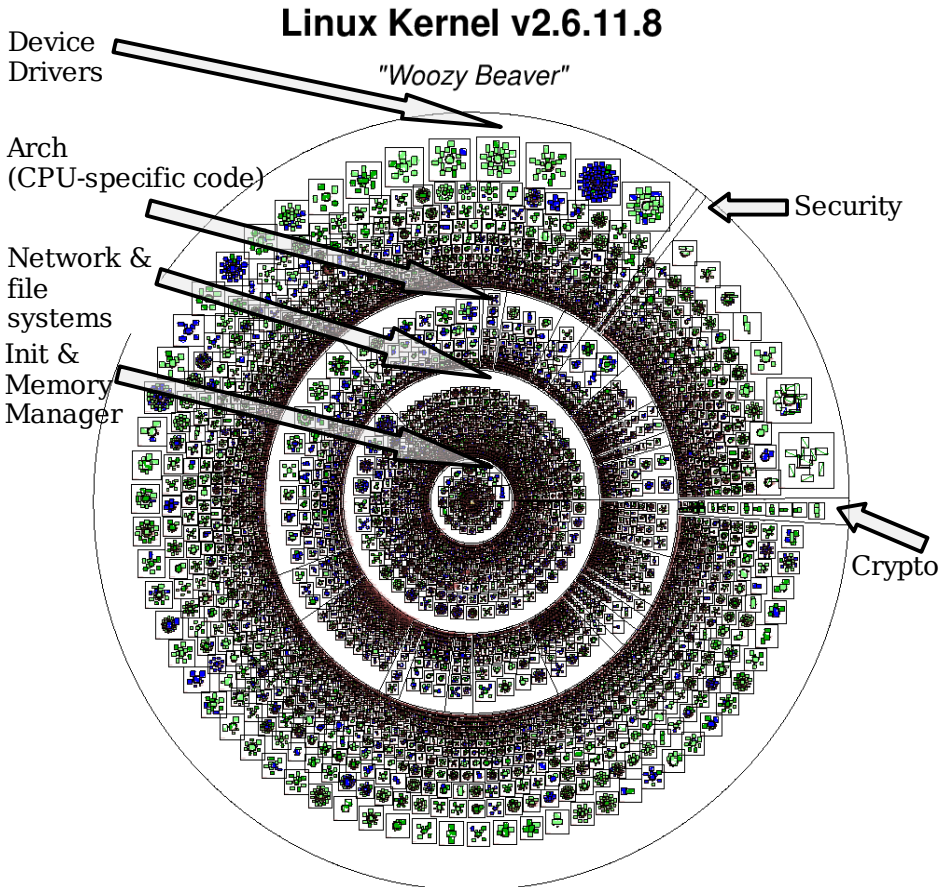
There are things that I tend to worry about, and that can be really painful, but they are pretty rare. The classic example is the old "Linus doesn't scale" argument, where it ended up being the case that I really had to fundamentally change the tools I use and how I work. And *that* was a lot more painful than worrying about the actual code.

# Linux Kernel Superiority

Here are the reasons Linux is superior to the Windows kernel:

## 1. Refactored Code (Reliability)

Here is a diagram of the Linux kernel:



OREGON STATE **LINUX** USERS GROUP  
lug.oregonstate.edu

*Layers of the Linux kernel "onion". The Linux kernel is 50% device drivers, and 25% CPU-specific code. The two inner layers are very generic.*

Notice that it is built as an onion and is comprised of many discrete components. The outermost layer of the diagram is device drivers, which is 50% of the code, and more than 75% of its code is hardware-specific. The Microsoft Windows NT kernel diagram, shown several pages back, puts all the device drivers into a little box in the lower left-hand corner, illustrating the difference between

theory and reality. In fact, if Microsoft had drawn the kernel mode drivers box as 50% of the Windows NT diagram, they might have understood how a kernel is mostly hardware-specific code, and reconsidered whether it was a business they wanted to get into.

Refactoring (smoothing, refining, simplifying, polishing) is done continuously in Linux. If many drivers have similar tasks, duplicate logic can be pulled out and put into a new subsystem that can then be used by all drivers. In many cases, it isn't clear until a lot of code is written, that this new subsystem is even worthwhile. There are a number of components in the Linux kernel that evolved out of duplicate logic in multiple places. This flexible but practical approach to writing software has led Linus Torvalds to describe Linux as "Evolution, not Intelligent Design."

One could argue that evolution is a sign of bad design, but evolution of Linux only happens when there is a need unmet by the current software. Linux initially supported only the Intel 80386 processor because that was what Linus owned. Linux evolved, via the work of many programmers, to support additional processors — more than Windows, and more than any other operating system ever has.

There is also a virtuous cycle here: the more code gets refactored, the less likely it is that a code change will cause a regression; the more code changes don't cause regressions, the more code can be refactored. You can think about this virtuous cycle two different ways: clean code will lead to even cleaner code, and the cleaner the code, the easier it is for the system to evolve, yet still be stable. Andrew Morton has said that the Linux codebase is steadily improving in quality, even as it has tripled in size.

Greg Kroah-Hartman, maintainer of the USB subsystem in Linux, has told me that as USB hardware design has evolved from version 1.0 to 1.1 to 2.0 over the last decade, the device drivers and internal kernel architecture have also dramatically changed. Because all of the drivers live within the kernel, when the architecture is altered to support the new hardware requirements, the drivers can be adjusted at the same time.

Microsoft doesn't have a single tree with all the device drivers. Because many hardware companies have their own drivers floating around, Microsoft is obligated to keep the old architecture around so that old code will still run. This increases the size and complexity of the Windows kernel, slows down its development, and in some cases reveals bugs or design flaws that can't even be fixed. These

backward compatibility constraints are one of the biggest reasons Windows takes years to ship. The problem exists not just at the driver layer, but up the entire software stack. When code isn't freely available and in one place, it makes it hard to evolve.

While the *internal* logic of Linux has evolved a lot in the last ten years, the *external* programmer interfaces have remained constant. The key to a stable interface is incorporating the right abstractions. One of the best abstractions that Linux adopted from Unix is the file abstraction. In order to perform almost any function on a Linux computer, from reading a web page on a remote website to downloading a picture from a camera, it is necessary to simply use the standard file commands: open and close, read and write.

On my computer, in order to read the temperature of the CPU, I just need to open the (virtual) text file `"/proc/acpi/thermal_zone/THM0/temperature"` and the data I request is inside:<sup>2</sup>

|                           |                   |
|---------------------------|-------------------|
| <code>temperature:</code> | <code>49 C</code> |
|---------------------------|-------------------|

In essence, the Linux kernel is a bundle of device drivers that communicate with hardware and reveal themselves as a file system. As new features, security issues, hardware requirements and scenarios confront the Linux kernel, the internal design evolves and improves, but the file system abstraction allows code outside the kernel to remain unchanged over longer periods of time.

---

<sup>2</sup> This should arguably be expressed as XML, but because there is common code that reads these values and provides them to applications, and because each file contains only one value, this problem isn't very significant; the kernel's configuration information will never be a part of a web mashup.

Here is a random sample of the change log of the Linux kernel from 2.6.14. As you can see, it is filled with all kinds of cleanup and bugfix work:

```
spinlock consolidation
fix numa caused compile warnings
ntfs build fix
i8042 - use kzalloc instead of kcalloc
clean up whitespace and formatting in drivers/char/keyboard.c
s3c2410_wdt.c-state_warning.patch
[SCSI] Fix SCSI module removal/device add race
[SCSI] qla2xxx: use wwn_to_u64() transport helper
[SPARC64]: Fix mask formation in tomatillo_wsync_handler()
[ARCNET]: Fix return value from arcnet_send_packet().
```

*Many of the Linux kernel's code changes are polish and cleanup. Clean code is more reliable and maintainable, and reflects the pride of the free software community.*

If you look at the code changes required to make a bugfix, in the vast majority of cases all that is needed is a revision of a few lines of code in a small number of files. A general guideline Linux has for bugfixes is this: if you can't look at the code change and prove to yourself that it fixes the problem, then perhaps the underlying code is confused, and this fix shouldn't be added near the end of a release cycle.

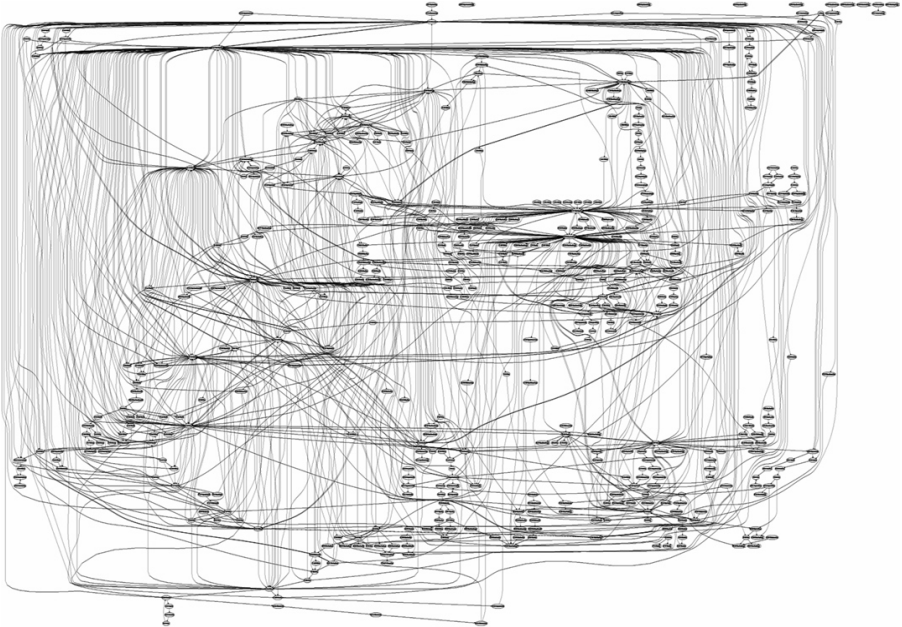
According to [Stanford University researchers](#), the Linux kernel has .17 bugs per 1,000 lines of code, 150 times less than average commercial code containing 20-30 bugs per 1,000 lines.<sup>3</sup> Microsoft's Windows bug databases aren't available on the Internet so it is impossible to make comparisons, but even if Linux isn't more reliable already, it is setup to become so because the code is simple, well-factored, and all in one place.

Within the free software community, different teams are disparate entities, and so the the idea of arbitrarily moving code from one part of the system to another can't easily happen. Inside Microsoft there are no boundaries, and so code is moved around for short-term performance gains at the cost of extra complexity.

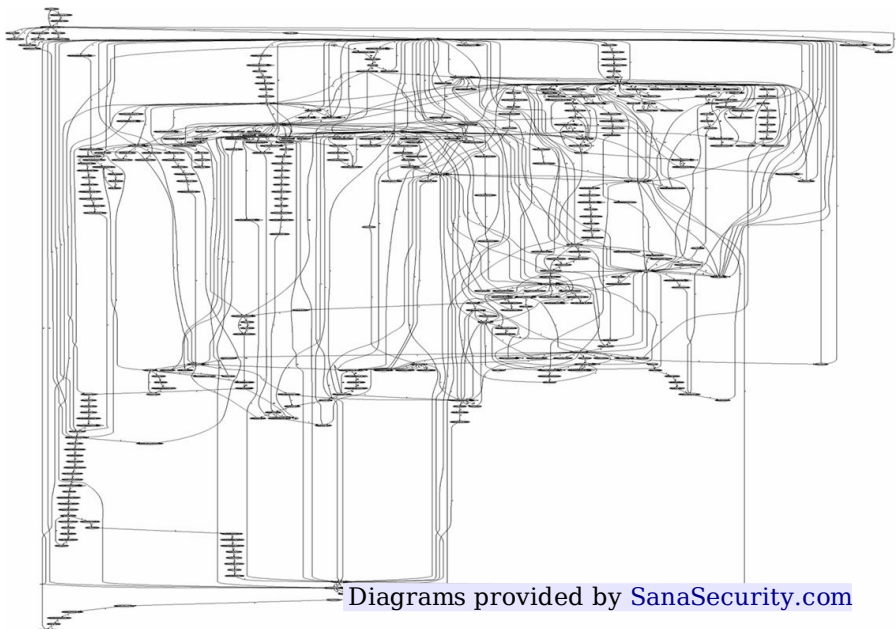
---

3 These studies have limited value because their tools usually analyze just a few types of coding errors. Then, they make the IT news, and get fixed quickly because of the publicity, which then makes the study meaningless. However, these tools do allow for comparisons between codebases. I believe the best analysis of the number of Linux bugs is the 1,400 bugs in its bug database, which for 8.2 million lines of code is .17 bugs per 1,000 lines of code. This is a tiny number, though it could easily be another 100 times smaller. Here is a link to the Linux kernel's active bugs: <http://tinyurl.com/LinuxBugs>.

Here is a graph of all the function calls into the OS required to return a simple web request. These pictures demonstrate a visual difference in complexity that often exists between free and proprietary software:



*System call graph in Microsoft's proprietary web server, IIS.*



Diagrams provided by [SanaSecurity.com](http://SanaSecurity.com)

*System call graph to return a picture in the free web server Apache.*



## 2. Uniform Codebase (Reliability, Maintainability, and Usability)

My job is to say “no”, to some extent. If you keep things clean enough and have some kind of requirement for what code can look like, you’re already ahead of the game.

—Linus Torvalds

Linux engineers have found a way to run the same codebase on a wide variety of processors, on devices from cellphones to supercomputers, an unprecedented achievement. Linux has been tuned to first, run correctly, and then run efficiently on two, four, eight, and now 1,000 processor machines. Software has infinite malleability, so such a universal kernel has always been possible — it just took a bunch of different hardware companies working together to make it happen.

Putting everything into one codebase helps reliability. Running the Linux kernel on a 32-processor computer shakes out multi-threaded bugs far more quickly than on a two-processor laptop. Running on low-end machines keeps the code small and simple, which makes it run faster on desktops. Features that first appear on laptops and tablets eventually trickle their way down to even smaller devices where the code undergoes even more testing and enhancement. The many hardware and server developers who want extreme reliability ensure that the kernel on my PC is as reliable as Linux’s most demanding customer.

Linux is more flexible than the Windows NT kernel, though both are very clean and flexible. For example, the National Security Agency (NSA) has created a free software component called SELinux (Security Enhancements to Linux) that adds a strong security enforcement mechanism known as Mandatory Access Control.<sup>4</sup> Making these mechanisms public helps ensure there are no back

---

4 This is a way to add additional security because the operating system can say, for example: Because a media player has no reason to write files to disk, the system can take away this permission. Before the kernel tries to do anything interesting, it will ask the Mandatory Access System (MAC) whether such an operation is allowed. The security checks in most other operating systems simply ask if the *person* is allowed to do something.

Creating a default policy adds additional work for application writers, and by itself doesn’t entirely solve the problem. A word processor needs complete read and write access, so how do you solve the problem of a virus in a document macro opening all of your files and writing junk? SELinux doesn’t deal with this situation because it doesn’t have this information. In GC programming languages, it is possible to walk the stack and determine more information about whether a macro, or the word processor itself, is asking to open a file.

doors to the NSA's computers. I will discuss in a later chapter why governments can adopt free software, even for high-security scenarios, but it appears the NSA already understands this concept.

Throughout the Linux world, one has many more choices for just the right tool to do the job. Some might argue that too much choice is a bad thing, but creating many components forces clear boundaries, and survival of the fittest whittles down the inferior choices over time.

### 3. Frequent Ship Cycles (Maintainability and Usability)

Microsoft had a motto: “Ship early, ship often.” This philosophy is wise for software development because it forces teams to maintain a high-quality product every day, and the earlier you release, the sooner you can receive and incorporate feedback.

However, this philosophy only works when adopted. Unfortunately, Microsoft's two biggest products, Windows and Office, do not follow this philosophy. Of course, paying \$240 every year for the latest upgrade to Windows “Ultimate” wouldn't be acceptable either!

The Linux kernel ships every three months. For a product of its size and complexity, Linux's rate of shipment is unprecedented. This pace has allowed the Linux kernel to ship drivers before Windows, and even before the hardware itself has shipped. Linux supported USB 3.0 before Microsoft, which did not include it in Windows 7. Because Linux is constantly near shipping, you can take any random build from Linus' computer, put it into a rocket and feel quite confident that it won't crash.

A big part of the Department of Justice lawsuit against Microsoft focused on the company's bundling of many software components with their operating system. The government accused Microsoft of excluding third-party software developers and stifling competition. But Microsoft's tying has been both a blessing and a curse.

The blessing is that by having the pieces work together, they can reuse code and be more integrated. The curse is that Microsoft has created a situation whereby it revises and adds new features to all of its interdependent components at the same time. As a consequence, its components take years to stabilize, and you can't ship until the last component is ready.<sup>5</sup>

---

5 The alternative is for each component to use the previous version of all of its dependent components, which means that the features in the latest Internet Explorer wouldn't show up in various places that are using the old version. However, does the Help system need the latest version?

By contrast, in a free OS, software components depend only on released versions. Every team doesn't try to ship on the same day, so the OS contains the latest versions of all released components.<sup>6</sup> Separate development organizations have enforced boundaries that have simplified dependencies, and allows everyone to move ahead at whatever pace they choose.

Many users wonder whether free software will ever be as good as proprietary software because presumably the free software guys can't afford things like usability studies. The problem with this thinking is that usability studies don't matter if you can't incorporate the changes quickly and easily.

The Internet, and all its communications mechanisms, provides a much bigger and richer feedback mechanism than any usability study. With clean codebases and frequent ship cycles, usability will happen automatically. I've spent a lot of time using Linux and find many applications are perfectly usable. This assumption about free software is contradicted by the facts.

Shipping a new platform every five years in theory provides Microsoft's partners a stable platform upon which to build. However, in reality, its results have flaws. For instance, I wasn't able to install an HP Photodesk 7960 printer driver on Windows Server 2003, although the driver installed perfectly on XP. The subtle and unintentionally undocumented differences between those operating systems, which shipped two years apart, has created compatibility headaches even on Microsoft's supposedly uniform platform.

## 4. Lower Development Costs (Maintainability)

It is much less expensive for hardware vendors to support Linux. If you want to build a device driver, a great place to start is by looking at existing shipping device drivers, an opportunity that Linux offers to everyone. A proprietary "Device Driver Toolkit" with its sample code is never as good as production code. Those expensive kits contain documentation, but not source code — so you sometimes have to guess at what is happening down below.

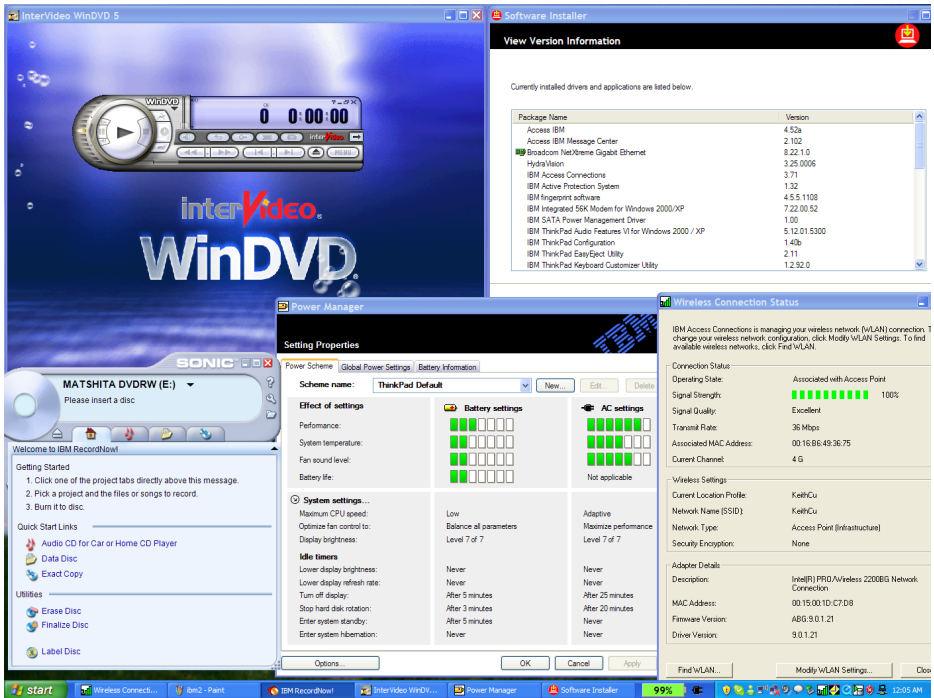
We find in Windows today that hardware manufacturers have duplicated a bunch of the functionality Windows provides but doesn't quite fit their needs. For example, IBM includes its own applet and status icon for wireless Internet, so Windows XP on IBM hard-

---

<sup>6</sup> Some components contain multiple versions to allow for a transition period.

ware has two. Presumably they weren't satisfied with the features Windows provided, and weren't able to fix them. And so they had to build new applets from scratch!

Here are five of the 100 applets IBM adds to Windows:



*Windows XP with 5 of IBM's 100 extra applets. Notice the large number of status icons on this almost-virgin installation.*

Building all of these applets, designing multilingual user interfaces, providing the means to install and configure, etc. is ten times more work than merely writing the device driver, leveraging other shipping drivers, and uploading it to the official codebase.

The reason my Photodesk printer driver didn't work on Windows Server 2003 was because of a crash in the installation code — which HP shouldn't even be bothering with in the first place.

## 5. Security (Reliability and Maintainability)

To mess up a Linux box, you need to work *at* it; to mess up your Windows box, you just need to work *on* it.

—Scott Granneman

Attempting to compare the security of operating systems is a complicated endeavor because there are an infinite number of risks.

It is like asking whether one car is safer than another. Against what: Getting hit from the front? Rolling over? Having the gas tank pierced? Its ability to brake?

Furthermore, neither Windows nor Linux are perfectly secure operating systems; both are prone to buffer-overflow viruses, an issue I will discuss in the tools chapter. Furthermore, new threats appear over time, and so each nefarious advancement requires new logic to defend against it. Given these caveats, it is still possible to make some comparisons.

Some of the previous advantages of Linux, such as its simplicity, modularity, support for SELinux, etc. all help with its security. In addition, the nature of how Linux is developed also helps. A document commissioned by the US Military said:

**Open source software is potentially subject to scrutiny by many eyes**

Therefore bugs, security flaws, and poor design cannot hide for long, at least when the software has a community of programmers to support it. And since fixing the code doesn't depend on a single vendor, patches are often distributed much more rapidly than patches to closed source software.

**Can increase code quality and security**

With closed source software, it's often difficult to evaluate the quality and security of the code. In addition, closed source software companies have an incentive to delay announcing security flaws or bugs in their product. Often this means that their customers don't learn of security flaws until weeks or months after the security exploit was known internally.

**—Open Technology Development Roadmap**

Another big difference between Linux and Windows is that Linux was adapted from Unix, which had a multiuser design right from the beginning. In Windows, users have historically had full Administrator access to the machine, including the ability to overwrite system files. When the computer is attacked by a virus, the virus can gain the same capabilities as the user and thereby hide itself inside system files, which make it very difficult to remove. On Linux, I can write only to my own files and have read-only access to all others. Linux's multiuser nature, and therefore its focus on minimal permissions, minimizes any damage.

One study found that there are about 60,000 known viruses for Windows, and only about 40 for Linux. Another study by *Evans Data* showed that 8% of Linux developers say their machines have been infected by malicious code, compared to 60% of Windows machines.

Brian Krebs of the *Washington Post* found that code to exploit flaws in Internet Explorer 6 existed for 284 days in 2006, while Firefox was at risk for just 9. Computer security expert Bruce Schneier recommended in December 2004 that people not run Internet Explorer. Some argue that Linux and Firefox have fewer viruses because they have fewer users, but Apache is well-respected for its security.

For most of my tenure at Microsoft, we worried about features, reliability, and performance, not security. Microsoft's Chief Research and Strategy Officer, Craig Mundie, said in 2002:

Many of the products we designed in the past have been less secure than they could have been because we were designing with features in mind rather than security.

Microsoft has greatly increased its focus on security in the past few years, and I am sure the security of every product improves with every release, but the [baggage](#) of their codebases serve as an ongoing impediment.

Having browsed through the sources to a number of Linux applications, one can say the free codebases are typically cleaner than their Windows counterparts, which makes them easier to secure. The default server-oriented Debian 4.0 Linux distribution requires a mere 170MB of disk space, whereas Windows Server 2003 requires 3 GB. All other things being equal, the fact that Debian is 17 times smaller means it will be more secure. The free database MySQL is a mere 26MB download; the code is clean and small, and therefore much more likely to be reliable and secure.

Another advantage of Linux is that all the applications in a Linux operating system receive security updates. In the Microsoft world, only Microsoft's code is protected by Windows Update.

While Linux is free to acquire, it can also be cheaper to run and maintain than Windows because of its better security. The city of Manchester in England [spent](#) \$2 million in 2009 to remove the Conficker worm from their computers.

## 6. Linux has learned from Windows

While the Windows NT kernel was state of the art at the time it was released in 1993, most of its good ideas have been learned well and absorbed, in spite of the fact that the code has never been released.

For example, the Linux kernel supports asynchronous I/O (input/output), an innovative way to do reads and writes without

tying up “thread” resources. This was an innovation first made widespread in Windows NT.

The ability to load code dynamically is another important feature the Linux kernel adopted from NT and others. Plug and play and suspend and hibernate was a collaboration between Microsoft and hardware companies in the 1990s, and Linux now supports this feature.

Throughout the free software stack, developers have incorporated good ideas from the outside world. There is no Not Invented Here syndrome in free software; a good idea is a good idea, and existing code is even better. In software today, the biggest impediment to sharing ideas is not ego, but license agreements.

The Linux kernel has even learned from Microsoft's mistakes. For example, one feature added to the Windows NT 4.0 kernel was to put the code that draws widgets into the kernel itself. While this can improve graphics performance, it also means that a bug in the code of a button has the capacity to crash the entire system. The best way to keep a system secure and reliable is to keep as much code as possible in user mode above the kernel, and Linux follows this strategy.<sup>7</sup>

## The Feature Race

One of Microsoft's best arguments against free software over the years has been to tout their new features and use that as “proof” that free software will always be behind proprietary software. The moment a Microsoft product comes out, it always offers features that no one else has. However, most of the features are in fact ones that others do have, and Microsoft is catching up. Nearly every new feature is an evolutionary step forward in a place that Microsoft felt needed work.

But like Microsoft, each team in the free software community is improving their code every day. In fact, because Microsoft takes so long to ship, the free software community has often added features before Microsoft. The website <http://kernelnewbies.org> displays the latest list of the Linux features added since the previous release 3-4 months before, and it is typically 15 pages long! For example, here is just the list of driver features added to the 2.6.26 version of the Linux kernel, which had a 3-month dev cycle.

---

<sup>7</sup> In Windows Vista, Microsoft moved *some* of the device drivers to user mode but they should have kept the device drivers small, simple, and in the kernel and instead moved the widgets and fluff to user mode.

## Linus 2.6.26 driver workitems

### 4.1. IDE/SATA

#### IDE

Add warm-plug support for IDE devices  
 Mark "idebus=" kernel parameter as obsolete (take 2)  
 Remove ide=reverse IDE core  
 Add "vlib|pci\_clock=" parameter  
 Add "noacpi" / "acpigt" / "acpionboot" parameters  
 Add "cdrom=" and "chs=" parameters  
 Add "nodma|noflush|noprobe|nowerr=" parameters  
 Add Intel SCH PATA driver  
 Add ide-4drives host driver (take 3)  
 gayle: add "doubler" parameter  
 Remove the broken ETRAX\_IDE driver

#### SATA

sata\_inic162x: add cardbus support  
 libata: prefer hardreset  
 ata: SWNCQ should be enabled by default  
 Make SFF support optional  
 libata: make PMP support optional  
 sata\_mv: disable hotplug for now, enable NCQ on SOC, add basic port multiplier support  
 sata\_fsl: Fix broken driver, add port multiplier (PMP) support

### 4.2. Networking

ssb: add a new Gigabit Ethernet driver to the ssb core  
 Add new qeth device driver,  
 Add new ctm driver that reemplaces the old ctc one,  
 New driver "sfc" for Solarstorm SFC4000 controller.  
 Driver for IXP4xx built-in Ethernet ports  
 Add support the Korina (IDT RC32434) Ethernet MAC  
 iwliwif: Support the HT (802.11n) improvements,,... add default WEP key host command, add 1X HW WEP support, add default WEP HW encryption, use HW acceleration decryption by default, hook iwliwif with Linux rfkill, add TX/RX statistics to driver, add debugs to iwlc core, enables HW TKIP encryption, add led support, enables RX TKIP decryption in HW, remove IWLC{4965,3945}. QOS  
 ath5k: Add RF2413 srev values, add RF2413 initial settings, identify RF2413 and deal with PHY\_SPENDING, more RF2413 stuff, port to new bitrate/channel API, use software encryption for now  
 pasci\_mac: jumbo frame support, enable GSO by default, basic ethtool support, netpoll support  
 rt2x00: Add per-interface structure, enable master and adhoc mode again, enable LED class support for rt2500usb/rt73usb  
 e1000e: Add interrupt moderation run-time ethtool interface, add support for BM PHYs on ICH9  
 niu: Add support for Neptune FEM/NEM cards for C10 server blades, Add Support for Sun ATCA Blade Server.  
 gianfar: Support NAPI for TX Frames  
 ehea: Add DLPAR memory remove support  
 sfc: Add TSO support  
 b43: Add QOS support, add HostFlags HI support, use SSB block-I/O to do PIO  
 S2io: Multiqueue network device support implementation,, enable multi ring support, add napi support when MSIX is enabled.  
 ixgbe: Introduce MSI-X queue vector code, introduce Multiqueue TX, add optional DCA infrastructure, introduce adaptive interrupt moderation  
 uli526x: add support for netpoll  
 fmvj18x\_cs: add NextCom NC5310 rev B support  
 zd1211rw: support for mesh interface and beaconing  
 libertas: implement SSID scanning for SIOCSIWSCAN  
 ethtool: Add support for large eeproms  
 The scheduled bcm43xx removal  
 The scheduled ieee80211 softmac removal  
 The scheduled rc80211-simple.c removal  
 Remove obsolete driver sk98lin  
 Remove the obsolete xircom\_tulip\_cb driver

### 4.3. Graphics

radeon: Initial r500 support,,  
 intel\_agp: Add support for Intel 4 series chipsets  
 i915: Add support for Intel series 4 chipsets  
 Add support for Radeon Mobility 9000 chipset  
 fb: add support for foreign endianness  
 pxa6f: preliminary smart panel interface support,

### 4.6. Video

cx88: Add support for the Dvico PCI Nano, add xc2028/3028 boards, add support for tuner-xc3028  
 saa7134: add support for the MSI TV@nywhere A/D v1.1 card, add support for the Creatix CTX953 V1.4.3 Hybrid  
 saa717x: add new audio/video decoder i2c driver  
 Support DVB-T tuning on the DViCO FusionHDTV DVB-T Pro  
 Add support for xc3028-based boards  
 itvt: add support for Japanese variant of the Adaptec AVC-2410  
 Add basic support for Prolink Pixelview MPEG 8000GT  
 btvt: added support for Kozumi KTIV-01C card  
 Add support for Kworld ATSC 120  
 CX24123: preparing support for CX24113 tuner  
 Added support for Terratec Cinergy T USB XXS  
 budget: Add support for Fujitsu Siemens DVB-T Activy Budget  
 Support for DVB-S demod PNI1010 (clone of S5H1420) added  
 Added support for SkyStar2 rev2.7 and ITD1000 DVB-S tuner  
 em28xx-dvb: Add support for HVR950, add support for the HVR-900  
 Add support for Hauppauge HVR950Q/HVR850/FusioHDTV7-USB  
 HVR950Q Hauppauge eeprom support  
 Adding support for the NXP TDA10048HN DVB OFDM demodulator  
 Add support for the Hauppauge HVR-1200  
 pvrusb2-dvb: add DVB-T support for Hauppauge pvrusb2 model 73xxx  
 Add support for Beholder BeholdTV H6  
 cx18: new driver for the Conexant CX23418 MPEG encoder chip  
 s5h1411: Adding support for this ATSC/QAM demodulator

### 4.7. SCSI

zfcp: Add trace records for recovery thread and its queues, add traces for state changes,, trace all triggers of error recovery activity,register new recovery trace,, remove obsolete erp\_dbf trace, add trace records for recovery actions.  
 gla2xxx: Add support for host supported speeds FC transport attribute,, add FC-transport Asynchronous Event Notification support,, add hardware trace-logging support,, add Flash Descriptor Table layout support,, add ISP84XX support,, add midlayer target/device reset support.  
 iscsi: extended cdb support, bidi support at the generic libiscsi level, bidi support for iscsi\_tcp  
 scsi\_debug: support large non-fake virtual disk  
 gdth: convert to PCI hotplug API  
 st: add option to use SILI in variable block reads  
 megaraidd\_sas: Add the new controller(1078DE) support to the driver  
 m68k: new mac\_esp scsi driver  
 bsg: add large command support  
 Add support for variable length extended commands  
 aacraid: Add Power Management support  
 dpt\_i2o: 64 bit support, sysfs  
 Firmware: add iSCSI iBFT Support

### 4.8. WATCHDOG

Add a watchdog driver based on the CS5535/CS5536 MFGPT timers  
 Add ICH9DO into the iTCO\_wdt.c driver

### 4.9. HWMON

thermal: add hwmon sysfs I/F  
 ibmaem: new driver for power/energy/temp meters in IBM System X hardware  
 i5k\_amb: support Intel 5400 chipset

### 4.10. USB

ISP1760 HCD driver  
 pxa27x\_udc driver  
 CDC WDM driver  
 Add Cypress c67x00 OTG controller core driver,,  
 Add HP hs2300 Broadband Wireless Module to sierra.c  
 Partial USB embedded host support  
 Add usb-serial scpc8x5 driver  
 r8a66597-hcd: Add support for SH7366 USB host  
 Add Zoom Telephonics Model 3095F V.92 USB Mini External modem to cdc-acm  
 Support for the ET502HS HSDPA modem  
 atmel\_usb\_a\_udc: Add support for AT91CAP9 UDPHS



|  |   |
|--|---|
| <p>Driver for Freescale 8610 and 5121 DIU</p> <p>intelfb: add support for the Intel Integrated Graphics Controller 965G/965GM</p> <p>Add support for Blackfin/Linux logo for framebuffer console</p> <p><b>4.4. Sound</b></p> <p>hda-codec - Allow multiple SPDIF devices, add SI HDMI codec support, add support for the OOO Model 2, add support of Zepto laptops, support RV7xx HDMI Audio, add model=mobile for AD1884A &amp; co, add support of AD1883/1884A/1984A/1984B, add model for cx20549 to support laptop HP530, add model for alc883 to support FUJITSU P12515, add support for Toshiba Equium L30, Map 3stack-6ch-dig ALC662 model for Asus P5GC-MX, support of Lenovo Thinkpad X300, add Quanta IL1 ALC267 model, add support of AD1989A/AD1989B, add model for alc262 to support Lenovo 3000, add model for ASUS P5K-E/WIFI-AP, added support for Foxconn P35AX-S mainboard, add drivers for the Texas Instruments OMAP processors, add support of Medion RIM 2150, support IDT 92HD206 codec</p> <p>ice1724 - Enable AK4114 support for Audiophile192</p> <p>ice1712: Added support for Delta1010E (newer revisions of Delta1010), added support for M-Audio Delta 66E, add Terrasonix TS88 support</p> <p>Davinci ASoC support</p> <p>intel8x0 - Add support of 8 channel sound</p> <p>ASoC: WM9713 driver</p> <p>Emagic Audiowerk 2 ALSA driver.</p> <p>Add PC-speaker sound driver</p> <p>oxygen: add monitor controls</p> <p>virtuoso: add Xonar DX support</p> <p>soc - Support PXA3xx AC97</p> <p>pxa2xx-ac97: Support PXA3xx AC97</p> <p><b>4.5. Input</b></p> <p>Add support for WM97xx family touchscreens</p> <p>WM97xx - add chip driver for WM9705 touchscreen, add chip driver for WM9712 touchscreen, add chip driver for WM97123 touchscreen, add support for streaming mode on Mainstone wacom: add support for Cintiq 20WSX</p> <p>xpad: add support for wireless xbox360 controllers</p> <p>Add PS/2 serio driver for AVR32 devices</p> <p>aiptek: add support for Genius G-PEN 560 tablet</p> <p>Add Zhen Hua driver</p> <p>HID: force feedback driver for Logitech Rumblepad 2, Logitech diNovo Mini pad support</p> <p><b>4.6. Video</b></p> <p>V4L2 soc_camera driver for PXA270,,</p> <p>Add support for the MT9M001 camera</p> <p>Add support for the MT9V022 camera</p> <p>Add support for the ISL6405 dual LNB supply chip</p> <p>Initial DVB-S support for MD8800 /CTX948</p> <p>cx23885: Add support for the Hauppauge HVR1400, add generic cx23417 hardware encoder support</p> <p>Add mxl5505s driver for MaxiLinear 5505 chipsets, basic digital support.</p> | <p><b>4.11. FireWire</b></p> <p><b>release notes at linux1394-user</b></p> <p><b>4.12. Infiniband</b></p> <p>IPoIB: Use checksum offload support if available, add LSO support, add basic ethtool support, support modifying IPoIB CQ event moderation, handle 4K IB MTU for UD (datagram) mode</p> <p>ipath: Enable 4KB MTU, add code to support multiple link speeds and widths, EEPROM support for 7220 devices, robustness improvements, cleanup, add support for IBTA 1.2 Heartbeat</p> <p>Add support for IBA7220,,,,,,,,</p> <p>mthca: Add checksum offload support</p> <p>mlx4: Add checksum offload support, add IPoIB LSO support to mlx4,</p> <p>RDMA/cxgb3: Support peer-2-peer connection setup</p> <p><b>4.13. ACPI and Power Management</b></p> <p>ACPICA: Disassembler support for new ACPI tables</p> <p>eeepc-laptop: add base driver, add backlight, add hwmon fan control</p> <p>thinkpad-acpi: add sysfs led class support for thinklight (v3.1), add sysfs led class support to thinkpad leds (v3.2)</p> <p>Remove legacy PM</p> <p><b>4.14. MTD</b></p> <p>m25p80: add FAST_READ access support to M25Pxx, add Support for ATMEL AT25DF641 64-Megabit SPI Flash</p> <p>JEDEC: add support for the ST M29W400DB flash chip</p> <p>NAND: support for pxa3xx</p> <p>NOR: Add JEDEC support for the SST 36VF3203 flash chip</p> <p>NAND: FSL UPM NAND driver</p> <p>AR7 mtd partition map</p> <p>NAND: S3C2410 Large page NAND support</p> <p>NAND: Hardware ECC controller on at91sam9263 / at91sam9260</p> <p><b>4.15. I2C</b></p> <p>Add support for device alias names</p> <p>Convert most new-style drivers to use module aliasing</p> <p>Renesas SH7760 I2C master driver</p> <p>New driver for the SuperH Mobile I2C bus controller</p> <p>Convert remaining new-style drivers to use module aliasing</p> <p><b>4.16. Various</b></p> <p>MMC: OMAP: Add back cover switch support</p> <p>MMC: OMAP: Introduce new multislot structure and change driver to use it</p> <p>mmc: mmc host test driver</p> <p>4981/1: [KS8695] Simple LED driver</p> <p>leds: Add mail LED support for "Clevo D400P"</p> <p>leds: Add support to leds with readable status</p> <p>leds: Add new driver for the LEDs on the Freecom FSG-3</p> <p>RAPIDIO:</p> <p>Add RapidIO multi mport support</p> <p>Add OF-tree support to RapidIO controller driver</p> <p>Add serial RapidIO controller support, which includes MPC8548, MPC8641</p> <p>edac: new support for Intel 3100 chipset</p> <p>Basic braille screen reader support</p> <p>ntp: support for TAI</p> <p>RTC: Ramtron FM3130 RTC support</p> |
|--|---|

*Don't worry if you don't understand what these things mean as I don't either. It is just important to understand that even the hardware of computers are too big and complicated for one company to oversee the development of.*

This is just a portion of the code changes for that version of Linux, and doesn't include work in the file systems, networking, performance, architecture-specific work, and so forth.

Free software has added countless features before Microsoft. However, many of these features are so obscure that your eyes would glaze over reading the list. Anyone who focuses on the highly-touted features in Microsoft's latest release is missing the larger

picture: even when these features are compelling, they often make you more deeply tied into the proprietary world and make it harder to migrate away.

Microsoft's new software always has new features, but usually only one piece of code for a scenario. Linux supports 60 file systems, including several clustering solutions, that allow a seamless scale-out of disk storage amongst a group of servers. Microsoft supports just one, which is tied very tightly to its system, which in turn limits its flexibility. Free software also approaches solutions from different directions. There are several ways to attack virtualization and Linux is working on them in parallel and finding what is common between them.<sup>8</sup>

## Linux is Inexorably Winning

The fact that the Linux kernel has these many advantages over Windows means two things. First, the argument that free software engineers are unable to innovate and only copy the work of others is not true.

The head of Windows Server recently [said](#) that free software, “by its very nature, does not allow intellectual property to be built.” This statement is factually incorrect: the Linux kernel is doing many things no kernel has ever done before, just as Wikipedia is doing things no encyclopedia has ever done before.

Second, even if Microsoft gave away the source code to the Windows kernel, the existing free software community would dismiss it. A worldwide community has pounded Linux into the shape it should be. Linus calls Windows “witchcraft”, so why work on an inferior codebase whose details up till now have been opaque?

---

<sup>8</sup> Two of the biggest differences in strategy is User Mode Linux (UML) which changes Linux to run as an application on top of another instance of Linux, and the standard virtualization, which runs the guest OS in kernel mode, though it doesn't actually talk to the hardware. The Linux kernel is evolving towards figuring out the architecture, and what is shared between the different strategies.

## Charging for an OS

A Linux operating system is an entirely different beast compared to a Microsoft operating system. Microsoft was constantly torn about how much value to invest in Windows, and how much to set aside for extra licensing revenue in other products. Windows Vista has five different versions, (originally they announced eight!), each with basically the same code, but with dramatically different prices:

| <b>Product<br/>(Amazon.com)</b> | <b>Upgrade</b>        | <b>New</b> |
|---------------------------------|-----------------------|------------|
| Windows Vista Ultimate          | \$243                 | \$350      |
| Windows Vista Business          | \$176                 | \$260      |
| Windows Vista Home Premium      | \$140                 | \$219      |
| Windows Vista Home Basic        | \$85                  | \$157      |
| Windows Vista Enterprise        | Custom volume license |            |

*Microsoft charges \$85 to \$350 for Windows Vista, but the code in each version is 99% the same.*

Microsoft's natural goal is to put as many new and compelling features into high-end, high-profit versions even though the prices are not correlated to the work required to build the various Vista versions.

Creating multiple versions is tricky because if third-party applications depend on features that aren't in a particular version of Windows, then the applications won't run, and the Windows brand is weakened. Therefore, Microsoft would sometimes add logic to cripple the performance of high-end features on low-end versions. In the free software world, no one deliberately cripples their own creation.

When I worked at Microsoft, there were numerous turf wars. For example, the Word team fought with the Windows team over whether WordPad, a tiny Windows applet, should have the ability to read Word's DOC files. The Windows team wanted to create an operating system that had the ability to display the hundreds of millions of DOC files, but the Word team didn't want to create a reason for people not to purchase Word. There were also running battles between the teams responsible for Outlook and Outlook Express, Exchange and SQL, Works and Word, FoxPro and Access, Access and VB, SQL Server and Access, PC and XBox. Each team was paranoid about another team adding functionality that would discourage someone from buying *their* product.

Microsoft also was torn between bundling important features in the operating system, like they did with web browsers, instant messaging and multimedia, and leaving features out in order to reap more money later. Windows doesn't ship with a dictionary because Office has one. Windows doesn't come with development tools because those are part of Microsoft's Visual Studio business. With Linux, anything free is welcomed into the operating system.

In addition to Microsoft's strategic decisions to exclude certain features, there are also cases of benign neglect. The Sound Recorder in Windows XP lets you only record for 1 minute, a limitation that exists from the days of 16-bit Windows that no one bothered to fix. Microsoft's official solution is for the customer to purchase Office OneNote.

Applets, command-line tools, and many other important but unsexy parts of an operating system were always allocated very limited resources. Ask Steve Ballmer for resources for the unimportant Sound Recorder, and you would receive a dirty, polystyrene fork hurtling in your direction. In the free software model, anyone, on their time frame, can improve a piece of code that is useful or interesting to them, whether “strategic” or not. Eric Raymond calls this phenomenon a developer “scratching their own itch.” Wikipedia is built almost entirely from this mechanism, and only free software can capture every little advancement.

## Complexity of License Agreements

It would be possible to fund the construction of all roads with tolls. This would entail having toll booths at all street corners. Such a system would provide a great incentive to improve roads. It would also have the virtue of causing the users of any given road to pay for that road. However, a toll booth is an artificial obstruction to smooth driving—artificial, because it is not a consequence of how roads or cars work.

—Richard Stallman

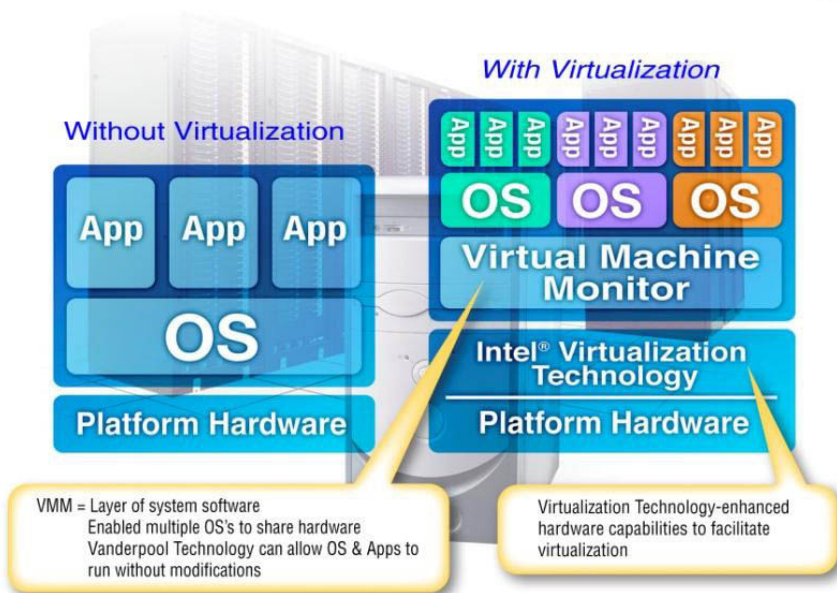
Even if you believe it is perfectly okay to charge for software, it is hard to know *what* to charge consumers for software. Microsoft used to charge a fixed amount for a server product. However, one day they realized that customers who had 10 users connected to a server should be paying less than those who had 100. This observation resulted in the creation of client access licenses (CALs); a concept that required payment for each individual user of server software, but which is an additional burden on the customer. If you have 1,000 employees accessing 1,000 servers, you need to fill out a lot of paperwork.

This was all before the invention of the Internet whereby the number of users of a server could easily be in the thousands and which made many usages of CALs expensive and unsustainable. Therefore, Microsoft moved towards a model where the cost was based on the number of processors in the computer so that little boxes would cost less than big boxes.

This model worked until Intel introduced the concept of “hyper-threading”, which fools a computer into thinking there are two processors inside the computer, but which adds only 15-30% more performance. Microsoft's customers would of course be unhappy at the thought of purchasing a bunch of new licenses for such a small performance improvement, so Microsoft ended up giving free licenses for hyperthreaded processors.

Then, virtualization was created:

## Intel® Virtualization Technology



*Intel virtualization marketing diagram: Virtualization allows the ability to run multiple operating systems on one computer: each instance thinks it controls the machine. The best uses for virtualization are web hosting, inside corporate data centers, and for software developers.*

Virtualization allows different applications to be fully isolated from each other inside different complete instances of an operating system, but at the same time share the CPU and other hardware resources. Isolation is important because even within a single com-

pany's data center, different departments don't want to run their code on the same machine. If Hotmail went down, they didn't want it to be the fault of the Microsoft Bob web page.

I was told that inside the cavernous datacenters of a Fortune 500 company, the computers used on average only 15% of their CPU's capacity. The different departments all maintained their own hardware, which they'd built out to handle the maximum possible load; this is an idea as silly as utilizing only 15% of an office building.

Virtualization gives you software isolation but allows you to share hardware. However, when you install more proprietary software, licensing issues arise. If I put three copies of a database server in separate virtualization instances on a four-processor computer, under many of Microsoft's licensing models I would have to purchase enough licenses for 12 processors — even though the computer only has four. Like hyperthreading, virtualization is another technology that wasn't conceived of when Microsoft created their per-processor licensing model.

In a free software environment, you can add new hardware and add or remove applications without paying or keeping track of anything. I will write more about economic aspects later, but for now it is only important to understand that free software sidesteps these hassles.

## Free Software Only Costs PCs

Creating a unified industrial software base built around free software will not only make PCs more powerful, but it will also allow us to push their intelligence everywhere, from cars to medical equipment. Free software will be built because it is so valuable to businesses; everyone else will come along for the ride and get it for free.

The proprietary software model has hurt many of the smaller software markets. In fact, except for gaming business which is as much about artistry as software, Microsoft is nearly the last proprietary software company standing. When I was at Microsoft, our biggest competitors were companies such as Borland, WordPerfect, Corel, Lotus, Netscape and Sybase — names you never hear anymore.

There is no Microsoft of educational software selling products used in every school. There are several companies selling proprietary products, perhaps receiving enough revenue to stay in business, but the proprietary software model has prevented them from achieving critical mass.

Recently, I saw an ad for a trivial software applet capable of converting a DVD to the iPod video format. Microsoft managed to convince everybody that each little software product is worth selling. Now you see why Stallman's analogy of proprietary software as toll-booths is apt — they serve as permanent, pervasive obstacles holding up progress for software that was written years earlier. Free software flows with much less friction. In fact, in the free software world, the definition of a PC operating system completely changes.

## A Free Operating System

Our most potent operating system competitor is Linux and the phenomena around open source and free software. The same phenomena fuels competitors to all of our products. The ease of picking up Linux to learn it or to modify some piece of it is very attractive. The academic community, start-up companies, foreign governments and many other constituencies are putting their best work into Linux.

—Bill Gates

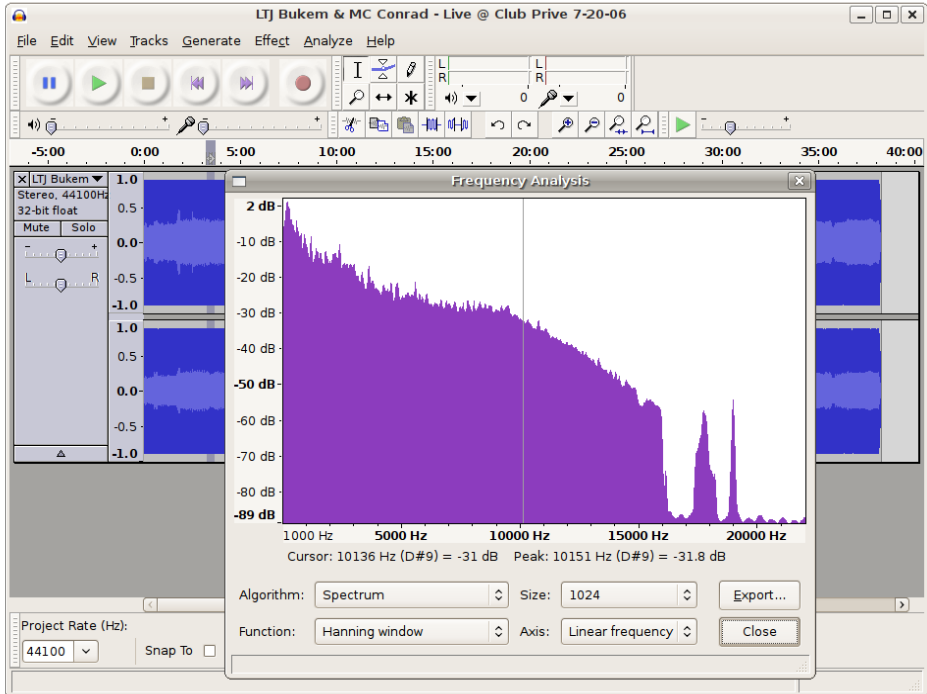


*The One Laptop Per Child has as much CPU power as a workstation of 15 years ago, but would be just a shiny box without free software and content.*

The biggest difference between Windows and Linux is that free software contains thousands of applications, installable with one click, and managed as one set. A Linux operating system includes all the obvious stuff like a spreadsheet, web browser, and instant mes-

saging, but it also includes tools for making pictures and music, children's applications, server software, the Bible, development tools, and much more.

Audacity is the most popular free audio editor on Linux. It doesn't have a talking paper clip: "It looks like you're trying to add echo. Would you like some help?" But it does provide a well-rounded set of features, and has many effects for the manipulation of sound.



*A rich and reliable free audio manipulation tool*

Audacity, of course, doesn't have a one-minute limitation on its ability to record as Windows XP does. Its most important feature is its plugins for import, export, and effects — extensibility is one of free software's best features.

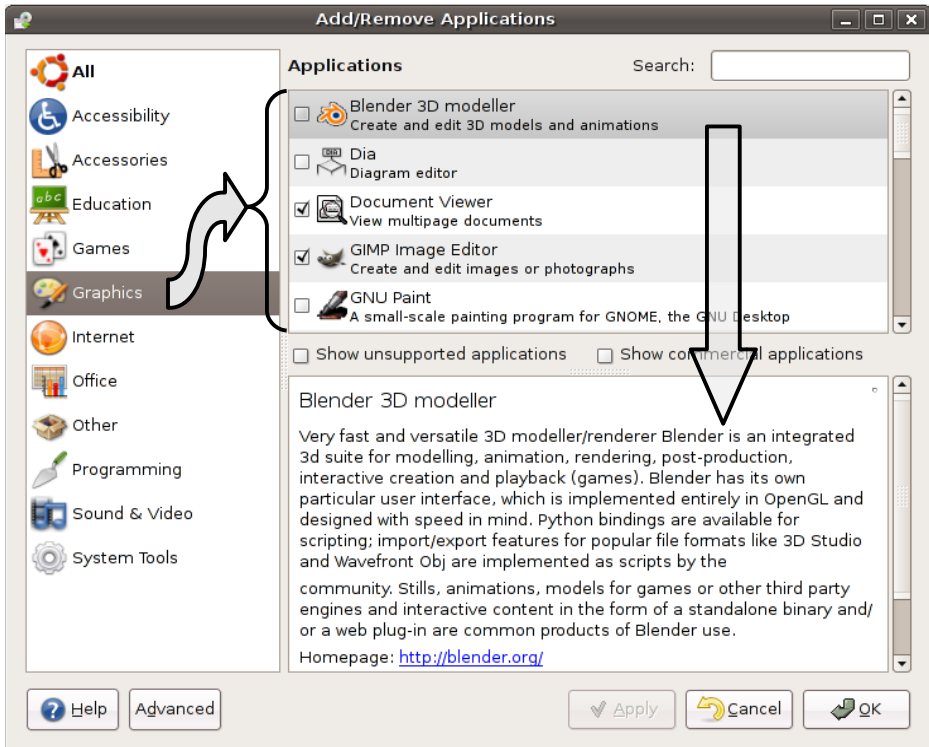


| Audacity Audio Editor Feature List  |  |
|---|--|
| <p><b>Recording</b></p> <ul style="list-style-type: none"> <li>• Audacity can record live audio through a microphone or mixer, or digitize recordings from cassette tapes, vinyl records, or minidisks. With some sound cards, it can also capture streaming audio.</li> <li>• Record from microphone, line input, or other sources.</li> <li>• Dub over existing tracks to create multi-track recordings.</li> <li>• Record up to 16 channels at once (requires multi-channel hardware).</li> <li>• Level meters can monitor volume levels before, during, and after recording.</li> </ul> <p><b>Import and Export</b></p> <ul style="list-style-type: none"> <li>• Import sound files, edit them, and combine them with other files or new recordings. Export your recordings in several common file formats.</li> <li>• Import and export WAV, AIFF, AU, and <a href="#">Ogg Vorbis</a> files.</li> <li>• Import MPEG audio (including MP2 and MP3 files) with <a href="#">libmad</a>.</li> <li>• Export MP3s with the optional LAME encoder library.</li> <li>• Create WAV or AIFF files suitable for burning to CD.</li> <li>• Open raw (headerless) audio files using the “Import Raw” command.</li> <li>• <b>Note:</b> Audacity does not currently support WMA, AAC, or most other proprietary or restricted file formats.</li> </ul> <p><b>Editing</b></p> <ul style="list-style-type: none"> <li>• Easy editing with Cut, Copy, Paste, and Delete.</li> <li>• Use unlimited Undo (and Redo) to go back any number of steps.</li> <li>• Very fast editing of large files.</li> <li>• Edit and mix an unlimited number of tracks.</li> <li>• Use the Drawing tool to alter individual sample points.</li> <li>• Fade the volume up or down smoothly with the Envelope tool.</li> </ul> | <p><b>Effects</b></p> <ul style="list-style-type: none"> <li>• Change the pitch without altering the tempo, or vice-versa.</li> <li>• Remove static, hiss, hum, or other constant background noises.</li> <li>• Alter frequencies with Equalization, FFT Filter, and Bass Boost effects.</li> <li>• Adjust volumes with Compressor, Amplify, and Normalize effects.</li> <li>• Other built-in effects include:</li> <li>• Echo</li> <li>• Phaser</li> <li>• Wahwah</li> <li>• Reverse</li> </ul> <p><b>Sound Quality</b></p> <ul style="list-style-type: none"> <li>• Record and edit 16-bit, 24-bit, and 32-bit (floating point) samples.</li> <li>• Record at up to 96 KHz.</li> <li>• Sample rates and formats are converted using high-quality resampling and dithering.</li> <li>• Mix tracks with different sample rates or formats, and Audacity will convert them automatically in real time.</li> </ul> <p><b>Plug-Ins</b></p> <ul style="list-style-type: none"> <li>• Add new effects with <a href="#">LADSPA plug-ins</a>.</li> <li>• Audacity includes some sample plug-ins by <a href="#">Steve Harris</a>.</li> <li>• Load VST plugins for Windows and Mac, with the optional <a href="#">VST Enabler</a>.</li> <li>• Write new effects with the built-in <a href="#">Nyquist</a> programming language.</li> </ul> <p><b>Analysis</b></p> <ul style="list-style-type: none"> <li>• Spectrogram mode for visualizing frequencies.</li> <li>• “Plot Spectrum” command for detailed frequency analysis.</li> </ul> <p><b>Free and Cross-Platform</b></p> <ul style="list-style-type: none"> <li>• Licensed under the <a href="#">GNU General Public License (GPL)</a>.</li> <li>• Runs on Mac OS X, Windows, and GNU/Linux.</li> </ul> |

Developers of free software applications tend to build extensibility plugins as a fundamental way of writing their software because they know their tool will never by itself be able to do all the things people will want. A plugin provides a boundary between things that

manage data, and things that manipulate it. The most popular plug-ins eventually become a part of the base system, but by being built separately, they have forced clean boundaries and modularity.<sup>9</sup>

Every application that Linux has that Windows doesn't is a feature Windows is missing:



*Richard Stallman's free software vision realized: A free Linux operating system contains an entire store of free applications available with one click, and built to work together. Having so many tools at your disposal makes computers more personal, powerful, productive, and enjoyable. Your computing experience becomes limited only by your creativity.*

A free operating system is where Metcalfe's law meets software: the more people using free software, the more applications will be built for it. Just one piece of free software isn't useful, but with an entire stack, we can enter a shining age.

<sup>9</sup> I argue in another place in the book that software has no clear boundaries. What I meant was that one never really knows precisely what the interface between manager and manipulator should be. For audio files, the boundary seems clear: here is some audio data, chew on it. However even there you must ask: what DSP APIs are available to the plugins? Otherwise, each plugin will need lots of duplicate code that the manager already likely has! It is the new hardware capabilities that create a need for a change at this boundary. The lesson here is to keep your boundaries simple, but assume you may need to change them.

Going from today's 20 million Linux users to the anticipated one billion means the potential for 50 times more resources. The free software stack has several challenges I will discuss throughout this book, but it is important to mention here that few of the applications in the dialog box are as polished or reliable as Firefox. However, many are very powerful, and more than good enough to depend on in a business.

While Linux still needs work, Windows is no day at the beach. Here is an email from Bill Gates' describing his experience of installing Microsoft MovieMaker on Windows:

**From:** Bill Gates  
**Sent:** Wednesday, January 15, 2003 10:05 AM  
**To:** Jim Allchin  
**Cc:** Chris Jones (WINDOWS); Bharat Shah (NT); Joe Peterson; Will Poole; Brian Valentine; Anoop Gupta (RESEARCH)  
**Subject:** Windows Usability degradation flame

I am quite disappointed at how Windows Usability has been going backwards and the program management groups don't drive usability issues.

Let me give you my experience from yesterday.

I decided to download (Moviemaker) and buy the Digital Plus pack ... so I went to Microsoft.com. They have a download place so I went there.

The first 5 times I used the site it timed out while trying to bring up the download page. Then after an 8 second delay I got it to come up.

This site is so slow it is unusable.

It wasn't in the top 5 so I expanded the other 45.

These 45 names are totally confusing. These names make stuff like: C:\Documents and Settings\billg\My Documents\My Pictures seem clear.

They are not filtered by the system ... and so many of the things are strange.

I tried scoping to Media stuff. Still no moviemaker. I typed in movie. Nothing. I typed in movie maker. Nothing.

So I gave up and sent mail to Amir saying - where is this Moviemaker download? Does it exist?

So they told me that using the download page to download something was not something they anticipated.

They told me to go to the main page search button and type movie maker (not moviemaker!)

I tried that. The site was pathetically slow but after 6 seconds of waiting up it came.

I thought for sure now I would see a button to just go do the download.

In fact it is more like a puzzle that you get to solve. It told me to go to Windows Update and do a bunch of incantations.

This struck me as completely odd. Why should I have to go somewhere else and do a scan to download moviemaker?

So I went to Windows update. Windows Update decides I need to download a bunch of controls. (Not) just once but multiple times where I get to see weird dialog boxes.

Doesn't Windows update know some key to talk to Windows?

Then I did the scan. This took quite some time and I was told it was critical for me to download 17megs of stuff.

This is after I was told we were doing delta patches to things but instead just to get 6 things that are labeled in the SCARIEST possible way I had to download 17meg.

So I did the download. That part was fast. Then it wanted to do an install. This took 6 minutes and the machine was so slow I couldn't use it for anything else during this time.

What the heck is going on during those 6 minutes? That is crazy. This is after the download was finished.

Then it told me to reboot my machine. Why should I do that? I reboot every night — why should I reboot at that time?

So I did the reboot because it INSISTED on it. Of course that meant completely getting rid of all my Outlook state.

So I got back up and running and went to Windows Update again. I forgot why I was in Windows Update at all since all I wanted was to get Moviemaker.

So I went back to Microsoft.com and looked at the instructions. I have to click on a folder called WindowsXP. Why should I do that? Windows Update knows I am on Windows XP.

What does it mean to have to click on that folder? So I get a bunch of confusing stuff but sure enough one of them is Moviemaker.

So I do the download. The download is fast but the Install takes many minutes. Amazing how slow this thing is.

At some point I get told I need to go get Windows Media Series 9 to download.

So I decide I will go do that. This time I get dialogs saying things like "Open" or "Save". No guidance in the instructions which to do. I have no clue which to do.

The download is fast and the install takes 7 minutes for this thing.

So now I think I am going to have Moviemaker. I go to my add/remove programs place to make sure it is there.

It is not there.

What is there? The following garbage is there. Microsoft Autoupdate Exclusive test package, Microsoft Autoupdate Reboot test package, Microsoft Autoupdate testpackage1. Microsoft AUtoupdate testpackage2, Microsoft Autoupdate Test package3.

Someone decided to trash the one part of Windows that was usable? The file system is no longer usable. The registry is not usable. This program listing was one sane place but now it is all crapped up.

But that is just the start of the crap. Later I have listed things like Windows XP Hotfix see Q329048 for more information. What is Q329048? Why are these series of patches listed here? Some of the patches just things like Q810655 instead of saying see Q329048 for more information.

What an absolute mess.

Moviemaker is just not there at all.

So I give up on Moviemaker and decide to download the Digital Plus Package.

I get told I need to go enter a bunch of information about myself.

I enter it all in and because it decides I have mistyped something I have to try again. Of course it has cleared out most of what I typed.

I try (typing) the right stuff in 5 times and it just keeps clearing things out for me to type them in again.

So after more than an hour of craziness and making my programs list garbage and being scared and seeing that Microsoft.com is a terrible website I haven't run Moviemaker and I haven't got the plus package.

The lack of attention to usability represented by these experiences blows my mind. I thought we had reached a low with Windows Network places or the messages I get when I try to use 802.11. (don't you just love that root certificate message?)

When I really get to use the stuff I am sure I will have more feedback.

## Linux Distributions

With Linux, each OS distribution carves out a niche to meet its users' needs. There are specialized versions of Linux containing educational software, tools for musicians, versions dedicated to embedded or low-end hardware, and regional versions of Linux produced in places like Spain and China.

The various distributions have much in common, including the Linux kernel, but use different free software and installation mechanisms. One distribution called Gentoo downloads only one binary, a

bootstrapping compiler. The rest of its deliverables are the source code to the components they offer. This gives the user the ability to build a system highly optimized for his hardware.

Some distributions are optimized to run well on old hardware and fit on CDs the size of a credit card:



*Damn Small Linux is the most popular Linux for old computers and ships on 80x60 mm CDs.*

Linux is very popular on servers, which require an additional focus on performance, reliability, and security. One of the simplest ways to decrease bloat and security risks is to remove the graphical interface:

```
top - 12:54:17 up 62 days, 20:14, 2 users,  load average: 0.16, 0.42, 0.43
Tasks: 127 total,  1 running, 126 sleeping,  0 stopped,  0 zombie
Cpu(s):  0.2% us,  0.3% sy,  0.0% ni, 98.5% id,  0.2% wa,  0.7% hi,  0.2% si
Mem:   514248k total,  489360k used,    24888k free,    79128k buffers
Swap: 1020088k total,  18416k used,  1001672k free,  177528k cached
```

| PID  | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+   | COMMAND     |
|------|------|----|----|------|-----|-----|---|------|------|---------|-------------|
| 2041 | root | 10 | -5 | 0    | 0   | 0   | S | 0    | 0.0  | 0:11.74 | usb-storage |
| 1    | root | 16 | 0  | 1564 | 528 | 460 | S | 0    | 0.1  | 0:01.09 | init        |
| 2    | root | RT | 0  | 0    | 0   | 0   | S | 0    | 0.0  | 0:00.01 | migration/0 |
| 3    | root | 34 | 19 | 0    | 0   | 0   | S | 0    | 0.0  | 0:14.63 | ksoftirqd/0 |
| 4    | root | RT | 0  | 0    | 0   | 0   | S | 0    | 0.0  | 0:00.00 | watchdog/0  |

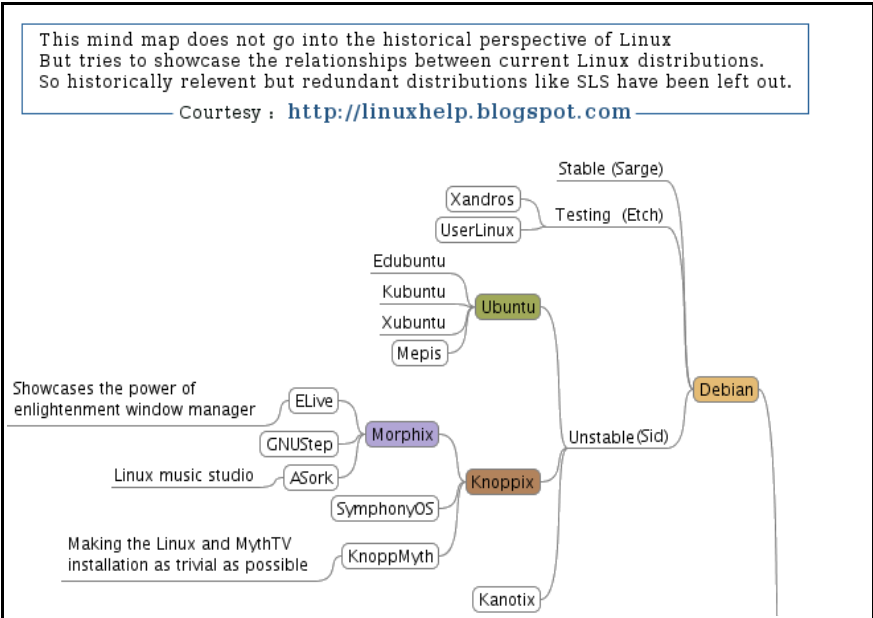
*Screenshot of a text-based Linux process viewer. Just because a computer displays only text doesn't mean there isn't a modern, rock-solid operating system underneath.*

It is as hard to write a graphical user interface (GUI) for a server as it is to write the server itself: every new feature in the server needs a corresponding improvement in the administration interface. If your server can generate 1,000 records per second, then your user interface had better be able to handle that without choking.<sup>10</sup>

While most devices do not need a GUI, desktop users do, and like everything in the free software world today, there are several good choices.

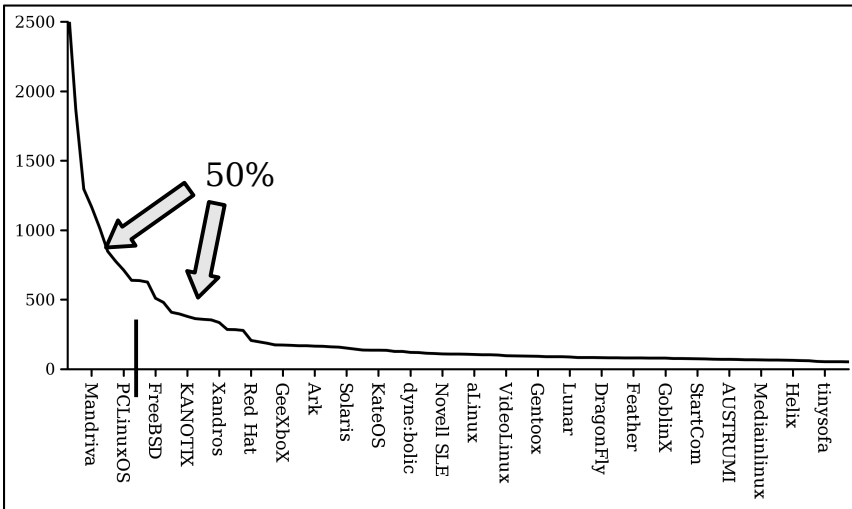
<sup>10</sup> Many embedded scenarios don't include a GUI because there is no standard one yet. Most GUIs are too complicated and slow for embedded scenarios: the brand-new cash register at my local Starbucks has a touch display with a text interface.

The following map shows the Debian branch of the Linux distribution family tree; they derive from each other, just like in a biological ecosystem:



*A portion of the Linux family tree showing the Debian branch, the biggest free software distribution.*

Here is a chart showing the relative popularity of Linux distros:  
**Linux Distro Popularity Curve (Distrowatch.com)**



*Linux distributions, sorted by popularity. The line shows the divide between both halves of the popularity curve.*

What you see here is an almost perfectly smooth curve that illustrates a relatively new idea called the “Long Tail.” One way to think about this idea is to look at the English language. Words like “the” are used with great frequency, but *many more* words like “teabag” are used infrequently. There is a long tail of infrequently used English words, and to just ignore them would be to throw away much of what makes our language so special.

The lesson of the long tail in business is the importance of catering to customers with special interests. The long tail of Linux distributions means that the creation of a free software ecosystem doesn't mean the end of the free market, or of competition.

Wikipedia and the Linux kernel are two of the best examples of the fact that free software and the free exchange of ideas can create superior products without licensing fees. The mere existence of these premier products, without a gigantic company behind them, is proof that the proprietary development model is doomed.



# AI AND GOOGLE

The future is open source everything.

—Linus Torvalds

That knowledge has become *the* resource, rather than *a* resource, is what makes our society post-capitalist.

—Peter Drucker, 1993

**I**magine 1,000 people, broken up into groups of five, working on two hundred separate encyclopedias, versus that same number of people working on one encyclopedia? Which one will be the best? This sounds like a silly analogy when described in the context of an encyclopedia, but it is exactly what is going on in artificial intelligence (AI) research today.<sup>1</sup>

In early drafts of this book, I had positioned this chapter after the one explaining economic and legal issues around free software. However, I now believe it is important to discuss artificial intelligence separately and first, because AI is the holy-grail of computing, and the reason we haven't solved AI is that there are no free software codebases that have gained critical mass. Far more than enough people are out there, but they are usually working in teams of less than five, and very often in proprietary codebases.

## Deep Blue has been Deep-Sixed

Some people worry that artificial intelligence will make us feel inferior, but then, anybody in his right mind should have an inferiority complex every time he looks at a flower.

—Alan Kay, computer scientist

The source code for IBM's Deep Blue, the first chess machine to beat then-reigning World Champion Gary Kasparov, was built by a team of about five people. That code has been languishing in a vault at IBM ever since because it was not created under a license that would enable further use by anyone, even though IBM is not attempting to make money from the code or using it for anything.

The second best chess engine in the world, Deep Junior, is also not free, and is therefore being worked on by a very small team. If

---

<sup>1</sup> One [website](#) documents 60 pieces of source code that perform Fourier transformations, which is an important software building block. The situation is the same for neural networks, [computer vision](#), and many other advanced technologies.

we have only small teams of people attacking AI, or writing code and then locking it away, we are not going to make progress any time soon towards truly smart software.

Today's chess computers have no true AI in them; they simply play moves, and then use human-created analysis to measure the result. If you were to go tweak the computer's value for how much a queen is worth compared to a pawn, the machine would start losing and wouldn't even understand why. It comes off as intelligent only because it has very smart chess experts programming the computer precisely how to analyze moves, and to rate the relative importance of pieces and their locations, etc.

Deep Blue could analyze two hundred million positions per second, compared to grandmasters who can analyze only 3 positions per second. Who is to say where that code might be today if chess AI aficionados around the world had been hacking on it for the last 10 years?

## DARPA Grand Challenge

Proprietary software developers have the advantages money provides; free software developers need to make advantages for each other. I hope some day we will have a large collection of free libraries that have no parallel available to proprietary software, providing useful modules to serve as building blocks in new free software, and adding up to a major advantage for further free software development.

What does society need? It needs information that is truly available to its citizens—for example, programs that people can read, fix, adapt, and improve, not just operate. But what software owners typically deliver is a black box that we can't study or change.

—Richard Stallman

The hardest computing challenges we face are man-made: language, roads and spam. Take, for instance, robot-driven cars. We could do this without a vision system, and modify every road on the planet by adding driving rails or other guides for robot-driven cars, but it is much cheaper and safer to build software for cars to travel on roads as they exist today — a chaotic mess.

At the annual American Association for the Advancement of Science (AAAS) conference in February 2007, the “[consensus](#)” among the scientists was that we will have driverless cars by 2030. This prediction is meaningless because those working on the problem are not working together, just as those working on the best chess soft-

ware are not working together. Furthermore, as American cancer researcher Sidney Farber has said, “Any man who predicts a date for discovery is no longer a scientist.”

Today, Lexus has a car that can parallel park itself, but its vision system needs only a very vague idea of the obstacles around it to accomplish this task. The challenge of building a robot-driven car rests in creating a vision system that makes sense of painted lines, freeway signs, and the other obstacles on the road, including dirt-bags not following “the rules”.

The Defense Advanced Research Projects Agency (DARPA), which unlike Al Gore, really invented the Internet, has sponsored several contests to build robot-driven vehicles:



*Stanley, Stanford University's winning entry for the 2005 challenge. It might not run over a Stop sign, but it wouldn't know to stop.*

Like the parallel parking scenario, the DARPA Grand Challenge of 2004 required only a simple vision system. Competing cars traveled over a mostly empty dirt road and were given a detailed series of map points. Even so, many of the cars didn't finish, or perform confidently. There is an expression in engineering called “garbage in, garbage out”; as such, if a car sees “poorly”, it is helpless.

What was disappointing about the first challenge was that an enormous amount of software was written to operate these vehicles yet none of it has been released (especially the vision system) for others to review, comment on, improve, etc. I visited Stanford's

Stanley [website](#) and could find no link to the source code, or even information such as the programming language it was written in.

Some might wonder why people should work together in a contest, but if all the cars used rubber tires, Intel processors and the Linux kernel, would you say they were not competing? It is a race, with the fastest hardware and driving style winning in the end. By working together on some of the software, engineers can focus more on the hardware, which is the fun stuff.

The following is a description of the computer vision pipeline required to successfully operate a driverless car. Whereas Stanley's entire software team involved only 12 part-time people, the vision software alone is a problem so complicated it will take an effort comparable in complexity to the Linux kernel to build it:

**Image acquisition:** Converting sensor inputs from 2 or more cameras, radar, heat, etc. into a 3-dimensional image sequence

**Pre-processing:** Noise reduction, contrast enhancement

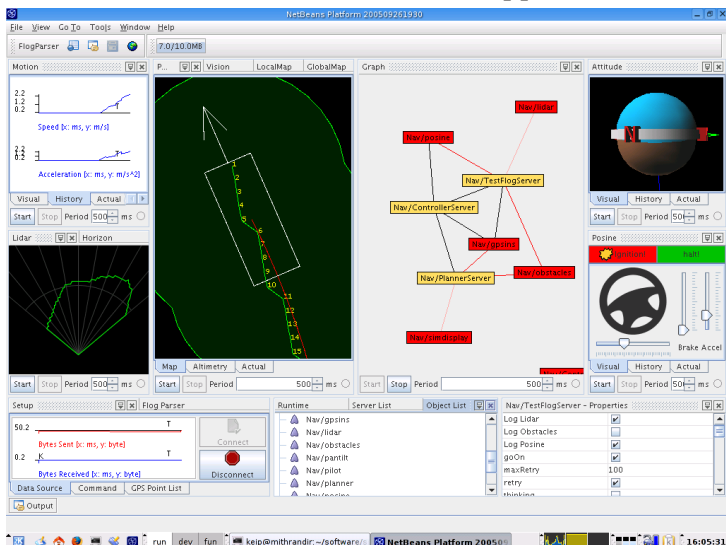
**Feature extraction:** lines, edges, shape, motion

**Detection/Segmentation:** Find portions of the images that need further analysis (highway signs)

**High-level processing:** Data verification, text recognition, object analysis and categorization

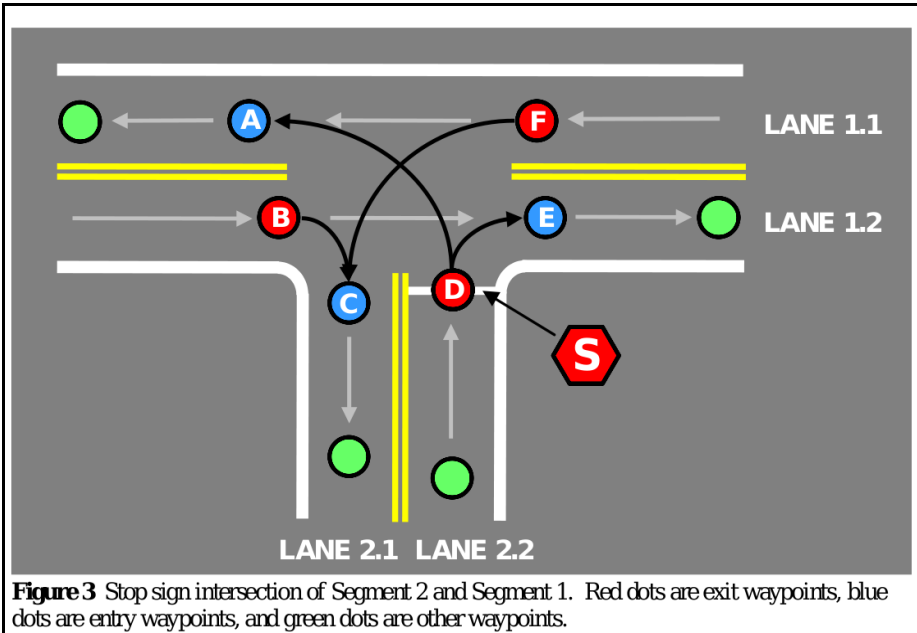
*The 5 stages of an image recognition pipeline.*

A lot of software needs to be written in support of such a system:



*The vision pipeline is the hardest part of creating a robot-driven car, but even such diagnostic software is non-trivial.*

In 2007, there was a new DARPA Urban challenge. This is a sample of the information given to the contestants:



*It is easier and safer to program a car to recognize a Stop sign than it is to point out the location of all of them.*

Constructing a vision pipeline that can drive in an urban environment presents a much harder software problem. However, if you look at the vision requirements needed to solve the Urban Challenge, it is clear that recognizing shapes and motion is all that is required, and those are the same requirements as had existed in the 2004 challenge! But even in the 2007 contest, there was no more sharing than in the previous contest.

Once we develop the vision system, everything else is technically easy. Video games contain computer-controlled drivers that can race you while shooting and swearing at you. Their trick is that they already have detailed information about all of the objects in their simulated world.

After we've built a vision system, there are still many fun challenges to tackle: preparing for Congressional hearings to argue that these cars should have a speed limit controlled by the computer, or telling your car not to drive aggressively and spill your champagne, or testing and building confidence in such a system.<sup>2</sup>

<sup>2</sup> There are various privacy issues inherent in robot-driven cars. When computers know their location, it becomes easy to build a "black box" that would record all

Eventually, our roads will get smart. Once we have traffic information, we can have computers efficiently route vehicles around any congestion. A [study](#) found that traffic jams cost the average large city \$1 billion dollars a year.

No organization today, including Microsoft and Google, contains hundreds of computer vision experts. Do you think GM would be gutsy enough to fund a team of 100 vision experts even if they thought they could corner this market?

There are enough people worldwide working on the vision problem right now. If we could pool their efforts into one codebase, written in a modern programming language, we could have robot-driven cars in five years. It is not a matter of invention, it is a matter of engineering. The world simply needs a Linus Torvalds of computer vision to step up and lead these efforts.

---

this information and even transmit it to the government. We need to make sure that machines owned by a human stay under his control, and do not become controlled by the government without a court order and a compelling burden of proof.

# Software and the Singularity

Futurists talk about the “Singularity”, the time when computational capacity will surpass the capacity of human intelligence. Ray Kurzweil predicts it will happen in 2045.<sup>3</sup> The flaw with any date estimate, other than the fact that they are always prone to extreme error, is that our software today has no learning capacity, because the idea of continuous learning is not yet a part of the foundation. Even the learning capabilities of an ant would be useful.

I believe the benefits inherent in the singularity will happen as soon as our software becomes “smart”. I don't believe we need to wait for any further Moore's law progress for that to happen. Computers today can do billions of operations per second, like add 123,456,789 and 987,654,321. Even if you could do that calculation in your head in one second, it would take you 30 years to do the billion that your computer can do in that second.

Even if you don't think computers have the necessary hardware horsepower to be smart today, understand that in many scenarios, the size of the input is the driving factor to the processing power required. In image recognition, for example, the amount of work required to interpret an image is mostly a function of the size of the image. Each step in the image recognition pipeline, and the processes that take place in our brain, dramatically reduce the amount of data from the previous step. At the beginning of the analysis might

---

3 His prediction is that the number of computers, times their computational capacity, will surpass the number of humans, times their computational capacity, in 2045. Therefore, the world will be amazing then.

This calculation is flawed for several reasons:

1. We will be swimming in computational capacity long before 2040. Today, my computer is typically running at 2% CPU when I am using it, and therefore has 50 times more computational capacity than I need. An intelligent agent twice as fast as the previous one is not necessarily more useful.
2. Many of the neurons of the brain are not spent on reason, and so shouldn't be in the calculations.
3. Billions of humans are merely subsisting, and are not plugged into the global grid, and so shouldn't be measured.
4. There is no amount of continuous learning built in to today's software.

Each of these would tend to push Singularity forward and support the argument that the benefits of singularity are not waiting on hardware. Humans make computers smarter, and computers make humans smarter, and this feedback loop makes 2045 a meaningless moment.

Who in the past fretted: “When will man build a device that is better at carrying things than me?” Computers will do anything we want, at any hour, on our command. A computer plays chess or music because we want it to. Robotic firemen will run into a burning building to save our pets. Computers have no purpose without us. We should worry about robots killing humans as much as we worry about someone stealing an Apache helicopter and killing humans today.

be a one million pixel image, requiring 3 million bytes of memory. At the end of the analysis is the data that you are looking at your house, a concept that requires only 10 bytes to represent. The first step, working on the raw image, requires the most processing power, so therefore it is the image resolution (and frame rate) that set the requirements, values that are trivial to change. No one has shown robust vision recognition software running at any speed, on any sized image!

While a brain is different from a computer in that it does work in parallel, such parallelization only makes it happen faster, it does not change the result. Anything accomplished in our parallel brain could also be accomplished on computers of today, which can do only one thing at a time, but at the rate of billions per second. A 1-gigahertz processor can do 1,000 different operations on a million pieces of data in one second. With such speed, you don't even need multiple processors! Even so, more parallelism is coming.<sup>4</sup> Once we build software as smart as an ant, we will build software as smart as a human the same day, because it is the same software.

---

4 Most computers today contain a dual-core CPU and processor folks promise that 10 and more are coming. Intel's processors also have limited 4-way parallel processing capabilities known as MMX and SSE. Intel could add even more of this parallel processing support if applications put them to better use. Furthermore, graphics cards exist to do work in parallel, and this hardware could also be adapted to AI if it is not usable already.



# Google

One of the problems faced by the monopoly, as its leadership now well understands, is that any community that it can buy is weaker than the community that we have built.

—Eben Moglen

In 1950, Alan Turing proposed a thought experiment as a definition of AI in which a computer's responses (presumed to be textual) were so life-like that, after questioning, you could not tell whether they were made by a human or a computer. Right now the search experience is rather primitive, but eventually, your search engine's response will be able to pass the Turing Test. Instead of simply doing glorified keyword matching, you could ask it to do things like: "Plot the population and GDP of the United States from 1900 – 2000."<sup>5</sup> Today, if you see such a chart, you know a human did a lot of work to make it.

The creation of machines that can pass the Turing Test will make the challenge of outsourcing seem like small potatoes. Why outsource work to humans in other countries when computers nearby can do the task?

AI is a meaningless term in a sense because building a piece of software that will never lose at Tic-Tac-Toe is a version of AI, but it is a very primitive type of AI, entirely specified by a human and executed by a computer that is just following simple rules.

Fortunately, the same primitive logic that can play Tic-Tac-Toe can be used to build arbitrarily "smart" software, like chess computers and robot-driven cars. We simply need to build systems with enough intelligence to fake it. This is known as "Weak AI", as opposed to "Strong AI", which is what we think about when we imagine robots that can pass the Turing Test, compose music, get depressed.

In Strong AI, you wouldn't give this machine a software program to play chess, just the rules. The first application of Strong AI is Search; the pennies for web clicks will pay for the creation of intelligent computers.

The most important and interesting service on the Internet is search. Without an index, a database is useless — imagine a phone directory where the names were in random order. There is an enormous turf war taking place between Google, Yahoo!, and Microsoft for the search business. Google has 200,000 servers, which at 200

---

5 Of course, there are some interesting complexities to the GDP aspect, like whether to plot the GDP in constant dollars and per person.

hits per second gives them the potential for three trillion transactions per day. Even at fractions of pennies per ad, the potential revenue is huge. Right now, Google has 65% of the search business, with Yahoo! at 20% and Microsoft at 7%. Bill Gates has said that Microsoft is working merely to keep Google “honest”, which reveals his acceptance that, unlike Windows and Office, MSN is not the leader. (Note that Microsoft's search and other online efforts have an inherent advantage because they get as much software as they want for free. Any other company which wanted to build services using Microsoft's software would have much higher costs.)

Furthermore, to supplant an incumbent, being 10% better is insufficient. It will take a major breakthrough by one of Google's competitors to change the game — Microsoft's Bing is not one of those. I use Google because I find its results good enough and because it keeps a search history, so that I can go back in time and retrieve past searches. If I started using a different search provider, I would lose this archive.

Google depends heavily on free software, but very little of their code is released to outsiders. One can *use* many of Google's services for free, as Google makes most of its money on advertising, but you cannot download any of their code to learn from it or improve it or re-use it in ways not envisioned by them. Probably 99% of the code on a typical server at Google is free software, but 99% of the code Google itself creates is not free software.<sup>6</sup> Google's source code is not only *not* freely available, it is not for sale.













































In fact, Google is an extremely secretive and opaque company. Even in casual conversation at conferences, its engineers quickly retreat to statements about how everything is confidential. Curiously, a paper explaining *PageRank*, written in 1998 by Google co-founders Sergey Brin and Larry Page, says, “With Google, we have a strong goal to push more development and understanding into the academic realm.” It seems they have since had a change of heart.

---

6 Although Google doesn't give away or sell their source code, they do sell an appliance for those who want a search engine for the documents on an internal Intranet. This appliance is a black box and is, by definition, managed separately than the other hardware and software in a datacenter.

It also doesn't allow tight integration with internal applications. An example of a feature important to Intranets is to have the search engine index all documents I have access to. The Internet doesn't really have this problem as basically everything is public. Applications are the only things that know who has access to all the data. It isn't clear that Google has attacked this problem and because the appliance is not extensible, no one other than Google can fix this either. This feature is one reason why search engines should be exposed as part of an application.

Google has sufficient momentum and sophistication to leave its competitors in the dust. Here is a list of Google's services:

|  |  |   |
|--|--|---|
| <b>Search</b>  |  <a href="#">Patent Search</a><br>Search the full text of US Patents  |  <a href="#">Gmail</a><br>Fast, searchable email with less spam                  |
|  <a href="#">Alerts</a><br>Get email updates on the topics of your choice                             |  <a href="#">Product Search</a><br>Search for stuff to buy  |  <a href="#">Groups</a><br>Create mailing lists and discussion groups            |
|  <a href="#">Blog Search</a><br>Find blogs on your favorite topics                                    |  <a href="#">Scholar</a><br>Search scholarly papers   |  <a href="#">Kno!</a><br>Share what you know                                     |
|  <a href="#">Book Search</a><br>Search the full text of books   |  <a href="#">Special Searches</a><br>Search within specific topics  |  <a href="#">Orkut</a><br>Meet new people and stay in touch with friends         |
|  <a href="#">Checkout</a><br>Complete online purchases more quickly and securely                      |  <a href="#">Toolbar</a><br>Add a search box to your browser  |  <a href="#">Picasa</a><br>Find, edit and share your photos                      |
|  <a href="#">Google Chrome</a> <sup>New!</sup><br>A browser built for speed, stability and security   |  <a href="#">Video</a><br>Search for videos on Google Video and YouTube                                     |  <a href="#">Reader</a><br>Get all your blogs and news feeds fast                |
|  <a href="#">Desktop</a><br>Search and personalize your computer                                      |  <a href="#">Web Search</a><br>Search over billions of web pages  |  <a href="#">Sites</a><br>Create websites and secure group wikis                 |
|  <a href="#">Earth</a><br>Explore the world from your computer  |  <a href="#">Web Search Features</a><br>Find movies, music, stocks, books, and more                         |  <a href="#">SketchUp</a><br>Build 3D models quickly and easily                  |
|  <a href="#">Finance</a><br>Business info, news, and interactive charts                               | <b>Explore and innovate</b>  |  <a href="#">Talk</a><br>IM and call your friends through your computer          |
|  <a href="#">GOOG-411</a><br>Find and connect with businesses from your phone, for free               |  <a href="#">Code</a><br>Download APIs and open source code   |  <a href="#">Translate</a><br>View web pages in other languages                  |
|  <a href="#">Google Health</a> <sup>New!</sup><br>Organize your medical records online              |  <a href="#">Custom Search</a> <sup>New!</sup><br>Create a customized search experience for your community |  <a href="#">YouTube</a><br>Watch, upload and share videos                      |
|  <a href="#">iGoogle</a><br>Add news, games and more to the Google homepage                         |  <a href="#">Labs</a><br>Explore Google's technology playground   | <b>Go mobile</b>  |
|  <a href="#">Images</a><br>Search for images on the web   | <b>Communicate, show &amp; share</b>   |  <a href="#">Maps for mobile</a><br>View maps and get directions on your phone |
|  <a href="#">Maps</a><br>View maps and directions   |  <a href="#">Blogger</a><br>Share your life online with a blog -- it's fast, easy, and free               |  <a href="#">Mobile</a><br>Use Google on your mobile phone                     |
|  <a href="#">News</a> - now with <a href="#">archive search</a><br>Search thousands of news stories |  <a href="#">Calendar</a><br>Organize your schedule and share events with friends                         |  <a href="#">SMS</a><br>Use text messaging for quick info                      |
|  <a href="#">Notebook</a><br>Clip and collect information as you surf the web                       |  <a href="#">Docs</a><br>Create and share your online documents, presentations, and spreadsheets          | <b>Make your computer work better</b>   |
|  |  |  <a href="#">Pack</a><br>A free collection of essential software               |

*Google is applying Metcalfe's law to the web: Gmail is a good product, but being a part of the Google brand is half of its reason for success.*

Even with all that Google is doing, search is its most important business. Google has tweaked its patented PageRank algorithm extensively and privately since it was first introduced in 1998, but the core logic remains intact: The most popular web pages that match your search are the ones whose results are pushed to the top.<sup>7</sup>

Today, PageRank can only rank what it understands. If the database contains words, it ranks the words. PageRank lets the wisdom in millions of web sites decide what is the most popular, and therefore the best search result — because the computer cannot make that decision today. PageRank is an excellent stopgap measure to the problem of returning relevant information, but the focus should be on putting richer information into the database.

I believe software intelligence will get put into web spiders, those programs that crawl the Internet and process the pages. Right now, they mostly just save text, but eventually they will start to understand it, and build a database of knowledge, rather than a database of words. The rest is mostly a parsing issue. (Some early search engines, treated digits as words: searching for 1972 would find any reference to 1, 9, 7 or 2; this is clearly not a smart search algorithm.)

The spiders that understand the information, because they've put it there, also become the librarians who take the search string you give it, and compare that to its knowledge.<sup>8</sup> You need a librarian to build a library, and a librarian needs the library she built to help you.

Today, web spiders are not getting a lot of attention in the search industry. Wikipedia documents 37 web crawlers, and it appears that the major focus for them is on performance and discovering spam websites containing only links which are used to distort rank.<sup>9</sup>

---

7 Some of Google's enhancements also **include**: freshness, which gives priority to recently-changed web pages. It also tries to classify queries into categories like places and products. Another new tweak is to not display too many results of one kind: they try to mix in news articles, advertisements, a Wikipedia entry, etc. These enhancements are nice, but are far from actually understanding what is in the articles, and it appears to be applying smarts to the search query rather than the data gathered by the spiders.

8 One might worry about how a spider that has only read a small portion of the Internet can help you with parts it has not seen? The truth is that these spiders all share a common memory.

9 Focusing on the spider side means continually adding new types of information into the database as it starts to understand things better. Let's say you build a spider that can now understand dates and guess the publication date of the page. (This can be tricky when a web page contains biographical information and therefore many dates.) Spiders will then start to tag all the web pages it reads in the

The case for why a free search engine is better is a difficult one to make, so I will start with a simpler example, Google's blogging software.

## Blogger

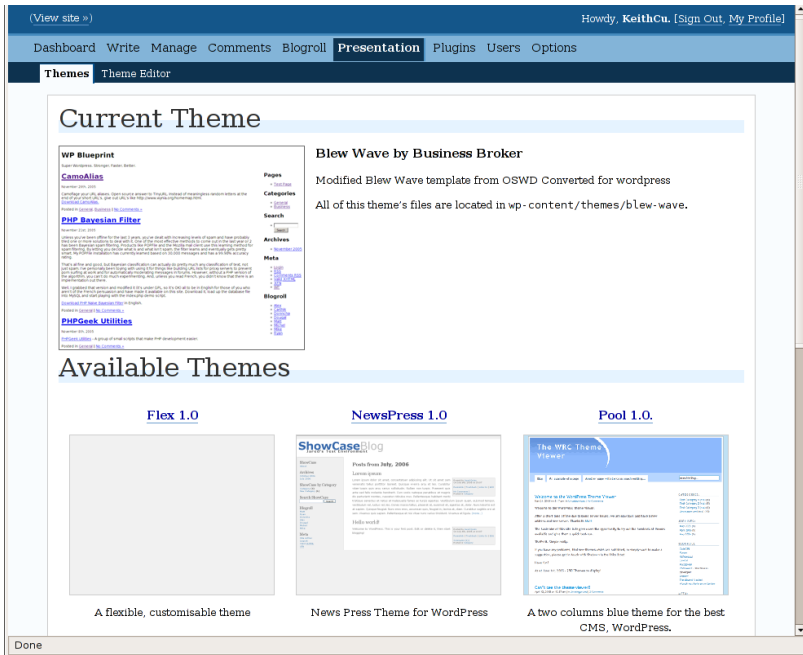
While Google has 65% of the billion-dollar search business, it has 10% or less of the blog business. There exists an enormous number of blog sites, the code for which is basically all the same. The technology involved in running Instapundit.com, one of the most influential current-events blogs, is little different than that running Myspace, the most popular diary and chatboard for Jay-Z-listening teenage girls.

Google purchased the proprietary blogging engine Blogger in 2000 for an undisclosed amount. Google doesn't release how many users they have because they consider that knowledge proprietary, but we do know that no community of hundreds of third party developers is working to extend Blogger to make it better and more useful.

---

future with this new information. All of this tagging happens when the data is fetched, so that is where the intelligence needs to go.

The most popular free blogging engine is WordPress, a core of only 40,000 lines of code. It has no formal organization behind it, yet we find that just like Wikipedia and the Linux kernel, WordPress is reliable, rich, and polished:



*WordPress, the most popular free blogging engine*

My only complaint is that third-party community contributions are not just one-click to install and upgrade, as they are in an operating system. Despite that tiny drawback, WordPress is supported by a community of developers, who have created plug-ins, written and translated documentation, and designed many themes to customize the look.

Here are the *categories* of plug-ins available for WordPress:

|   |  |
|---|--|
| <b>Administration</b><br>Administration Tools<br>Advertisement<br>Anti-Spam<br>Comments<br>Meta (tagging)<br>Restrictions<br>Statistics<br>Syntax Highlighting<br>Syndication<br>Translation and Languages<br>Tweaking<br><br><b>Monetizing</b><br><br><b>Design, Layout and Styles</b><br>Archive<br>Calendar - Event<br>Navigation<br>Randomness<br>Styles<br>Widgets<br><br><b>Links</b><br>3rd-parties services integration | <b>Graphics, Video, and Sound</b><br>Audio<br>Images<br>Multimedia<br>Video<br><br><b>Odds and Ends</b><br>Financial<br>Forums<br>Geo<br>Miscellaneous<br>Mood<br>Time<br>Weather<br><br><b>Outside Information</b><br>Del.icio.us<br>Technorati<br><br><b>Posts</b><br>Audio Posts<br>Editing Posts<br>Formatting Posts<br>Miscellaneous Post Plugins |
|---|--|

*There are hundreds of add-ons for WordPress that demonstrate the health of the developer community and which make it suitable for many websites, including an entire corporate presence. Like everything else in the free software community, it is being built seemingly by accident. This might look like a boring set of components, but if you broke apart the billion-dollar MySpace, or a website like CNN's, you would find much of the same functionality.*

Google acquired only six people when it purchased Pyra Labs, the original creators of Blogger, a number dwarfed by WordPress's hundreds of contributors. As with any thriving ecosystem, the success of WordPress traces back to many different people tweaking, extending and improving shared code.

In addition to blogging software, I see other examples where Google could have worked more closely with the free software community. Recently I received many e-mails whose first words were: "Your cr. rating doesn't matter" that I dutifully marked as spam. It

took weeks before Gmail's spam filter caught on. Spam is a very hard problem and cooperating with others could help improve Google faster, and lower their R&D costs.

## Search

Google tells us what words mean, what things look like, where to buy things, and who or what is most important to us.

Google's control over "results" constitutes an awesome ability to set the course of human knowledge.

—[Greg Lastowka](#), Professor of Law, Rutgers University

And I, for one, welcome our new Insect Overlords.

—News Anchorman Kent Brockman, *The Simpsons*

Why Google should have built Blogger as free software is an easier case to make because it isn't strategic to Google's business or profits, the search engine is a different question. Should Google have freed their search engine? I think a related, and more important question is this: Will it take the resources of the global software community to solve Strong AI and build intelligent search engines that pass the Turing Test?

Because search is an entire software platform, the best way to look at it is by examining its individual components. One of the most fundamental responsibilities for the Google web farm is to provide a distributed file system. The file system which manages the data blocks on one hard drive doesn't know how to scale across machines to something the size of Google's data. In fact, in the early days of Google, this was likely one of its biggest engineering efforts. There are (today) a number of free distributed file systems, but Google is not working with the free software community on this problem. One cannot imagine that a proprietary file system would provide Google any meaningful competitive advantage, nevertheless they have built one.

Another boring, but nontrivial task for a search engine is the parsing of PDFs, DOCs, and various other types of files in order to pull out the text to index them. It appears that this is also proprietary code that Google has written.

It is a lot easier to create a Google-scaled datacenter with all of its functionality using free software today than it was when Google was formed in 1998. Not only is Google not working with the free software community on the software they have created, they are actually the burdened first-movers. What you likely find running on a Google server is a base of Linux and other free software, upon which Google has created their custom, proprietary code. Google



might think their proprietary software gives them an advantage, but it is mostly sucking up resources, and preventing them from leveraging advancements from outside developers.

And like Microsoft's Windows NT kernel, even if Google were to release their infrastructure code, much of it would not be picked up because the free software community has developed their own solutions. In fact, Google has recently started to release bits and pieces of their most boring software, but there doesn't appear to be much uptake because it isn't nearly as interesting as it would have been ten years ago.

What about the core of Google's business, the code that takes your search request and attempts to make sense of it so that it can pass the Turing Test? Google has not even begun to solve this problem, and even many simpler problems, so it makes one wonder if it is something a single company can solve by itself.

There are two kinds of engineering challenges for Google:

Those necessary, boring and non-strategic, and at best loosely correlated to their profits, like blogging, language translation, and spam detection.

Then there is the daunting problem of building software with Strong AI which Google *had better* be working on with the rest of the world. The idea of Google "owning" Strong AI is at least as scary as Microsoft owning Windows and Office. Google has publicly stated that Microsoft's proprietary software model has been bad for the industry, but doesn't recognize that it is trying to do the exact same thing!

Google is one of the few new, large, and fast-growing software businesses in America and few people are publicly arguing that the company give away the farm by sharing their core technology with the free software community. This is especially scary because it is an irreversible step. However, software is not a datacenter or a relationship with customers and advertisers. Most of the users of Google's code would not be in competition with Google, but would be taking it to new places that they hadn't considered. Furthermore, Google would still have a significant first-mover advantage of the code they created.

In addition, if someone else eventually creates a free search engine that a worldwide community of researchers coalesce around, where will Google be then? Perhaps Microsoft or Yahoo! could flank Google by building a free search engine that scientists and researchers around the world could tinker in.

# Conclusion

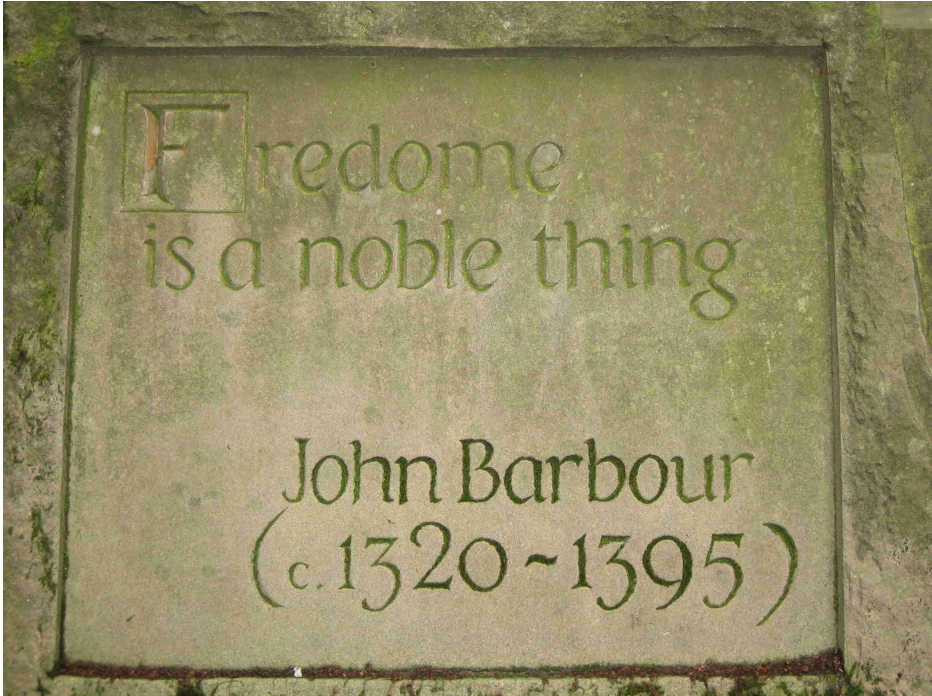
There is reason for optimism about the scientific challenges we confront because the global community's ability to solve problems is greater than the universe's ability to create them. The truth waiting for us on the nature of matter, DNA, intelligence, etc. has been around for billions of years.

There are millions of computer scientists sitting around wondering why we haven't yet solved the big problems in computer science. It should be no surprise that software is moving forward so slowly because there is such a lack of cooperation.

# FREE SOFTWARE

If you have an apple and I have an apple and we exchange these apples then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas.

—George Bernard Shaw



*Inscription found on a wall in Edinburgh.*

**M**uch of man's existence until the late 20<sup>th</sup> century involved an economy focused primarily on the manipulation of scarce, unmalleable atoms. Copyright law was created to protect writers, the true creators, and to regulate the few who had the means of making copies — the publishers. In a digital world, we can all be creators and publishers, so we need to revisit many fundamental questions, from the means we use to protect ideas, to the ways in which we purchase them.

This and the next chapter discusses details of free software, copyright and patent laws, but let's start by remembering that a shift towards a presumption of free digital intellectual property is actually a moral question. The Internet makes transmitting knowledge essentially free, so as free software evangelist Eben Moglen asks: "If

you could feed the world for free, would you? Likewise, if you could provide every child access to a library of human knowledge they would ever outgrow, would you?" It is the Internet that makes this question possible to ask, and necessary to answer!

Without the right software to decode and manipulate it, a digital idea is just a blob of bits to your computer. With the Internet, we can exchange bits, but with free software, we can exchange ideas. While free knowledge and free software are not any direct goal of the free market, they provide tremendous benefits to a free market because they allow *anyone* to create further value. If the larger goal to encourage as many programmers as possible to write software, then the free software approach has already demonstrated its superiority, even though it is only on 1% of desktops. In fact, the proprietary world was always destined to have just one company dominate; a clone of Bill Gates who came along later would have been unable to learn from and improve upon the innovations of the first.

Free software brings the libertarian benefit of allowing information to be used in unlimited new ways, combined with the communitarian benefit of ensuring that no one is left behind by the access cost of knowledge. Because "free software" is better for the free market than proprietary software, and an important element of a society characterized by the free exchange of ideas, I think it is a better name than "open source", although both represent the same basic idea.

# Software as a Science

In any intellectual field, one can reach greater heights by standing on the shoulders of others. But that is no longer generally allowed in the proprietary software field—you can only stand on the shoulders of the other people *in your own company*.

The associated psychosocial harm affects the spirit of scientific cooperation, which used to be so strong that scientists would cooperate even when their countries were at war. In this spirit, Japanese oceanographers abandoning their lab on an island in the Pacific carefully preserved their work for the invading U.S. Marines, and left a note asking them to take good care of it.

—Richard Stallman

Software is a science: you reason, you create a hypothesis in the form of source code, and then you test the hypothesis by running it on a computer. Linus Torvalds [summarized](#) the similarity:

Science may take a few hundred years to figure out how the world works, but it does actually get there, exactly because people can build on each others' knowledge, and it evolves over time. In contrast, witchcraft/alchemy may be about smart people, but the knowledge body never “accumulates” anywhere. It might be passed down to an apprentice, but the hiding of information basically means that it can never really become any better than what a single person/company can understand.

And that's exactly the same issue with open source (free) vs proprietary products. The proprietary people can design something that is smart, but it eventually becomes too complicated for a single entity (even a large company) to really understand and drive, and the company politics and the goals of that company will always limit it.

Even the word “university”, man's place for shared study, derives from the Latin *universitas magistrorum et scholarium*, meaning “a community of teachers and scholars.” Universities were long understood to be places where people were placed together to learn from each other.

Unfortunately, today, proprietary software has spread from the corporate world to universities and other public institutions. If corporations want to hoard their scientific advancements, that is fine, albeit short-sighted, but our public institutions should not be following suit! Not only Stanford's robot-driven car, Stanley, but also *a ton* of other proprietary software is written by public institutions today. Just unleashing our public institutions towards free software will greatly increase the pace of progress, without even accounting for the software funded by corporations.

Some think of free software as a Marxist idea, but science has always been public because it was understood that giving the knowledge away would spurn further innovation, and because the scientist needed some shoulders to stand on in the first place. Corporations were created not to hoard knowledge but to take the advancements in science and apply them to practical uses. There still is plenty of opportunity for competition and free markets, even if all of the advancements in science are freely available.

Scientists' work is reviewed by peers, published in journals, and discussed at conferences; the exchange of new snippets of knowledge between participants is fundamental to further progress. This even allows for competition to be the first to discover something, or put the advancement to commercial use.

With proprietary software, we've created something outside the classically accepted model of scientific research, and even worse, it has become a very dominant model in private and public institutions. Letting one company own an innovation in science might provide it an incentive to sell better products, but it limits the many other people who could use that advancement in other ways.

Science is not all knowledge. Science is not a business, a service, or a product. And to the extent that it took science to make a product, patents were created to protect ideas that were easily copied but not easily invented or discovered.

One could even argue that patents are not necessary to stimulate progress because the challenge someone has when going up against GE's light bulb idea is the infrastructure to produce and distribute such a product, and the knowledge gained in doing all of this. An old line in business is that the one who wins is "the firstest with the mostest." Large companies have economies of scale and satisfied customers that any newcomers would have to overcome, even if they built a better product.

Economies of scale are great for the free market because they are a powerful driver of increased efficiency and quality, but they also mean that to defeat someone you need to be more than 10% better. The need for breakthroughs to defeat incumbents is good for the free market because it forces newcomers to think big. Fortunately, transformative technologies come along frequently enough that no incumbent is ever truly safe.

Because software is a science, making it freely available may hurt proprietary software companies, but it will help every other type of company.

# Definition of Free Software

Making Linux use the GPL license was definitely the best thing I ever did.

—Linus Torvalds

Because software is a science, we need to create license agreements which allow, and even encourage, cooperation among programmers. Computer scientists need software to be freely available for them to do their work.

Richard Stallman has defined the four basic software freedoms:

1. The freedom to run the program, for any purpose. (You, not your software, are in control of what is happening.)
2. The freedom to study how the program works and adapt it to your needs.
3. The freedom to give a copy of the program to your neighbor. Sharing ideas is impossible without sharing the program to create and display them.
4. The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

The GNU General Public License (GPL) is the copyright mechanism he came up with to protect those freedoms, which will allow maximal re-use of advancements. The goal is to protect the freedom of the user of free software. Without the ability to study and manipulate the software, you are greatly limited in your ability to use it and further advance it.

Copyright was created to protect the creators, from the publishers (those with the means to make copies), by granting the creator exclusive rights. The GNU GPL is sometimes called “*copyleft*”, because it grants the same expansive rights to everyone, creator, and user.

It sounds backwards to protect users rather than creators, but protecting users also helps creators. All Linux programmers except Linus started off as users. Linus was the first user of Linux as well as being the first contributor to Linux. He fully owned his code and could fix any problem he found. Copyleft ensures that code protected with this license provides those same guarantees to future users and creators.

Copyleft helps Linus because it encourages, even requires users and programmers of Linux to give back to his invention. Linux only

ran on a 80386 CPU when first released because that is what Linus owned. All the improvements that it took to run on the other processors got put into Linux, to the benefit of Linus, and everyone else.

Whatever you think of free software today, using it is a choice. In fact, creating it is a charitable act, and we should be grateful to Linus Torvalds for releasing his work, just like we should be grateful to Einstein for making his  $E=MC^2$  theory publicly available.

## Copyleft and Capitalism

With the GPL freedoms comes one important responsibility: ensuring future enhancements to free code have to stay as free as the original code. This encourages a scientific community to stay that way, but is a responsibility that only applies to the users who choose to join the set of creators.

Free software is free for everyone but for the developers who will take it into new places. Software protected by copyleft is called free software, but it is not truly free because of the assumption that future advancements will also be made free. There is no free lunch in this world, and copyleft ensures that people are giving back to this free software. This copyleft obligation is necessary but not expensive.

### Necessary

The reason it is necessary to have copyleft is that only 100% free software is something *someone else* can further improve. Improving a piece of free software, and making the enhancements proprietary, effectively makes that entire piece of software proprietary. Denying free availability of code enhancements creates a new boundary between science and alchemy.

### Not Expensive

Free software is not expensive because, in practical terms, advancements in software are nearly always based on existing knowledge and are very small in scope. What you must give back is much smaller than what you have freely received.

When Linus first released Linux, it was a mere 10,000 lines of code. He has since received 8.2 million lines in return, worth 4,000 man-years, or \$400 million. Free software, like science, is a pyramid scheme that works. The idea that someone gives you \$400 million worth of technology for free means that it is likely you will not need to do much to get it to work for you, and your copyleft obligation will be a very small fraction of what you have freely received.



A change in software is usually 10-100 hundred lines of code, which is .001% to .01% of a million-line codebase. Technology advances in a steady, stepwise fashion. Therefore, it is important that once a piece of software is free, every little advancement to it is also made free. Wikipedia and Linux both require the copyleft obligation, and I believe this is an important part of their success. They also demonstrate that the copyleft obligation is enough to sustain development of free software, without involving license fees.

Free software positions other people as the beneficiaries of your solutions, so they can then focus on other problems, to your benefit. Free software allows for an optimal division of labor, performed by people who don't necessarily understand the whole picture, but are just trying to make the software work better for them, or a customer of theirs.

A free license doesn't take away your ability to benefit from your work. And as long as it doesn't take away your ability to benefit, there will always be motivation to write free software. Just as the public availability of scientific advancements has not decreased the motivation to do science, the free flow of software innovations does not threaten to undermine the software market. If people only make changes to software when they want to, and those changes are captured for others to use, no one is worse off. It helps others, and it likely helps the creator because writing software is a science and hard work.

Microsoft, Apple, Google, and many of the other blue-chip computer companies do not yet accept the idea that software should be free in the ways that Stallman defines above. According to them, you can generally run the code for whatever purpose, but not copy, study, or enhance it. Changing the license for the world's most important pieces of software will increase the pace of progress.

## Is Copyleft a Requirement for Free Software?

An equitable and effective intellectual property system must take into account both first creators and those who come later to build upon their work.

—Committee for Economic Development

Richard Stallman's copyleft, the idea of ensuring that free ideas stay free, is one of the most important concepts in the digital age. It is such a radical idea that it is not universally accepted, even within the free software community.

In fact, source code is considered free by today's software community if it supports the first three freedoms (run, study, copy), but not copyleft (make those enhancement freely available to all.)

Two very popular licenses, the MIT and BSD licenses, are considered free but simply say: "Please include this copyright message at the top of the source code." You can use this code, copy it and study it, but you can also make it proprietary again. This sort of free software does not require anyone to contribute anything back if they enhance it.

Stallman considers these lax licenses; while they sound reasonable and are, strictly speaking, more "free" than copyleft, the problem is that this once-free software frequently becomes proprietary again. Keith Packard has told the story of how the Unix windowing system was initially created under a lax license, but was rewritten multiple times because it got hijacked and made proprietary multiple times. An enormous amount of programming work was wasted but because the codebase was not GPL right from the beginning. As Eben Moglen points out, things in public domain can be appropriated from in freedom-disrespecting ways.

One of the reasons why Unix was never much competition for Windows is that many of its vendors did not work together. Linux's GPL license nudges people into working together to save money in total development costs.

Further adoption of copyleft will increase the efficiency of the free software community and help it achieve world domination faster. Software protected by copyleft is not totally free software, and that is why it is taking off.

# Why write free software?

There is only one valid definition of a business purpose: to create a customer.

—Peter Drucker

What the free software movement showed was that by proving a concept within reach and offering some working model, no matter how defective, partial or bad, in the presence of a community sharing the objective, the achievement of the outcome is simply a matter of allowing people to work freely with one another.

In the world of software, which is low capital intensive in every sense, that's all that's required for the output to take form. We built tens, and then hundreds of billions of dollars, in valuable software using almost no venture capital inputs.

—Eben Moglen

While there are many reasons to write software, it is still worth asking whether it will remove future motivations when the work becomes available for others to use and build upon.

The problem with this question is that it's a trick: a person is only motivated to write software when there is a need to improve what currently exists. If software doesn't work properly for someone, *that* is a motivation to make it better. Having a license fee for an enhanced version is one motivation, but there can be others.

Windows and Office deliver half of Microsoft's revenues but are less than half of their engineering. Microsoft uses their dominance in those two markets to fund a number of other much less profitable ventures — pretty much the rest of the company. Even worse, because Windows and Office expertise and code are locked up, the entire industry must wait for Microsoft to improve its products on whatever shipping cycle they choose.

In a free software world, resources can flow where they are needed, in the quantity they are needed. Code to support a feature gets written if the feature is important. Free software enables a freer flow of resources than proprietary software, and therefore enables higher quality.

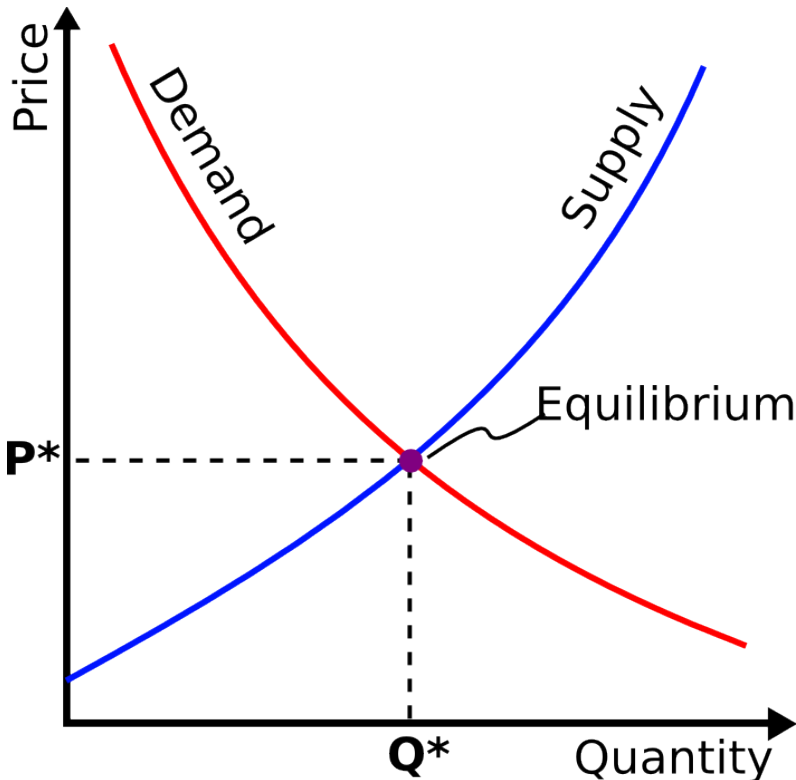
The job of a business is to make satisfied customers, and free software increases possibilities for service businesses, as I will discuss later. While law libraries and LexisNexis contain millions of *freely* available legal pleadings and decisions, no one has said this has decreased the motivation to become a lawyer! In fact, imagine trying to be a lawyer *without* access to these resources.

# 1. Laws of Supply and Demand Say So

Microsoft was the company that turned the software industry on its head by introducing lower-cost solutions years ago to undermine the Unix businesses of IBM and Hewlett-Packard, and the database businesses of Oracle and IBM. If anyone knows the importance of pitching the market on low-cost, high-value software, it's Microsoft.

—Matt Asay

There are many answers to the question of motivation, but it is important to start by mentioning that analyzing motivations is a secondary level of analysis. To understand the free market of free software you have to start with the laws of supply and demand:<sup>1</sup>



*Conventional models of economics do not apply to products with zero marginal cost. The model above assumes there can be supply shortages that would tend to increase prices.*

Copying and other manipulation of bits is a computational task whose marginal cost is the cost of the electricity, which is dropping according Moore's law because smaller transistors use less electric-

<sup>1</sup> This analysis doesn't consider the elasticity of demand, etc. but while those considerations add complications, they do not overturn the basic laws.

ity. When you assume the computational cost is zero, you no longer can have supply shortages of bits. The law of demand dictates that as prices are lowered, demand will increase. Consequently, a product with zero cost should, in principle, have infinite demand. One answer to why people will write free software in the future is because there will be infinite demand for it. Wikipedia, Linux, Fire-Fox, and many other free software products, have user bases which are growing every year, and are even taking away marketshare from proprietary products, as predicted.<sup>2</sup>

The law of supply says that higher prices give producers an incentive to supply more in the hope of making greater revenue. The supply curve seems to suggest that if the price of a product is zero, producers will have no incentive to make anything. However, the supply curve never envisioned that the marginal cost of production would be zero. It is this difference which upends the conventional economic rules.

Something which requires no marginal cost to make, and is acquired by consumers for no cost, should have infinite supply and infinite demand. Free software will take off because the most basic laws of economics say it should. If Wikipedia charged \$50 for the privilege of *browsing* their encyclopedia, they would not have had the millions of *contributors* and tens of millions of enhancements they have received so far. Wikipedia's infinite supply of free bits is creating an infinite demand.

---

2 There is the total cost of ownership (TCO), but that is a comparative measure of software quality. If you use some free software to help you make a website, the cost of that software is the amount of time you spend with it to accomplish your task. If one tool builds a website in half the time, or builds one twice as nice, then the TCO of those software packages is different, even if both are free. If a car company gave away their top of the line cars for free, the demand would be very high, even though the owners still had to purchase gas and had other ongoing costs. Even if two pieces of software have the same TCO, there is an additional cost: the cost of switching from one to the next. Software is not interchangeable in the way that the laws of supply and demand envision. In fact, the switching costs between software often dwarfs the difference in their TCO.

## 2. Services & Support

Free enterprise cannot be justified as being good for business. It can be justified only as being good for society.

This new knowledge economy will rely heavily on knowledge workers. The most striking growth will be in “knowledge technologists:” computer technicians, software designers, analysts in clinical labs, manufacturing technologists, and paralegals.

—Peter Drucker

There are many opportunities for volunteers and public institutions to create free software, but commercial enterprises *need* software to run their business, and they are an enormous potential source of funding. *For-profit service and support organizations will provide the persistence that corporations will demand of free software.*

As with other sciences, there should be many avenues for corporations to make money via the use and production of freely-available advances in computer science. Service companies will write free software when a customer pays them, just as a physicist or lawyer does work whenever a customer pays them. In fact, by some estimates, 75% of software is written for internal use inside a corporation, without any thought of selling it to others. This corporate software is “free” to its customers inside a corporation. Software companies making licensing revenue is already a very small part of the software industry today.

Free software is much more conducive to creating robust software services business because all of the relevant information is publicly available. In the world of proprietary software, the company that wrote the software is typically the only one capable of providing support. Microsoft has created a huge software ecosystem, but the free software service ecosystem has the potential to be *much* larger. Of course, there is no guarantee of quality of service providers, but this same issue exists today with car mechanics.

Today, many free software projects have thriving service and support communities around them. While others are not particularly healthy yet, this is a function of the small overall marketshare of free software, not any fundamental flaw in the business model.

In fact, the proprietary model creates fundamental limitations in the service business. When I left Microsoft, I took on a consulting job helping a team build a website which used Microsoft Passport as the authentication mechanism. However, as I ran into problems, even Google wasn't able to help because the knowledge I needed to

fix my problems was locked up behind the Microsoft firewalls. Even though I was an expert in many Microsoft technologies, I was stymied and needed to use my old contacts to get the code working. If I hadn't been a former employee, I would not have been able to solve my problem. Fixing a problem in proprietary software can sometimes feel like performing witchcraft — you have to try lots of random incantations because you can't know what is really going on.

I spoke with an IT employee in a hospital who told me that after her hospital purchased some software, their vendor became unresponsive. After making their money on the sale, they had no motivation to help the hospital anymore. The hospital fought with their software supplier about whether the enhancements they were requesting were *bugs*, which would be fixed quickly and for free, or *features*, which would cost money and weren't guaranteed to be made for a year or more. To the hospital, this distinction was irrelevant: lives were on the line, and they needed improvements, and they didn't care how their vendor categorized them!

In a services business, the vendor gets paid to make the customer happy, not before the product is in use. Restructuring the software industry into a services business based on free software will increase customer satisfaction and strengthen the relationship between vendor and customer.

There is an enormous market in software for running a business. One of the biggest markets is known as Enterprise Resource Planning (ERP), an umbrella term for software used to manage the back office of an enterprise. ERP covers everything from payroll to customers to inventory:

| <b><i>Category</i></b>                  | <b><i>Features</i></b>   |
|---|--|
| <b>Manufacturing</b>                    | Engineering, Bills of Material, Scheduling, Capacity, Workflow Management, Quality Control, Cost Management, Manufacturing Process, Manufacturing Projects, Manufacturing Flow |
| <b>Supply Chain Management</b>          | Inventory, Order Entry, Purchasing, Product Configuration, Supply Chain Planning, Supplier Scheduling, Inspection of goods, Claim Processing, Commission Calculation           |
| <b>Financials</b>                       | General Ledger, Cash Management, Accounts Payable, Accounts Receivable, Fixed Assets   |
| <b>Projects</b>                         | Costing, Billing, Time and Expense, Activity Management  |
| <b>Human Resources</b>                  | Human Resources, Payroll, Training, Time & Attendance, Benefits  |
| <b>Customer Relationship Management</b> | Sales and Marketing, Commissions, Service, Customer Contact and Call Center support  |
| <b>Data Warehouse</b>                   | and various Self-Service interfaces for Customers, Suppliers, and Employees  |

*The major modules of Enterprise Resource Planning (ERP).*

ERP is a multi-billion dollar industry, and today it is dominated by proprietary software. This is ironic because ERP solutions, unlike a word processor, are specifically customized to a particular business. The need for customization suggests that ERP software could be free, but with a robust service business behind it: customizing it, installing it, helping import and manage the data, providing training and technical support, etc. Enhancements made by various service providers could be fed back into the core product, and service providers could themselves be trained and certified, thus providing revenue for the core development.

One of the perceived weaknesses of free software is that there is no single owner of the work, and therefore no one has skin in the game, or a throat to choke when something goes wrong. This thinking is faulty because it assumes the computing environment is simple and homogeneous and makes about as much sense as having one doctor for your dermatology and proctology needs. Computing



systems today are not only large and complicated, they are also heterogeneous. In the 1970s, companies like IBM provided all the hardware, software, and services necessary to run a business, but today's computing environments are very different. Even in a homogeneous Microsoft shop where all of the servers are running Windows, SQL Server, and .Net, you still might use HP hardware and administration tools, Intel chips and drivers, an EMC disk farm, etc.

Computer software is not smart yet, but don't let that fool you into thinking that it is not large and complicated. A trained engineer can become an expert at 100,000 lines of code, but because a modern computer contains 100 million lines of code, you need at least 1,000 different people to help with all possible software problems. In other words, it is a fact of life in a modern IT department that, whether using free or proprietary software, support will require relationships with multiple organizations.

In fact, free software can allow service and support teams to better help you because they can build expertise in more than one area because all of the code and other relevant information is out there. Service companies can even build a hierarchy of relationships. You might call HP to get help with your server, and HP might have an escalation relationship with MySQL if they track it down to a database problem they can't fix. These hierarchies can provide one throat to choke.

### 3. Lowers Hardware and Software Costs

All hardware companies have a compelling reason to use and support free software: it lowers their costs. IBM and Cray are happy to give you a Linux OS for free, so you can put your money toward the supercomputer they are selling. The Playstation 3 runs Linux, with Sony's blessing, because it is another reason to buy their hardware and take their product to new places they have yet to exploit.

Free software lowers the cost of hardware, and its greater usage will stimulate new demand for computers and embedded devices. If a complete, free software stack were magically available today that enabled computer vision and speech, our toys would have them tomorrow. A world of rich free software is a world with amazing hardware.

Free software levels the playing field and makes the hardware market richer and more competitive. One of the reasons an MRI machine is expensive is because the software is proprietary. When a hardware vendor controls the software, it can set the price at the cost of the hardware plus the cost to develop the software, rather

than something approximating their hardware costs. If MRI software were free, the hardware cost would drop, more people could afford an MRI, and the quality would increase faster.

In fact, there already is free, high-quality scientific software suitable for building an MRI machine (in products like SciPy), but the current manufacturers build their products using proprietary software. They aren't colluding with each other, but it reminds me of the old days of databases where your only choice was whether to pay many thousands for Oracle or DB2. The healthcare hardware companies had better watch their backs!

Even proprietary software companies have an incentive to use free software, to lower *their* costs. It is ironic that Microsoft could make higher profits, and build better products, by using free software.

## 4. Educational uses

I once asked some of my computer science lecturers why they didn't get students to do something useful, like work on free software, instead of assigning them pointless busy work projects. Two main answers:

1. It's too hard to grade. (*Why?*)
2. It's seen by many to be exploitative. (*As opposed to busy-work?*)

—Slashdot.org commentator

Dear Ken Starks (founder of Helios Project), I am sure you strongly believe in what you are doing but I cannot either support your efforts or allow them to happen in my classroom. At this point, I am not sure what you are doing is legal. No software is free and spreading that misconception is harmful. I admire your attempts in getting computers in the hands of disadvantaged people but putting Linux on these machines is holding our kids back. This is a world where Windows runs on virtually every computer and putting on a carnival show for an operating system is not helping these children at all. I am sure if you contacted Microsoft, they would be more than happy to supply you with copies of an older version of Windows and that way, your computers would actually be of service to those receiving them.

—Karen, middle school teacher

Students in every field use software in their studies; this is free brainpower! In the computer world, there are two levels of programming: the boring work that engineers do, and the “fancy” coding that researchers do.

Given that software PhD-types follow the lead of other fields and release their ideas publicly in papers, and want people to use their

ideas, you'd think making their code freely available would be a part of the computer science research culture today, but it isn't, even in universities. There is a paper for Stanford's Stanley, but no code. Releasing software with the paper is the exception rather than the rule today.

Even though all the key ideas are available in a paper, re-using the ideas in such a document takes a lot more time than working with the software directly. You can reuse software without fully understanding it, but you can't re-implement software without fully understanding it!

At Microsoft, both a researcher's time and his code could get allocated to a product group if anyone found their work interesting. Researchers at Microsoft wanted to ship their code and many PhDs joined Microsoft because they knew their work had the potential to become widely used.<sup>3</sup>

In the future, when we get a complete set of GPL codebases, it will get interesting very fast because researchers will realize that the most popular free codebase is also the best one for their research.

## 5. New Sources of Revenue

Firefox made \$52 million in 2005 by sharing ad revenue with Google. Firefox made this money when their browser was configured to use the Google search service. Free programs can receive a cut of the business they create for a service provider. Your free photo management tool could charge for the right to be the default printing service of their application. Widespread use of free software will create new opportunities to extract value.

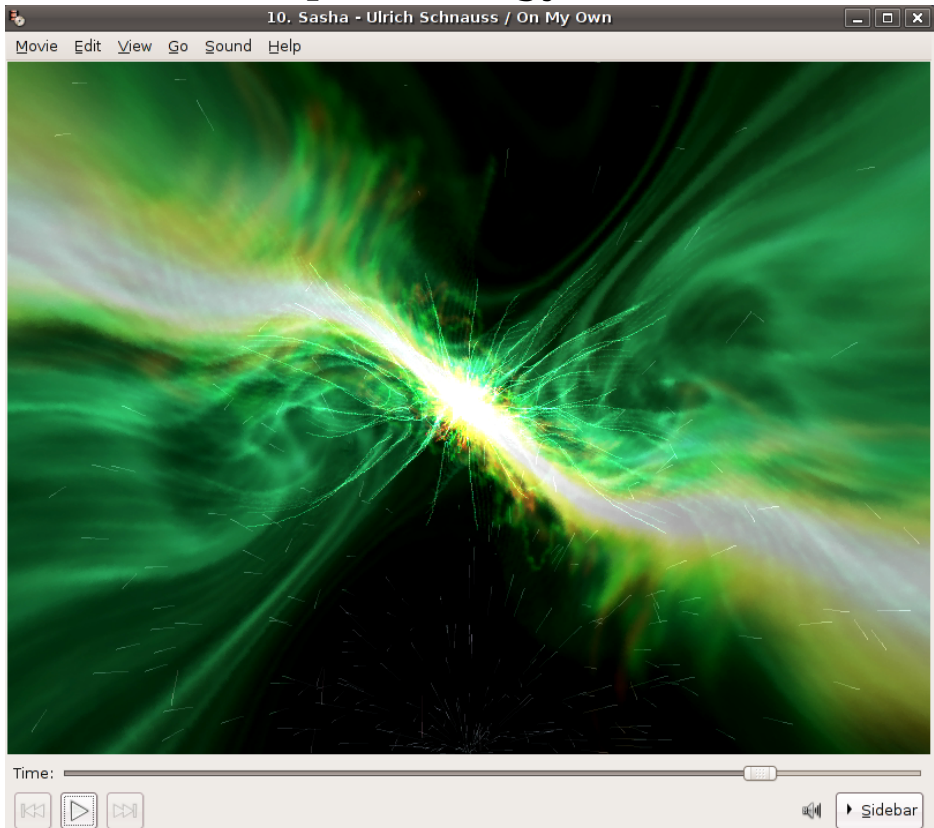
## 6. Fame

The self-satisfaction and adulation that people receive from producing things that others use and enjoy should not be underestimated, even in the world of software. It was a common feeling among my peers at Microsoft that we should pinch ourselves because we were getting paid to write code that millions of people used. It is almost as much fun to be appreciated in the software world as it is in other endeavors. I once played music for 200 people and I didn't even care that I wasn't getting paid when a girl who liked my music put my business card in her bra.

---

<sup>3</sup> Ideally, researchers would do work directly in a product group's codebase. Unfortunately, too many product group codebases were so big, old, and complicated that researchers typically couldn't work in them directly.

## 7. Man's Surplus Energy



*"Goom is the only software project I know admired by myself, the wife, my three-year-old son, and the mother-in-law." — Dave Prince*

The Linux kernel and the rest of the free software stack have spots of brilliance in many places, but not many you can visualize. My epiphany that Linux was going to win on the desktop happened when I cranked up the default Linux media player. Totem doesn't contain a web browser or try to sell you anything, and it plays a very wide variety of formats, but what impressed me was its elegant visualization engine, not-so-elegantly named Goom.

Goom is visual proof of the proposition that with the free flow of ideas, a clever engineer from Toulouse who you've never heard of, was never hired by anyone, never signed a non-disclosure agreement was able to write some beautiful code that now graces millions of computers. For all we know, he did this work in his pajamas —

like the bloggers who took out Dan Rather in the Ra<sup>th</sup>ergate<sup>4</sup> scandal. If you want to read the code to learn its secrets, post a question in a support forum, or send a donation to say thanks, the software repository SourceForge enables this, and Goom is one of its 100,000 projects.

No one edits Wikipedia for fame or swooning girls. But this energy spent is an example of the surplus intelligence of millions of people that can be harnessed and put to work for interesting things. The surplus intelligence of computer scientists is enough to write all of the software we need. Programmers employed by businesses are just icing on the cake, or to answer the phone at 3AM when a computer is sick.

## 8. Increased Demand for Content

Content is king. In the digital world, content is encoded in digital format. If that format is open, and manipulated by free software, then the format can spread to every device and potential supporter of the arts. Every free software tool that manipulates a format can become a content platform. We think of music and movies as content platforms, but they are just the final outputs of two pieces of software. Musicians could start selling the data files they use to make their songs. This would allow people to enjoy the music of their favorite musicians, and re-mix it in new ways.

I would like to see a huge selection of document templates for OpenOffice, the free competitor to MS Office. Some of the template creators could charge, or even request donations. Nothing about a software license agreement says that the content you produce using it must also be without cost. Free software doesn't concern itself at all with the cost of things built with the software. We might develop a world with lots of totally free content, but that is a completely different set of choices we would make.

If we have to choose between free software and free content, free software is a much better choice because it will allow us to understand each other's content, which is the first step towards collaboration. A world full of free software is a world full of many different forms of content.

---

4 The Rathergate scandal was sometimes written that way because the documents that Dan Rather broadcast, which were supposedly from the 1970s, had centered text, proportional fonts, and the letters "187<sup>th</sup>" written with superscript. Typewriters did not have superscript back then, so they were clearly forged!

# Should all Ideas be Free?

Science's complexity forces people to work together. Many therefore wonder if the success of Wikipedia and YouTube suggest that all forms of intellectual property, such as books and music, should also be free. This will be just one of the interesting economic questions for us to ponder in the early 21<sup>st</sup> century. As someone writing a book, this topic became very relevant to my wallet.

Fortunately, I have concluded that there is an important distinction between *static* and *dynamic* intellectual property. A song, once created, is not typically edited by a user. Software is often edited by a user because it often won't work until it is fixed to support an individual scenario.

Linus created Linux, but thousands of other people have put their stamp on it over the last 15 years, at a pace that is only increasing, improving on his humble creation to make it work better for themselves. Books, music and movies are often made by one person, or small teams of people, and are meant to be used as-is without the need for modification. (There may be a need to translate or re-dub a movie or song into other languages, but this doesn't require the same creative intellectual energy, and isn't something that requires further improvements over time.)

Writing has been a way to make a living for many hundreds of years, and I am not ready to propose that we tweak society in such a fundamental way and remove longstanding incentives to create non-science. There is no such thing as a "proprietary book", as by definition you can see what it contains and build on top of the ideas in it.

Richard Stallman talks about a world where the music is free, but music players have a button where you can click to send \$1 directly to the artist when you hear a song that you like. Such a world could provide a sufficient incentive to create free music, and it could be the choice of artists to create such an arrangement. We can create arbitrary rules, so let's experiment!

## Video Games and New Media

While books, music and movies are mostly static creations, video games are a mix of software and static multimedia. Video games contain the game engine that models the virtual world, but also contains 3-D models, music, a storyline, which are meant to be enjoyed as-is. Therefore, video games should continue to be sold for money to pay for the art, but the game engines could be GPL.

Making game engines free will allow for much more user-created content. iD Software, creators of the popular game Doom, did release their game engine, though it is a very old version of their code which they no longer use. Even so, a free software community has surrounded it! This version of Doom has never been used in any Xbox 360 game, but you can run it on an iPod.

Game creators today keep their game engine private because they consider it a competitive advantage — but it is also a huge part of their costs. The Halo series of video games, the most popular on the Xbox platform, spends three years between releases. A big part of the holdup is the software engine development.

A game engine is a piece of technology comparable in size to the Linux kernel because developers need lots of features:

- the ability to model a world, create the rules of the game, and efficiently send updates between computers
- text to speech, means to deal with cheaters, and other forms of AI.

If the thousands of game programmers around the world started working together on the software, I shudder to think what they might build! Hollywood might increase their investments in the game business if there were a free standard codebase: movies, video games and virtual reality are closely-related technologies.

## Pride of Ownership

While much free software is not paid for or organized by a company, the quality of it is just as good because the programmers do so with the same pride as they do a day job.

When I joined Microsoft, I casually signed away the right to use my work anywhere else. Despite the fact that I no longer owned my work, and would never see the code again after I left the company, I didn't treat it with any less importance. It is in the nature of man to do his best whether or not he exclusively owns an idea.

In fact, unpaid volunteer programmers might be more tempted to do it right because there *isn't* a deadline. A lot of lesser-known free software in use today was started purely for the enjoyment of the programmer. Linus wrote Linux to learn about his computer, and even today, [25% of contributors](#) to the kernel do not work for a company.

A programmer might have a boring job building websites — something that pays the bills, but writing fun code in his free time might

provide more of a personal reward. There is a free physics engine used in video games called Open Dynamic Engine maintained by a PhD in his free time, while he works for Google during the day. Gcompris is a free educational program created by Bruno Coudoin for his two children.<sup>5</sup> He didn't need to make it perfect, because other programmers who recently became fathers showed up to help!

In proprietary software, the pride motivation to do the right thing is actually one step removed because you write software in the hope that you can ship it and sell it. And that urge to get it out there may encourage you to cut corners.

In a free software environment, you write code because there is a need – an itch to scratch. This encourages you not to overbuild something that is of no use to you like the Microsoft Office paperclip character.

## Where Does Vision Fit In?

Ideas are somewhat like babies – they are born small, immature, and shapeless. They are promise rather than fulfillment. In the innovative company executives do not say, “This is a damn-fool idea.” Instead they ask, “What would be needed to make this embryonic, half-baked, foolish idea into something that makes sense, that is an opportunity for us?”

—Peter Drucker

In a world of free software where there is no strong leadership from the top, where does vision fit in? The answer is that there is vision in the big, and vision in the small. Vision in the big is when someone has an idea for a new project, describes this vision to others, creates a proof of concept, and an environment that encourages people to join the project. However, these new, big efforts like Wikipedia and the Linux kernel are very infrequently launched.

In actuality, on a daily basis, team members struggle to refine a product to make it better achieve that original vision. This refining activity requires lots of little work items with each step requiring a tiny bit of visionary insight moving the product forward in support of that big idea. In creating free products that anyone can contribute to, progress toward big visionary efforts will happen faster.

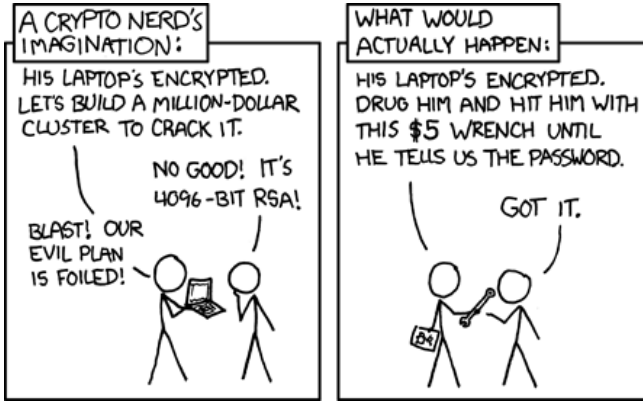
Note, free software doesn't imply that obstacles like ego and stubbornness will disappear, but it is also no different than what exists everywhere else.

---

<sup>5</sup> I did check to verify that Bruno had children, but like changing diapers, voluntarily writing free educational software for children is something only a parent would do!



# Governments and Free Software



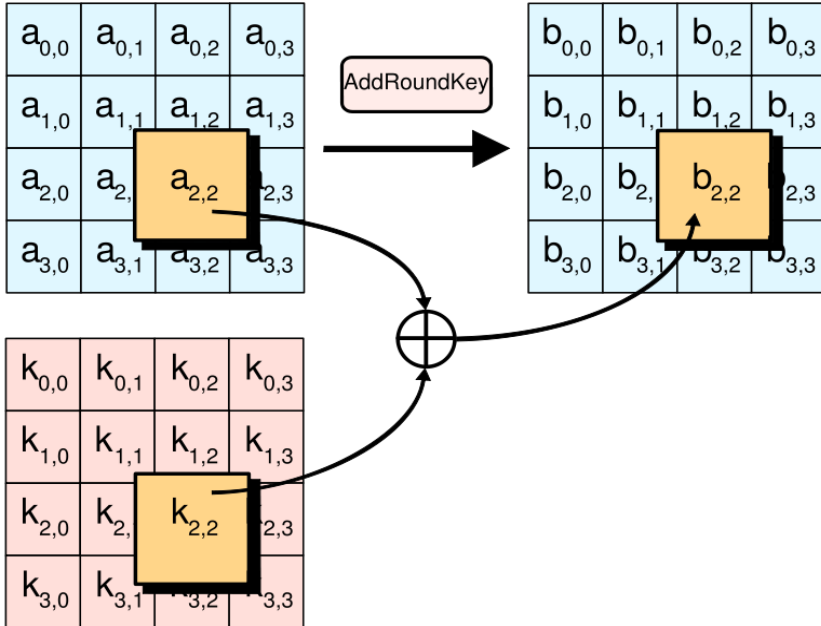
Comic from [xkcd.com](http://xkcd.com)

Documents produced as official duties of an employee of the US government are assumed to be in the public domain, unless classified. While creation of free software for profit-seeking corporations might still be up for debate, governments and other publicly funded institutions can embrace free software.

That works sponsored by a government should be placed in the public domain reminds us of their public nature. Copyleft is the 21<sup>st</sup> century equivalent of public domain. It is in the spirit of public domain, but provides better protection to the intellectual property.

The U. S. government is starting to understand this: NASA, the Department of Defense, and the NSA are making steady movement towards free software, which should also demonstrate to everyone that free software can meet even the most extreme requirements of security and reliability.

Why is free software secure? A common principle in software is that “security through obscurity” is a bad idea. For example, the most important security encryption algorithms today are designed and analyzed in public and have free software implementations:



*Math applied during encryption of data. The security of encryption algorithms is guaranteed by the many smart eyeballs who have analyzed it. Ultimately, if the mathematicians prove an encryption algorithm is secure, there are no back doors, and a password is the only key to the data.*

Would the GPL require that the U.S. government give away its top secret code? The GPL's goal is to ensure that all users of software have the right to inspect and make changes. The user of military software is the military itself, so the conditions are met.

The U.S. government might not want to give away certain source code to other countries, but this is also solvable. Since enforcement of copyright is the right granted to the U.S. Congress, they can create a law that says any GPL software stamped “Top Secret” is removed from copyleft obligations, something that should apply only to a small amount of code.

A question to ponder: If the military were to create a vision system to help it build more discriminate bombs less likely to hit civilians, would the military be inclined to give this away or would they be afraid that the enemy would use it to build more discriminate

IEDs? Software has so many different applications and purposes, it will be quite difficult for the government to make the correct trade-offs.

Because GPL should be usable by all of the U.S. government's code, I would like to see them endorse it for all publicly-funded software development. This would also help to ensure that the U.S. is not left behind when free software achieves world domination.

## Should all Software be GPL?

The era of procrastination, of half-measures, of soothing and baffling expedients, of delays, is coming to a close. In its place we are entering a period of consequences.

—Winston Churchill

I don't envision a world where all software need be free, but I do believe that all the world's large, interesting software projects require a collaborative effort and should therefore be built by a community and protected by a copyleft license.

There is a lot of generic mass in the lower layers of software. Eventually as you move up the stack, you reach areas that are specific to a particular business. At this point the importance of cooperation decreases, but it is also something which isn't even useful to competitors. In other words, whether in the lower or upper layers, all software may as well be free.

It is still the early days in the coming gold rush era of free software, and those who understand this fact first will gain all the users, and get all the help. While the Linux has locked in its spot, most of the rest of the free software stack isn't in this position yet, although many, like Firefox and OpenOffice, are close.

One other relevant point is that, the sooner the free software community is brought in, the better off the codebase will be. There are a number of products that started out non-free and were only given to the community much later. By this time, one usually finds that the codebase is a complicated mess, and it takes a while for the community to understand it and clean things up. Mozilla's Firefox and IBM's Eclipse Java IDE are examples of products that took a while to take off after being liberated.

# Microsoft's Responses to Free Software

## 1. Windows + Office = \$3

It's easier for our software to compete with Linux when there's piracy than when there's not.

—Bill Gates

One of Microsoft's responses to free software has been to lower prices. Microsoft has created student and community editions of certain products that were low in cost but also often crippled. Microsoft also announced a bundle, available to qualifying third-world countries, that includes a low-end Windows, a low-end Office, and a few other items for just \$3. There are fewer than one billion people with PCs, and so presumably a cheap bundle will make it easier for the remaining five billion people to get PCs. For potential customers there are several things wrong with this approach:

1. A Linux OS comes with a much larger set of applications than is included in this bundle.
2. The source code is not available, so engineers in these emerging markets cannot build up a service industry of companies who support and customize this software.
3. Customers become locked into Microsoft file formats and other proprietary technologies, and the countries become dependent on Microsoft for future progress.
4. The prices of the rest of Microsoft's software, from servers to development tools, have not been lowered. This three-dollar bundle only gives users access to a small fraction of Microsoft's technologies.

Of course the \$3 bundle is a very smart defensive move by Microsoft. It is charitable of them to lower their price, however, the move might be considered “dumping,” which Wikipedia defines as an “act of a manufacturer in one country exporting a product to another country at a price that is either below the price it charges in its home market or is below its costs of production.”

The low-cost bundle is a danger to the free software movement. I can think of nothing better than having millions of programmers around the world build their industrial base around a free license. Emerging markets should start off their economies on the right foot with software that's completely free.

If it weren't for piracy, Linux would have likely taken over the world already. I have been told that more than 90% of users in China run pirated software, and as no one can truly know, it could very well be closer to 99%. If it weren't for these illegal copies, China would have been forced to develop a suitable free software stack. And once something existed for a country such as China, it would have been usable for the rest of the world as well.

The United States needs to move rapidly towards free software so it can be relevant in building the future. The United States invented the transistor, but it has also spawned a ton of old, proprietary systems that are a drag on future productivity. America is the widespread purveyor of non-free software which means the transition to free software will be much more difficult than for other countries which don't have this baggage.

## 2. CodePlex

Microsoft has also created a number of websites where developers can use free code and collaborate, and the latest is called CodePlex. While it does demonstrate that Microsoft understands the benefits of free software, this website mostly contains tiny add-ons to proprietary Microsoft products. CodePlex may serve mostly to kill off Microsoft's community of partners who previously sold add-ons to Visual Basic and other products. While these serve as a bulwark against the free software movement and provide a way for Microsoft to claim that they get this new way of developing software, it ultimately undermines their business.

## 3. Interop

Another way Microsoft has responded to requests for increased open-ness is by publishing specifications for certain file formats and protocols. While a spec is a document, and a promise not to sue, it still requires code, or reverse-engineering in order to put this idea to use on a computer. A spec attached to proprietary software is a spec with a huge cost attached to it: the cost to build the free baseline implementation.

Someone who wants to improve the C# programming language doesn't want to start with the spec, but with a compiler for that language. In fact, the spec could be generated from the compiler's source code.

Likewise, the best way to make sure that a new audio format becomes available on every device is to make freely available the code to read and write it. If the details are public, why not make them useful to a computer?

## 4. Shared Source

Microsoft has also released some software under various “shared source” licenses. One of the first products Microsoft released under this license was C# and a portion of the .Net runtime. The language spec was always free, and there was a decision made to release some of the code as well in 2002. In addition to releasing the code, Microsoft seeded research efforts by sponsoring 80 projects in universities around the world. However, there is little activity today, and one reason is that the license is very restrictive:

You may not use or distribute this Software or any derivative works in any form for commercial purposes. Examples of commercial purposes would be *running business operations*, licensing, leasing, or selling the Software, or distributing the Software for use with commercial products.

—Microsoft Shared Source CLI License

*This is a license that shuns capitalism.*

There is a free project managed by Novell called Mono which has reverse-engineered the C# runtime and libraries from the public spec. It has gained interest in a number of communities, and this runtime, not Microsoft's, provides a way to run C# code on the Mac and Linux. This free implementation could eventually become the de facto one!

## Just a Stab

Richard Stallman, who started the free software movement in 1985, might be right when he says that the freer intellectual property is, the better off society is.

However, I don't think this should automatically apply to everything, even software. Furthermore, it is a choice for every creator to make. While it might make economic and moral sense for some ideas to be given away, that doesn't mean ideas are no longer owned. The GPL says that software is owned by its users.

Stallman reminds us that the concept of a free market is an idea that has taken a long time for the general public to understand and free software and the other intellectual property issues we grapple with today will also take time for society to grasp.

Computer pundit Tim O'Reilly makes the point that the GPL could become irrelevant in the coming cloud computing world. Nowadays, people focus on free code running on their own computer, but what about if you are using GPL code which is doing work on your behalf, but running on another processor? Currently, the GPL does not consider this scenario, but I think this is a loophole not within the spirit of copyleft. Perhaps this will be the battle for GPL v4, some years hence.

# PATENTS & COPYRIGHT

It has always been a strong goal of the JPEG committee that its standards should be implementable in their baseline form without payment of royalty and license fees. The up and coming JPEG 2000 standard has been prepared along these lines, and agreement reached with over 20 large organizations holding many patents in this area to allow use of their intellectual property in connection with the standard without payment of license fees or royalties.

—Statement by JPEG committee

If people had understood how patents would be granted when most of today's ideas were invented, and had taken out patents, the industry would be at a complete standstill today.

The solution is patenting as much as we can. A future startup with no patents of its own will be forced to pay whatever price the giants choose to impose. That price might be high. Established companies have an interest in excluding future competitors.

—Bill Gates, internal memo, 1991

**I**n February 2007, Microsoft lost a \$1.5 billion judgment for infringing on MP3 patents, even though Microsoft had licensed MP3! The problem with MP3 is that multiple companies have patent claims, which makes the situation a particular mess. We can argue as to whether patents add value, but they indisputably add costs.

The Software Freedom Law Center calculates that Microsoft has paid \$4 billion in lawsuits in the last three years to companies who have won patent claims against the company, which works out to a tax of \$20 per copy of Windows. (The costs are actually higher because some of the settlements are secret.)

As a Microsoft employee, if they filed a patent on technology you devised, you received a three-inch marble keepsake cube and a few hundred dollars. I remember interviewing for a new position within Microsoft and feeling my hands sweat when I saw a stack of cubes behind my interrogator. I received only two patents, late in my career, so I always felt a tinge of jealousy when I saw someone else's patents. On the other hand, I considered myself merely unlucky that the features I was assigned happened to be undeserving of a patent.

My friend Alex Mogilevsky added background spell checking to Microsoft Word '95. which draws red squiggly underlines below misspelled words. This is a feature we are all very familiar with now but



which was the biggest enhancement of that release and some say this was the most useful feature ever added to Word. In the end, Alex received U. S. patent #5,787,451, but was this feature truly worthy of a patent? These are the major elements of this patent:

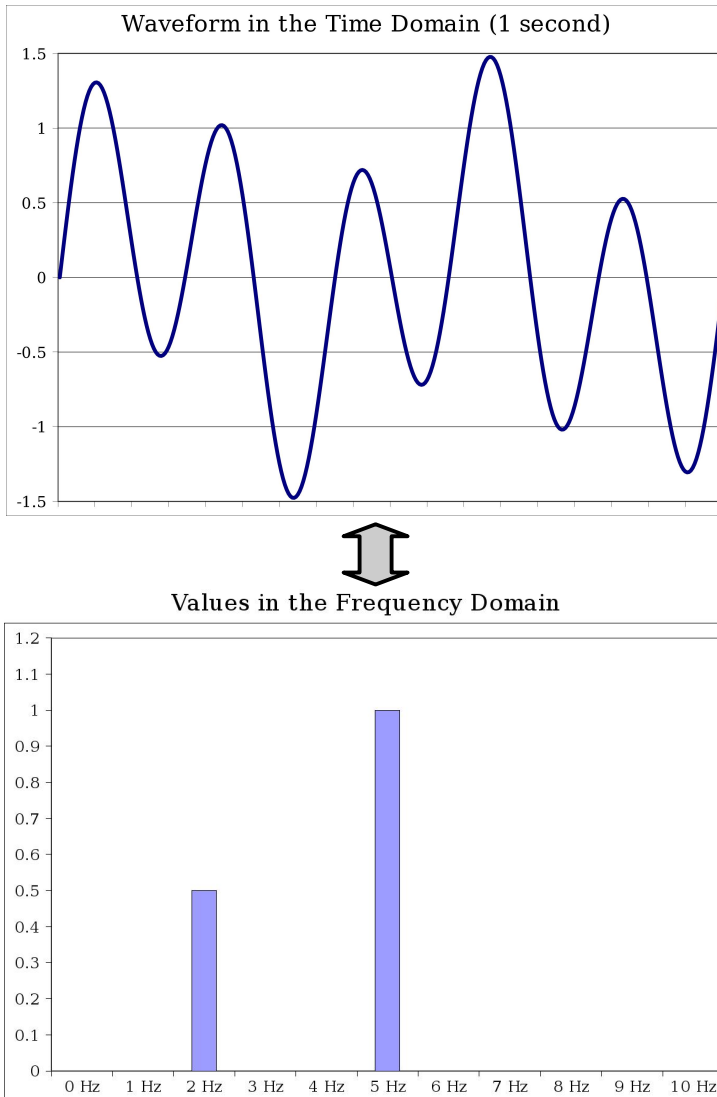
- Red underlines of misspelled words
- Spell checking happens as you type, removing the need to launch a dialog box as a separate step.

While adding this feature was a huge time-saving device, it isn't something so unique that other word processors wouldn't have eventually implemented it. Teachers have been circling misspelled words with red pens since time immemorial, this is just the digital version. Fortunately for the world, Microsoft has not enforced this patent and squiggly underlines can now show up almost everywhere you can type text.

One of Amazon's first patents was for "1-Click ordering." Once Amazon has your payment and shipping information on file, you are able to purchase a book with literally one click. However, isn't this an obvious innovation for anyone building an e-commerce website? *Zero-click* ordering would be an innovation worth patenting!

Amazon's patent didn't encourage innovation, it simply became a stick their lawyers could use to beat up Barnes & Noble. We are told that patents protect the little guy, but they actually create a complicated minefield that helps incumbents.

One of the biggest areas of patent headaches for the computer industry today deals with codecs (*compression – decompression*), little pieces of software that compress and decompress sound and images. Patenting codecs is a bad idea because the fundamentals of all codecs are the same:



*There are an infinite number of ways of converting sound to and from bits, but they are mathematically very similar. (The difference between codecs has to do with merely their efficiency, and their cleverness in removing data you cannot perceive.)*

There might be a new type of compression algorithm that is innovative, but the point of codecs is to enable the easy exchange of video and audio bits. Patents, therefore, only serve as a hindrance to this. The reason why digital audio and video is such a hassle today is because of the mess of proprietary formats, patents and licensing fees. These obstacles encourage the creation of *even more* formats, which just makes the problem worse.

In the mid-90s, Apple, Microsoft, Real, and others were out there hawking their proprietary audio and video formats, touting their advantages over the others. We have not recovered from this. Microsoft employee Ben Waggoner wrote:

Microsoft (well before my time) went down the codec standard route before with MPEG-4 part 2, which turns out to be a profound disappointment across the industry — it didn't offer that much of a compression advantage over MPEG-2, and the protracted license agreement discussions scared off a lot of adoption. I was involved in many digital media projects that wouldn't even touch MPEG-4 in the late '90s to early '00s because there was going to be a 'content fee' that hadn't been fully defined yet.

And even when they created standards like MPEG, certain companies would assert patent control over certain aspects. MPEG isn't a codec so much as a system of codecs, a land mine of proprietary and non-proprietary specifications that makes “supporting” MPEG very difficult. The reason many websites do their video using the proprietary Flash control is because the various interests didn't come together to produce a standard.

Many times in this industry, someone has invented a compression mechanism, patented it, implemented the code for their own use, but did not document it or give away code to encode and decode the format. Then they asked everyone to use their new format. This strategy is totally the wrong approach to making formats universally usable by computers and devices.

What is important is that we pick a simple and efficient algorithm, standardize it and then make the software to read and write it freely available. That way, every device and every application will work with every piece of sound or video. Today, there is nothing but chaos and incompatibility.

The most popular audio format today is MP3. Here there is not just one, but a number of different companies that have patent claims that do not expire until 2015! The core logic of a codec is

only a few thousand lines of software; this entire mess is over details too small to disagree over, yet somehow we do.<sup>1</sup> Patents and standards serve diametrically opposite purposes.

## Software is math

Software is math. In the 1930s, Alonzo Church created a mathematical system known as lambda ( $\lambda$ ) calculus, an early programming language that used math as its foundation, and which could express any program written today.

A patent on software is therefore a patent on math, something that historically has not been patentable. Donald Knuth, one of America's most preeminent computer scientists, [wrote](#) in a letter to the U. S. Patent Office in 2003:

I am told that the courts are trying to make a distinction between mathematical algorithms and non mathematical algorithms. To a computer scientist, this makes no sense, because every algorithm is as mathematical as anything could be. An algorithm is an abstract concept unrelated to the physical laws of the universe.

Nor is it possible to distinguish between “numerical” and “non-numerical” algorithms, as if numbers were somehow different from other kinds of precise information. All data are numbers, and all numbers are data.

Congress wisely decided long ago that mathematical things cannot be patented. Surely nobody could apply mathematics if it were necessary to pay a license fee whenever the theorem of Pythagoras is employed. The basic algorithmic ideas that people are now rushing to patent are so fundamental, the result threatens to be like what would happen if we allowed authors to have patents on individual words and concepts.

I strongly believe that the recent trend to patenting algorithms is of benefit only to a very small number of attorneys and inventors, while it is seriously harmful to the vast majority of people who want to do useful things with computers.

Software doesn't look like math, but it is built up from just a few primitive operations that have a mathematical foundation. Allowing people to patent particular algorithms just means that a section of our math is now owned by someone. If these patents are owned by many different entities, then math could become unusable by anyone. Then where would we be?

---

<sup>1</sup> The free Vorbis decoder for embedded devices is less than 7,000 lines of code, and much of that code is infrastructure logic to do, for example, all math using integers because low-end processors often do not have floating point capabilities.

Ironically, when defending itself in a patent infringement case with AT&T, Microsoft argued that software cannot be patented:

Justice Scalia: "You can't patent, you know, on-off, on-off code in the abstract, can you?"

Microsoft attorney, Ted Olson: "That's correct, Justice Scalia. [...] An idea or a principle, two plus two equals four, can't be patented."

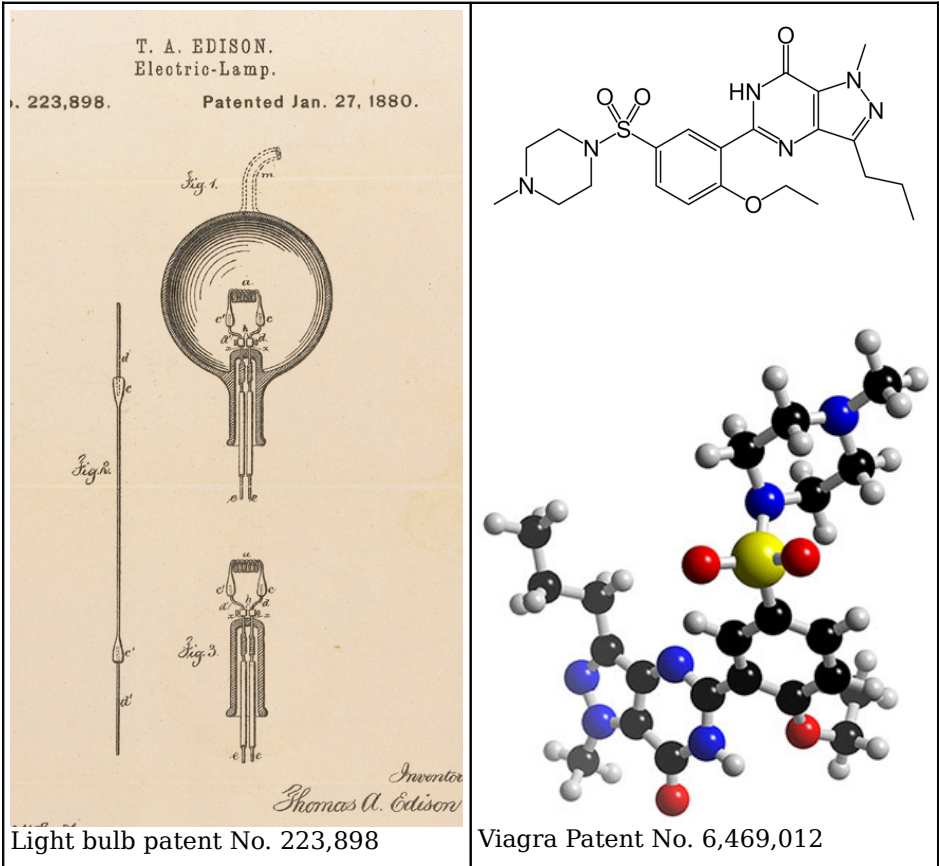
The Supreme Court ruled that "unincorporated software, because it is intangible information, cannot be typed a 'component' of an invention." One might read this and conclude that the Supreme Court, unknowingly, and at Microsoft's behest, outlawed software patents!<sup>2</sup>

---

<sup>2</sup> Justice Alito, in a concurring opinion states: "To be sure, if these computers could not run Windows without inserting and keeping a CD-ROM in the appropriate drive, then the CD-ROMs might be components of the computer. But that is not the case here." Actually, Linux Live CDs allow you to run the OS directly off of the hard drive. I don't see a distinction between using a CD-ROM every time, or using it once and copying it to a faster medium and then discarding it. Reading the decision made my head hurt.

# Software is big

Beyond software being math, software also differs from things that were patented previously. In the biotechnology world, and even the world of Edison, a patent typically covers one product:



*Usually, a patent is one product.*

Software, however, is different because it is enormous and incorporates thousands of differing ideas. Software products today are not patented in their entirety, only tiny portions of them. Richard Stallman compares software to music: imagine if people could patent chords, how would that affect our ability to create new music?

Jerry Baker, Senior VP of Oracle wrote:

Our engineers and patent counsel have advised me that it may be virtually impossible to develop a complicated software product today without infringing numerous broad existing patents.

Microsoft has recently [claimed](#) that Linux violates 235 of its patents spread throughout the software stack. This is an enormous number; Verizon has threatened to put Vonage out of business for violating just three obvious patents:

1. Internet to phone-system connection technology
2. Internet phone features such as voice mail and call-waiting
3. Wireless to Internet phone calls

## Software is a fast-moving industry

In the drug industry, it takes years of clinical trials to prove a medicine. The Tufts Center For the Study of Drug Development reported recently that the average time to get approvals for drugs was 6 years. The current patent length of 17 years is an eternity for the fast-moving field of computing. In his letter to the Patent Office, software scientist Knuth also wrote:

Software patents will have the effect of freezing progress at essentially its current level. If present trends continue, the only recourse available to the majority of America's brilliant software developers will be to give up software or to emigrate.

Therefore, even if you believe in software patents, shrinking their length of exclusive ownership to just a few years would be a compromise. Even decreasing the duration of protection would also decrease the number of spurious patents, which would make it easier to be in compliance.

## Copyright provides sufficient protection

Patents are a powerful right because they give their owners exclusive access to an idea. Proving that you invented an idea independently is not a defense. A much less exclusive right is what is allowed in copyright law. Copyright law protects someone from stealing words or code, but if you can prove you came up with it via independent means, you have a sufficient defense.

In the old days of software, Word and WordPerfect kept adding each other's features in order to convert users. They didn't need exclusive access to an idea to be motivated to write software, and patents would have decreased the level of competition.

Giving exclusive access to an idea can encourage people to rest on their laurels, or even be a squatter on an idea, taking money from

anyone who happens to run across it, but not using it for their own purposes. Many times, patents are created merely as a defensive measure against other companies. A company will patent things with the hope it can trip up anyone who might come calling with claims *against* them.

In a world filled with free software, it is the copyleft mechanism, not the patent mechanism, that will provide protection for software. Even proprietary software would not stop improving if software patents disappeared, though their lawyers would scream like stuck pigs.

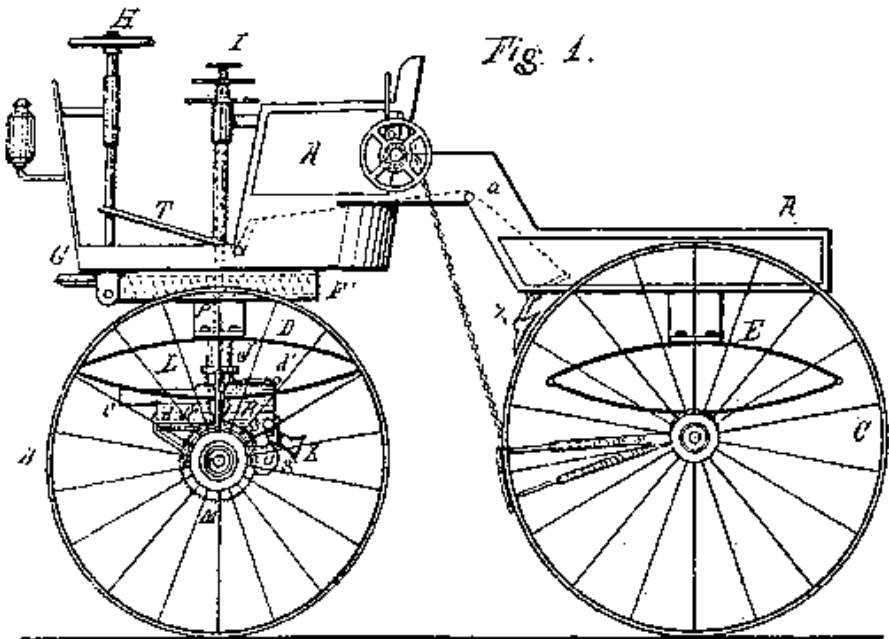
## Conclusion

In the early days of cars, there were patent lawsuits sent between the lawyers, on horseback of course:

G. B. SELDEN.  
ROAD ENGINE.

No. 549,160.

Patented Nov. 5, 1895.



*George Selden, the man who patented the car in 1895, didn't sell one until 14 years later. Was he just a squatter on this idea? The magic of the early car was the internal combustion engine, which Selden did not invent, and which required only the science of fire, something man exploited long before he understood it.*



I believe research will show that at least 99% of software patents today are natural extensions of an existing idea. Even for the few truly unique ideas, it is probably better that the concept be widely available and used by many rather than guarded and stifled by a few.

If someone invents the software equivalent of an idea like  $E=MC^2$ , do we really want just one entity to “own” that idea? Would Einstein want that? Anyone who supports software patents should hold up a good one and explain how it is truly unique and yet an idea that only one company should own.

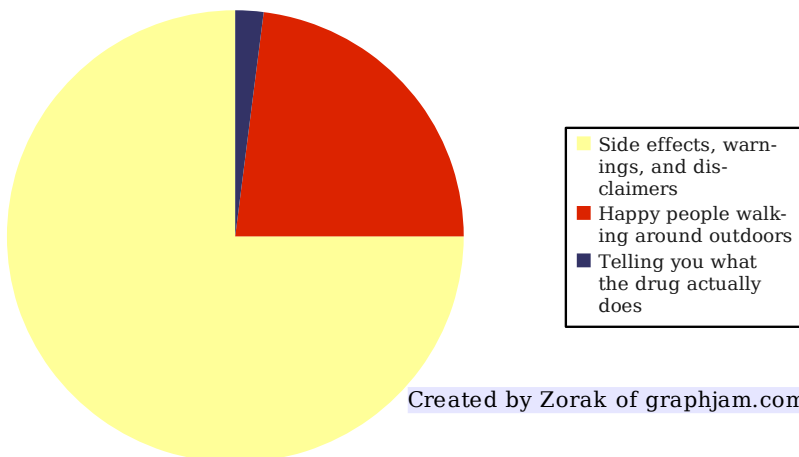
If we outlawed software patents, the pace of progress in software would increase, the squatters and their lawyers would disappear, legal uncertainties surrounding software technology would decrease, and there would still be many other motivations to write software. In fact, many of the hassles in computing, like playing MP3s and DVDs, exist because of patent issues. As of 2005, the U.S. Patent and Trademark Office has [granted](#) 265,000 software patents, a very large minefield.

## Biotechnology Patents

There are more than 50 proteins possibly involved in cancer that the company is not working on because the patent holders either would not allow it or were demanding unreasonable royalties.

—Peter Ringrose, Chief Scientific Officer, Bristol-Myers Squibb

### Pharmaceutical TV Ads



Created by Zorak of [graphjam.com](#)

*Chart of the content in pharmaceutical TV ads*

Although this book is about software, it is worthwhile to look at patents for biotechnology. Michael Crichton recently [argued](#) that because companies have been able to patent genes, that are not even invented by man, biotechnology research is threatened. Patenting drugs is one thing; patenting genes is something entirely different.

*Against Intellectual Monopoly*, by Michele Boldrin and David K. Levine, is an interesting book that discusses many patent and copyright issues. It describes the case for removing patent protection for pharmaceutical companies that produce relatively simple chemicals requiring extensive research and testing. This is a brief summary of their arguments against patent protection for drugs:

- Before the pharmaceutical industry came along, the most important chemical industry was for paints and dyes. The authors' study of the history of those companies shows that companies in countries with patent protection had slower rates of innovation. One of the biggest reasons is because "the chemical industry is a classic case of innovation chains – new compounds and processes built on the knowledge of existing ones." Patents, therefore, get in the way of this.
- Countries such as Switzerland and Italy, which for many years didn't have patent protection on pharmaceuticals, still had robust industries. Adding patent protection has actually hurt investments in those countries' pharmaceutical industries.
- Pharmaceutical companies spend 13% of revenues on R&D and spend twice as much on marketing and promotion. The cost of R&D is not such a big deal as the drug companies say it is.
- Two-thirds of the money spent on pharmaceutical research in the United States comes from the government. This public money is getting turned into private patents.
- Many drugs are inexpensive to produce, but drug companies do not make them available to poor countries. If they did, they could be re-imported, hurting their profits in the higher-priced Western markets. Furthermore, the fact that there are such large profits to be made in Western markets has discouraged the development and production of drugs to treat diseases plaguing the poor countries of Africa and Latin America.

- A study by the U. S. Centers for Disease Control and Prevention (CDC) found that of the top ten public health achievements of the 20<sup>th</sup> century in the United States, none was motivated by a desire to patent.
- The current patent system perverts pharmaceutical companies and makes them focus on copycat drugs. According to an article from *The New Republic*:

Turn on your television and note which drugs are being marketed most aggressively. Ads for Celebrex may imply that it will enable arthritics to jump rope, but the drug actually relieves pain no better than basic ibuprofen; its principal supposed benefit is causing fewer ulcers, but the FDA recently rejected even that claim. Clarinex is a differently packaged version of Claritin, which is of questionable efficacy in the first place and is sold over the counter abroad for vastly less. Promoted as though it must be some sort of elixir, the ubiquitous “purple pill,” Nexium, is essentially AstraZeneca's old heartburn drug Prilosec with a minor chemical twist that allowed the company to extend its patent.

- Money spent on research and marketing of these me-too drugs is money not being efficiently allocated. From Boldrin and Levine:  
Monopolies innovate as little as possible and only when forced to; in general they would rather spend time seeking rents via political protection, while trying to sell, at a high price, their old refurbished products to powerless consumers, via massive doses of advertising.
- The largest cost of making new drugs is clinical trials. In some cases this can cost \$800 million, and in others \$6.5 million. Many of the clinical trials are for me-too drugs, which are a waste. Furthermore, companies paying for clinical trials have a conflict of interest. Boldrin and Levine argue that clinical trials should be paid for by competitive NIH grants. The information about the effects of a drug is a public good and therefore should be paid for by the public.
- There are three stages of clinical trials, and so the authors suggest drug companies pay for stage I, but that public money be used for stages II and III. Taking this cost away would allow the drug companies to focus more on fundamental R&D. The biotechnology firms still have a long way to go in their understanding of chemistry and DNA.

Boldrin and Levine don't necessarily suggest that patents be eliminated; even shortening the period of protection would decrease the

adverse affects of the above issues. Furthermore, even if we assume that lowering or removing patent protection will hurt investments, the government could give R&D tax credits to offset the decrease. Finally, they argue that too many drugs are available by prescription only, which increases costs and lowers revenues for the pharmaceutical companies.

Congress, in mid-2007, was considering some patent reforms. Some of the changes included changing the patent system to grant them to the first one to file, not the first one to invent, and to limit the amount of infringement damages. Both of these ideas are bad, however, and are just nibbling around the edges. Boldrin and Levine argue that the following reforms should be considered:

- Proof that you independently invented something should be a defense against a patent infringement.
- Licensing fees for patents should be set close to R&D costs.
- Have the government fund clinical trials.
- Reduce the number of drugs requiring a prescription.
- Decrease the term of patents to four years.
- Suspend drug patents in poor countries. To prevent drug re-importation, suspend free trade for medicines.

Boldrin and Levine argue that rather than allowing for the creation of patents, subsidizing a portion of R&D via tax credits is a more efficient allocation of public resources.

## Openness in Health Care

Throughout this book, I advocate cooperation in the future development of software. While doing research for this book, I came across a report by the Committee for Economic Development (CED), an independent, non-profit, non-partisan think tank that wrote a report entitled: “[Harnessing Openness to Transform American Health Care](#)”. According to the report:

\$2 Trillion dollars, 16-17% of our GDP, is spent on health care overuse, underuse, misuse, duplication, system failure, unnecessary repetition, poor communications and inefficiency. The U.S. employer-based health care system is failing. Health care costs are rising faster than wages, quality of care is low, and access to coverage is deteriorating.

The report details ways in which better cooperation can improve healthcare. For instance: The search for the human genome sequence was a competitive race. Celera, a private-sector firm,

sought to be first to establish the sequence while, as was the norm, keeping much of its data private, to be made available on commercial terms to other researchers.

In contrast, the publicly funded Human Genome Project (HGP) followed an open model making its data publicly available and welcoming input from around the world. HGP pushed participating researchers to disclose their findings as quickly as possible. While Celera made important contributions to the sequencing, it was the HGP's model of discovery that has transformed the research process by reducing "transaction costs and secrecy that may impede follow-on research."

The HGP researchers not only put raw sequencing data into the public domain, but as the data were being produced, an open-source software program known as the distributed annotation system (DAS) was set up to facilitate collaborative improvement and annotation of the genome. This allowed any researcher to choose the annotation they wanted to view and enabled the ranking of annotations by the number of researchers that used them, something akin to Google's methods for ranking search results.

This open model is now being used in a federally funded international effort to create a map of haplotypes (HapMap) which describes variations in the human genome that tend to occur together in "neighborhoods" or haplotypes. Data about the genotype of the individual haplotypes is being released publicly as soon as it is identified. The openness of the HapMap effort is reinforced by its use of a licensing system that is "self-consciously modeled on the copyleft system of open-source software licensing" and which prevents those who utilize the data from attempting to close it to others via patents.

Utilizing the results of the HapMap process, a public-private partnership, the SNP Consortium, is identifying panels of a few hundred thousand single-nucleotide polymorphisms (SNPs) that can be used to identify common variants in an individual's entire 3-billion base-pair genome that might be associated with a disease. As with the HapMap project, participants in the consortium have agreed to put the data they produce in to the public domain.

In the reasonably near future, according to Dr. Francis Collins, leader of the National Human Genome Research Institute (NHGRI) in the National Institutes of Health (NIH), the HapMap should help make practical case-controlled studies using SNP's to identify gene variants that "contribute to diabetes, heart disease, Alzheimer disease, common cancers, mental illness, hypertension, asthma, and a host of other common disorders." That future seems nearer than ever today with scientists finding correlations between diseases such as multiple sclerosis and breast cancer and specific genetic variations.

To harness the power of openness to speed the development of medicines and vaccines for less lucrative commercial markets, the Bill & Melinda Gates Foundation now conditions its grants to require researchers to share their results promptly so that others can build on successes, avoid pitfalls, and eliminate redundancy. It appears that Bill believes in openness for medical research, but not for software yet.

The CED reports concludes:

Openness is ultimately about an attitude that sees the opportunity for many to benefit from greater access to information, as well as to contribute much to the benefit of us all. Greater openness is likely to become increasingly important in more and more areas driven by the relentless progress of information and communications technology. We offer these recommendations with the hope that modest changes based on greater access to information by more people, and more possibilities for them to contribute based on their own expertise and energy, can help improve health care in the United States and around the world.

Currently, a lot of bioinformatics software is proprietary, following the current path of PC software, but that is likewise slowing progress.

## The Scope of Copyright

The Congress shall have Power...To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries

—U. S. Constitution, Section 8

The utility of patents is a big debatable question, but few argue against copyright law. Copyright law is important even for the copy-left license because it allows for enforcement of the terms.

The two biggest issues surrounding copyright law are the duration of the copyright and your ability to do things outside the scope of copyright, i.e. "I'm legally using your copyrighted materials without needing to get your permission."

## Length of Copyright

Our Founding Fathers had it right, once again, when they determined that authors and inventors should have exclusive rights to their creations for "limited times." Their thinking suggests a presumption that ideas will eventually flow into the public domain

because, at some point, increasing the years of protection isn't guaranteed to promote further progress; instead, it may serve as an impediment.

When copyright was first created in the U. S., the term was 14 years, renewable once if the author was still alive. Today, the time frame that has been chosen for U.S. copyright law is the life of the author plus 70 years. This value was extended from the life of the author plus 50 years in 1998. (In *Eldred v. Ashcroft*, the U.S. Supreme Court upheld the constitutionality of this law. The way to fix copyright law is via Congress, not via the courts.)

It is hard to believe that people are motivated to make music so that they can sell it after they are dead, so one could say that this value is not "limited times." Shortening the copyright law perhaps decreases the motivation to create things, but it also allows more people to have access to things they would never have otherwise. We need a new compromise and having a copyright term of 10 years should be the starting point for discussion. (In engineering, back of the envelope estimates are a good basis for analysis. If 10 years is too short, then next try 20, 40 and 80 years.)

Do you still buy 80s music? 90s music? How many of your movies, music and books were less than ten years old when you purchased them? I do not own any Beatles albums and I would never buy one even if money were burning a hole in my pocket. The only way I will hear Beatles music is if I can listen to it for free. I'll bet that most things purchased at Amazon were produced in the last ten years. Amazon could provide useful data to Congress on these points; the writers of the Constitution didn't define copyright length because they didn't have such data available to them.

When the term of copyright expires, it does not mean that no one will sell the newly public domain item anymore. In fact, companies might be willing to make CDs and DVDs of things in the public domain for prices close to the marginal costs, yet still make a profit.

If you set the term of copyright beyond the point of drop off in sales, you will not have materially hurt the artists, but you will have created a richer world; the benefit of a free culture is that the next generation of budding artists will have so much greater access to art and they will become better artists for it.

# Fair Use

The fair use clause of copyright allows you to use copyrighted materials for limited purposes without getting permission. This right was recognized by the courts in common law and was incorporated into U. S. Copyright law in 1976:

The fair use of a copyrighted work, including such use by reproduction in copies or prerecords or by any other means specified by that section, for purposes such as criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship, or research, is not an infringement of copyright. In determining whether the use made of a work in any particular case is a fair use the factors to be considered shall include—

1. the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes;
2. the nature of the copyrighted work;
3. the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and
4. the effect of the use upon the potential market for, or value of, the copyrighted work.

There are questions unanswered here. Am I allowed to: make a mix-CD and give it to ten of my friends? Let people download it via the Internet? Sell it for shipping and handling plus \$1?

The biggest problem with fair use is that it is biased against the commercial domain. Fair use was not meant to take money out of art which reuses other art, but it has had that unintended consequence.<sup>3</sup>

---

3 Creative Commons is a non-profit organization that has created several “CC” licenses. These licenses allow creators to easily communicate which rights they reserve, and which rights they waive for the benefit of users and other creators. A number of the Creative Commons licenses make it easy to use their product, but only in non-commercial enterprises. This anti-capitalist thinking goes against the freedom to use an idea for any purpose and further takes money away from art.



# Digital Rights Management (DRM)

Before the digital world, the enforcement of copyright law was limited by the physical restrictions of the real world. Book authors didn't worry much about people making unauthorized copies of their work because the expense of making an unauthorized copy approached the cost of purchasing a legal copy.

Enforcing arbitrary copyright provisions was not possible. You could not charge someone more who planned to lend out a book than someone who planned on keeping it for themselves, because the cost of enforcement would be greater than the profits.

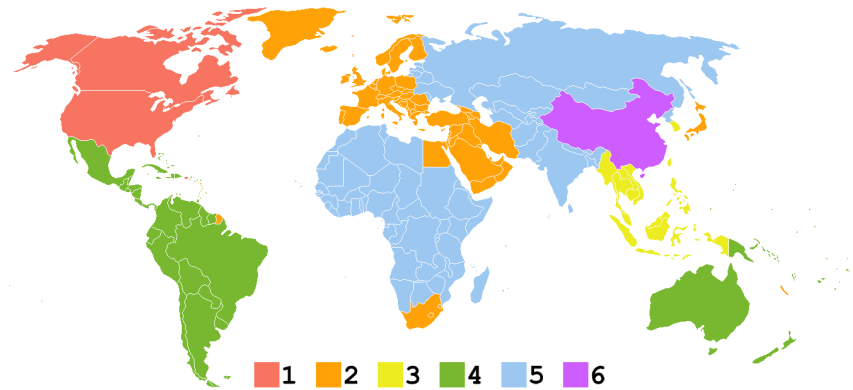
In a digital world, however, anything is possible when it comes to creating rules for things. Stanford University law professor Lawrence Lessig has written about what terms like copyright and trespass, mean in a digital world in an interesting work entitled *Code 2.0*. I will focus on just one aspect, digital enforcement of copyright, or Digital Rights Management (DRM). In fact, with DRM, arbitrary copyright provisions can now be created and enforced.

An unintended consequence, however, is that whatever rules are created, whether onerous or not, you are provided no recourse because the rules are encoded in software. A paper book cannot prevent me from accessing its contents, but a digital one can. Here is a quote from a [user](#) of the Microsoft Zune music service:

The vast majority of the music I had *purchased* last year is completely gone. There's no refund, the music doesn't exist on the service anymore, the files are just garbage now. Here's the error: "This item is no longer available at Zune Marketplace. Because of this, you can no longer play it or sync it with your Zune."

DRM has so far had mixed results. The first major use of DRM is that which exists in DVD players. Most people don't realize that DVD players have DRM because DVDs can be very easily exchanged, unlike an iTunes song, whose hassles are well-documented.

In fact the only time you can notice the DRM is when dealing with region encoding:



*DVD region encoding map*

While this picture has a harmless, Benetton-esque, multicultural feel to it, it also creates a hassle for anybody who crosses those boundaries, and arguably decreases revenues. During layovers in European airports, I have often browsed selections of books, music and clothes, but I didn't purchase the European DVDs because they will not play on my North American player. I have a friend who lives in Latvia, and his choices in entertainment for his children are a tiny fraction of what is available in the U. S. because of this mechanism. The DVD region-encoding reinforces his country's third-world status.

In general, the DRM in DVD players is not onerous because it actually doesn't protect anything. The contents of the DVD are protected by DRM via encryption — but the key to its contents is also on the DVD!<sup>4</sup> Typing in a password to play a DVD player would not go over very well, so the password is put on the DVD. Once you can read any DVD, you can read them all. It is impossible to lock an object on its own.

---

<sup>4</sup> The DVD key is 40-bits which is very small, and was made small because of the US export restrictions which existed in 1996. It has been said that computers powerful enough to play a DVD are powerful enough to crack the password!

Furthermore, the use of a DVD involves decoding it; the decoding mechanism by itself cannot distinguish whether the user intends to just watch the movie or post it on the Internet, so creating security to prevent people from copying DVDs is impossible.

Moreover, if a DVD is illegally posted on the Internet, then the security mechanism on all other copies of that DVD is now pointless because the information has already leaked out. In fact, the unencrypted version of the data is more useful because it doesn't have the restrictions on it.

The key to security in any system is to have multiple layers of protection, a concept known as defense in depth. Inside a locked house, you might have a safe to store important documents and jewelry. An alarm system and cameras add additional levels of protection.

If you want to truly secure the contents of a DVD, you should put it in a safe, encrypt the contents with a key stored away from the DVD, secure access to the machine playing the DVD, make sure no third-party applications are running on the computer playing the DVD, etc.

One can obviously see that these mechanisms don't make sense for the content we own. While I understand the interest in protecting copyrighted materials, we should not add complexity that serves no purpose.

Because DVD decoding is protected by patents with per-copy licensing fees, Microsoft decided not to include a means of playing DVDs in Windows XP. (Microsoft likes to charge per-copy license fees for its software but never likes to sign such license agreements itself! DVD playback is one of the applications that hardware vendors must include with Windows to make it fully-functional.)

In fact, the DRM mechanisms create obstacles for the proprietary software vendors, who try to legally jump through these non-existent security hoops, even more than for the free software guys, who have just written code to do the job — and been prosecuted.

If you sat down to write code to play your DVDs on your own computer (not an easy task, I admit) you would be breaking the law! Jon Lech Johansen, one of three people who first wrote free code to play DVDs, suffered years of legal harassment.

The Digital Millennium Copyright Act (DMCA) of 1996 says that writing or distributing DVD decoding software is illegal because it “circumvents a technological measure that effectively controls access to a work.” It is the DVD industry that has deigned itself exclusive provider of technology to play your DVDs. Even if you wrote this software just to play your own DVDs, you are breaking the law, and *Universal v. Reimerdes*, the first lawsuit testing the DMCA, has upheld this. According to Judge Lewis Kaplan:

In the final analysis, the dispute between these parties is simply put if not necessarily simply resolved. Plaintiffs have invested huge sums over the years in producing motion pictures in reliance upon a legal framework that, through the law of copyright, has ensured that they will have the exclusive right to copy and distribute those motion pictures for economic gain. They contend that the advent of new technology should not alter this long established structure. Defendants, on the other hand, are adherents of a movement that believes that information should be available without charge to anyone clever enough to break into the computer systems or data storage media in which it is located. Less radically, they have raised a legitimate concern about the possible impact on traditional fair use of access control measures in the digital era. Each side is entitled to its views. In our society, however, clashes of competing interests like this are resolved by Congress. For now, at least, Congress has resolved this clash in the DMCA and in plaintiffs' favor. Given the peculiar characteristics of computer programs for circumventing encryption and other access control measures, the DMCA as applied to posting and linking here does not contravene the First Amendment.

It appears that the defendants never argued the point that it should be possible to write your own software to play your own DVDs.

The Digital Media Consumer Rights Act, a bill proposed but not passed by Congress, redefines such software to the status of barely legal. You would be allowed to run “unlicensed” code to play DVDs as long as you are doing it for legal purposes. This makes good sense: playing a DVD isn't a problem, posting that movie on the Internet is the problem!

DRM scenarios are also very limited in their scope. Your iPod's DRM is incompatible with your cable box's DRM. As such, you will never be able to watch your TV shows on your iPod because the industry is unable to arrive at a DRM scheme on which everyone can agree. (In fact, you can't even share videos between cable boxes in your home today!)

There could certainly be uses of DRM that make sense. For example, you could rent a movie which would expire after a period of time, using DRM as the enforcement of the expiration. The debate should focus on rent versus own scenarios as wholly separate classes of transactions.

Another compromise is to create a mechanism where media is stamped with the textual string naming the owner, the digital equivalent of writing one's name on the inside cover of a book. This can be done without encrypting the data.

In general, however, the focus shouldn't be on ways to restrict information, but on ways to make it available. It seems like the recording industry spends more time coming up with new ways to inconvenience customers than in coming up with new ways to increase the quality of the user experience of digital media. When you figure a global middle class of billions of people, worrying about the pirates who live on the edge of society, or on the edge of subsistence, is a mistake.

People will steal if they want to steal. One of the reasons people stole video games in the past was because it was not easy to try out a game before you bought it. Now, with the Internet, you can download free demo games and make an informed decision what games you want to buy in the future. Does providing free demos help or hurt sales? I don't know the answer to this question, but it does make for a better informed and discriminating customer; something from which we all benefit in the long run.<sup>5</sup>

## Music versus Drivers

Compared to the sociological need to protect music and movies, hardware manufacturers in the computing industry are on weaker footing when it comes to "protecting" the intellectual property of the software drivers for their hardware. Drivers are simply the minimal glue that allows their hardware to work with the rest of a system.

Copyright laws were created to stimulate progress, but proprietary drivers serve as an impediment. Rather, they make people hate their computer, dreading the day of work necessary to install a fresh operating system. When you go to the store and purchase a \$500 video card, you would like that hardware to work with your

---

5 I downloaded a World War II flying simulator for my Xbox 360, but I couldn't configure the controls like the controls on an RC model airplane. Don't the programmers of that game recognize the crossover market? Relearning muscle memory is hard and pointless.

computer right out of the box. Given the time to design, manufacture and distribute hardware, software drivers should be available first. There is no reason to have to download any drivers when setting up a new computer using Linux. (Linux often has drivers available before the hardware is released.)

While most drivers for Linux today are free, the graphics card vendors are the most notable holdouts. They look at all the other smaller drivers and argue that their complex software has more worth. Of course, their current “worth” is a mixed bag because proprietary drivers not part of the free software tree are, by definition, out-of-date and incompatible. Linux is working hard to make a 3-D desktop, but it didn't run on an old computer because the proprietary driver didn't support the necessary features.<sup>6</sup>

Good driver support could be a competitive advantage for a hardware company, and Intel is aggressively moving into the 3-D video card space with free drivers. However, it is best for the industry to have the hardware guys fight it out on hardware features and cost, not on software licensing agreements. (An important reminder here is that much of the funding for the Linux kernel comes from hardware companies. Linux and the rest of the free software movement is not being built only by volunteers.)

---

6 When I do enable their nascent 3-D features, the computer doesn't suspend and resume properly anymore. And the cursor would sometimes become malformed. Bugs like this would not survive long if the code were free. In fact, many teams are re-engineering proprietary drivers from scratch just to be able to fix bugs.

# THE OS BATTLE

Free software works well in a complex environment. Maybe nobody at all understands the big picture, but evolution doesn't require global understanding, it just requires small local improvements and an open market ("survival of the fittest").

—Linus Torvalds

I've been a big proponent of Microsoft Windows Vista over the past few months, even going so far as loading it onto most of my computers and spending hours tweaking and optimizing it. So why, nine months after launch, am I so frustrated? The litany of what doesn't work and what still frustrates me stretches on endlessly.

Take sleep mode, for example. Vista promised a new low-power sleep mode that would save energy yet enable nearly instantaneous resume. Poppycock. The brand-new dual-core system I built a few months ago totters off to sleep but never returns. I have to cold-start it to bring it back. This after replacing virtually every driver inside.

Take my media center PC, for example. It's supposed to serve up photos, videos, and music. Instead, it often simply drops off the network for absolutely no reason.

I could go on and on about the lack of drivers, the bizarre wake-up rituals, the strange and nonreproducible system quirks, and more. But I won't bore you with the details.

—Jim Louderback, Editor in Chief of PC Magazine.

**T**o a Linux distribution, the kernel is the software that makes all other software run, and is one of the thousands of components they integrate. The Linux kernel by itself will not defeat Windows — it requires an entire distribution.

There are many producers of Linux distributions, but in the PC world the four most important teams are: Red Hat, Novell, Debian, and a four year-old upstart, Ubuntu. There are hundreds of other Linux distributors, but most of them are merely using the big four's efforts and tweaking them further for more specialized markets. Each of these four will be discussed over the next few pages, but it is worth mentioning a name that isn't on the list, IBM.

# IBM



*"Prodigy", a powerful ad describing the power of free software, created by IBM in 2003. Given IBM's tepid support for free software in 2008, their management should watch this ad a few more times.*

IBM was the first, and is still the biggest computer company, and has built many operating systems over the decades. However, it has yet to exhibit an interest in producing a Linux distribution for PCs. In fact, you cannot get Linux pre-installed on any of their computers today, even though IBM/Lenovo laptops are extremely popular in the Linux community. It is possible they gave up hope after their antitrust lawsuit in the 1970s, the distraction of mainframes and minicomputers, and the expensive and humiliating defeat of OS/2 by Windows in the early 1990s.

IBM has touted its support of free software and Linux for eight years, but has done very little to even ensure its hardware runs smoothly on Linux. I met an Intel employee whose job was to write device drivers for IBM hardware because IBM wasn't working on the problem. On my IBM laptop, the system generally works, but a number of the devices do not have Linux support.<sup>1</sup> IBM could have

<sup>1</sup> These are the most notable devices that don't work: the fingerprint reader, the broadband modem and the accelerometer – which can signal to the hard drive to



easily done the work, or just put a little pressure on its suppliers to get them to write the drivers. If you can design hardware, you can easily write the corresponding drivers!

In general, various people have labored to make IBM's hardware work for the important user scenarios. But if I un-dock my DVD-player on my IBM laptop, Linux will hang. This sort of dynamic functionality isn't needed, as one can just swap out parts when the computer is off. This code is tricky, the work is boring, and it is not necessary, thus the last bits like this have been left out. It is the sort of code that will not become functional until IBM understands free software can achieve world domination if only they'd lend a hand!

In addition to the above issues, IBM has made chunks of their code free, but still left a fair amount proprietary. For example, random products like Notes, Rational and Websphere have long been sold to enterprises. I wouldn't run those old dogs on my computer at any price, which is why IBM might feel that if they've got suckers who are willing to continue to pay for it, then they should just continue to take their money!

Nevertheless, there still must surely be a divide inside the company on the utility of free software. IBM has purchased the rights to some code and made them free, but often the codebases are ungainly (Eclipse developer tool) or irrelevant (Cloudscape database). The complacency of IBM and others toward Linux and free software contributes significantly to the continuing dominance of proprietary software.

---

park itself if the computer senses it is being dropped.

There are also missing utilities, like the ability to enable an external monitor, the ability to recondition the laptop battery, or set the maximum charge to only 95% of capacity to lengthen the lifetime of the battery. All of these exist on Windows. Then there is the gray area of support: my old laptop contained an ATI graphics card that generally worked, but because it the driver was proprietary, didn't support 3-D, and was buggy.

# Red Hat



The first Linux distribution I used was Red Hat, the largest commercial Linux producer. One might conclude, therefore, that a big part of the reason Linux hasn't taken off with desktops is because Red Hat didn't really focus on building a user-friendly experience. Red Hat built a platform usable by Google but not by our moms. Instead, they focused on developers, servers, grid computing, the web, etc. For many years, certain basic user scenarios, like setting up a shared printer, have been cumbersome on Linux.

Red Hat's chief technology officer, Brian Stevens, was recently asked, "When is Red Hat getting into the desktop space?" His response:

To us, the desktop metaphor is dead. It's a dinosaur. Today's users aren't sitting at home, sitting at a desk in isolation anymore. They are collaborative. They are sharing. They work and play online. We don't believe that re-creating the Windows paradigm with just pure open source models does anything to advance the productivity or the lives of the users.

Stevens is confusing the desktop as a personal and powerful tool *with mass appeal*, with the desktop as a user interface metaphor! If they don't get what the desktop is, it should be no surprise that the Linux desktop hasn't taken off after 10 years of work. Furthermore, there is nothing wrong with the current desktop metaphors of windows, icons, and menus. These concepts *do* allow people to work and play online, and share documents over the Internet. We might

come up with user interfaces like those out of science-fiction movies, but that isn't necessary, or sufficient for that matter, to enable the scenarios that Stevens describes.

Red Hat produces two versions of a Linux operating system: a free, community-oriented product known as Fedora, and a commercial version known as Red Hat Enterprise Linux (RHEL), which is typically purchased by corporations as part of a support contract.

The wisdom of this dual-market strategy is debatable because it separates the user base and the brand. Red Hat's enterprise product generates the most revenue, so this is where the company's focus lies. Unfortunately, few normal users run the enterprise product because it is rather expensive<sup>2</sup> and it ships every 18 months, which is a long time given the fast-moving state of free software.

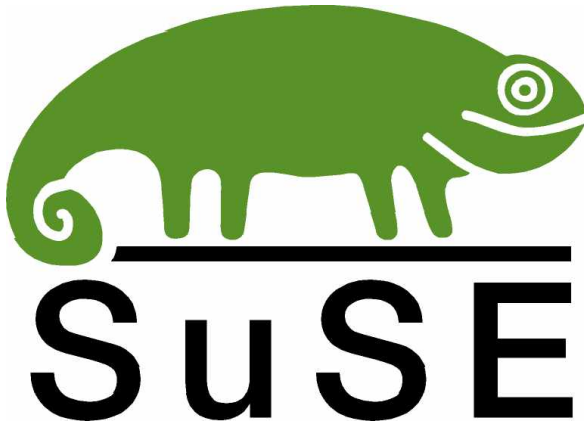
Red Hat's concentration on its for-profit enterprise offerings has distracted it from creating a healthy community of interested geeks who could have contributed free improvements and additions to both versions.

It is an open question as to how much revenue Red Hat would make if they chose not to charge for their enterprise version, but charged for support. Red Hat has been charging for their enterprise version for many years, and effectively bundling support for free, but in fact people typically buy RHEL because they want support, not because the software is significantly better than the community version. In fact, Red Hat could decide to sell support for both versions. These changes might even generate more revenue for them! With only two million RHEL customers today, the potential for growth is huge.

---

<sup>2</sup> \$349 - \$1300 per year, depending on whether you get two-day web response or 24/7 phone response.

# Novell



Those who have heard of Novell remember it for their proprietary, server-oriented operating system known as NetWare. The product never gained critical mass and soon lost out to IBM and Microsoft.<sup>3</sup> However, Novell acquired German Linux distributor SuSE in 2003 for \$210 million, and the company has since refocused itself on Linux and the free software movement.

While Red Hat has concentrated on data centers, Novell is targeting business desktops, which need directory services, management tools, and Microsoft compatibility. Novell's experience with NetWare and its other products gives it the institutional knowledge necessary to understanding how to meet the needs of enterprises through free software.

This merger appears to be working fairly well and Novell is a well-respected company in the free software community. SuSE is one of the most popular Linux distributions, but like Red Hat, they have separate community and enterprise versions, and communities, that is potentially stunting their growth.

---

<sup>3</sup> The kernel also had fatal flaws for server scenarios: it didn't support preemptive multitasking and ran application code in the kernel, therefore hurting reliability.

# Debian

Large organizations cannot be versatile. A large organization is effective through its mass rather than through its agility. Fleas can jump many times their own height, but not an elephant.

—Peter Drucker



*Welcome banner for the 2007 Debconf. Debian is one of the largest engineering teams you've never heard of.*

Debian, the big dog of community-based Linux distributions, is used on servers, desktops and embedded devices, and is respected for its power and reliability. Like Wikipedia, it is built by a community rather than a corporation, and was created by a college sophomore in 1993. The name Debian is a conjunction of his girlfriend Debra, and Ian, the founder's first name.

Ian Murdock, a student at Purdue, wrote in his “Debian Manifesto” that writing an operating system is a job which is too boring for any corporation to do well. Ian was dissatisfied with the quality of the products produced by commercial companies building and selling Linux distributions:

Operating systems are neither easy nor glamorous to construct and require a great deal of ongoing effort from the creator to keep the distribution bug-free and up-to-date: to ensure that the system is easy for others to install, is installable and usable under a wide variety of hardware configurations, contains software that others will find useful, and is updated when the components themselves are improved.

This argument is the reverse of the typical argument made about free versus proprietary software. Perhaps some software, like an operating system, is so large and tedious to maintain that no one corporation can do a good job. Whether Ian is right or not, it is true that the free software community can build a good product: Debian is a completely free software distribution which has harnessed an army of 1,000 developers to build a rich and reliable system. There are challenges facing Debian, but none of these challenges relates to the quality of the engineering

At last count, these engineers have incorporated 283 million lines of code. By one metric, the total cost to write the software is \$14 billion! Debian sets a high standard for inclusion into their platform, and they have been very influential in making sure that the free software community stays reliable and free. Nobody knows when free software will take over, but much of the code is already written and available in Debian.

Windows and Apple's Macintosh don't get better with more users because those users can't contribute anything technical back. Any conversation between a customer and a Microsoft employee is just an unprofitable distraction from writing and selling new software. To a lesser extent, this is even true with Red Hat and Novell's Linux because most of their development, even of the free versions, is done mostly by full-time employees as their efforts to harness their community of users has long been an afterthought.

Team size is a very important metric, in fact one of the most critical ones. All other things like productivity being equal, the team that has the most engineers will win. The biggest reason why Internet Explorer beat Netscape is that Microsoft created a bigger team.<sup>4</sup> A larger team can do more, including absorb new people faster and build up institutional expertise in more areas. The lesson of Metcalfe's law is that the first to achieve critical mass wins: Google, YouTube, Wikipedia.

In 2007, I went to the annual Debian Conference and was very impressed with the strength of the team; many of the attendees were of a similar caliber to my former co-workers at Microsoft, even though they weren't screened via a day-long intensive interview process.

In fact, many of Debian's components are packaged, updated, and maintained by people who are using that component for their personal or professional use. Hewlett-Packard and other companies contribute to Debian to make it work better for their customers, and all have a voice at the table. This perspective which transcends companies and geographies, can lead to a healthy state of affairs.

Debian has governance structures, although their leaders play a very small role in guiding the team in any particular direction. A feature is added simply because someone decides it is a good idea. Details are hashed out in e-mail discussions, blogs and conferences. The person doing the work makes the final decision, which is why free software has been called a Do-ocracy, but the Internet allows him to get questions answered and leverage the expertise of others. Debian's collective expertise in understanding all of the software on its DVD is its greatest asset.

A Debian developer's primary job is to update the many free software components to the latest version and then find and fix interaction bugs between components in the system. Debian does write its own code, but that is a small yet important part of what is actually on a Debian CD.

Debian is a spiritual leader of the free software community and has a very ambitious goal embodied in their motto: to be "The Universal Operating System." While they have yet to achieve this objective, they have come very close. Debian contains 18,200 software applications that run on 15 hardware platforms and support hundreds of languages.

---

4 Microsoft's nearly unlimited supply of smart and experienced developers, whose expertise in text processing, windowing, control widgets and dealing with the challenges of building large codebases meant Netscape didn't stand a chance.

Debian's biggest mistake so far is that its releases have never truly targeted a desktop operating environment. Their product has shipped every two to three years, which is an eternity in the fast-moving free software world. Long shipment cycles encourage procrastination by contributors, thereby lengthening the release cycle even more.

With software distribution, the ship date is as much a choice of the community as it is a technical limitation. As I wrote earlier, Debian writes little code and ships only stabilized versions of its components. That is why they can ship on any schedule they choose. The latest code could be pushed out every day if you had thorough testing. In fact most Debian developers run the “Debian Unstable” release which is precisely that.

While Debian has a 2-3 year release cycle, many other free software organizations have moved to more frequent releases, and there haven't been a lot of complaints about decreasing quality. In fact, the faster you release, the faster you can incorporate the latest and greatest code, which is better in every way. Furthermore, frequent releases increase the excitement, so that bugs get fixed faster.

Debian had the potential to be the “Wikipedia” of Linux distributions, but it hasn't come close to achieving this status. It was one of the most popular Linux distributions in the early years, but it has now fallen far behind the leaders to number seven.

So what happened? Mark Shuttleworth, a former Debian developer who became a dot-com billionaire, decided he wanted his own Linux-based operating system. And so, with \$10 million and what is arguably an unethical interpretation of the spirit of free software, created a brand-new Linux distribution system.

Enter Ubuntu – which sat on a platform that was 100% Debian's. Now, with changes and improvements, Ubuntu is 99.9% Debian, and is taking over the Linux distribution market primarily by leveraging the strengths of Debian.



# Ubuntu

2006年11月2日

MARK SHUTTLEWORTH 在中国

UBUNTU LINUX 的创始人

开源世界的精神领袖

宇航员？企业家？还是科学奇才？

与MARK SHUTTLEWORTH会面, 详情请参阅:

[HTTP://WWW.UBUNTU.COM.CN/RELEASEPARTY](http://www.ubuntu.com.cn/releaseparty)

NOVEMBER 2ND, 2006

MARK SHUTTLEWORTH IN CHINA

FOUNDER OF 'UBUNTU' LINUX

SPIRITUAL LEADER IN OPEN SOURCE WORLD

A COSMONAUT? AN ENTREPRENEUR? OR A REAL GEEK?

TO MEET MARK SHUTTLEWORTH, PLEASE FIND DETAILS IN:

[HTTP://WWW.UBUNTU.COM.CN/RELEASEPARTY](http://www.ubuntu.com.cn/releaseparty)



[HTTP://WWW.UBUNTU.COM](http://www.ubuntu.com)



*Mark Shuttleworth, dot-com billionaire, space tourist, and creator of Ubuntu Linux*

The GPL (General Public License) allows anyone to access Debian's code and take it in a new direction. That is what each of the 130 derivatives of Debian have done. However, most of that work has so far only involved relatively small changes, mostly subsetting Debian, with little fragmentation of the Debian community or brand. All that changed when Ubuntu was created.

What began in 2004 has become the most popular and fastest-growing Linux distribution system. While its name is bizarre, Ubuntu is all the buzz in the Linux community. Free software luminary Eric Raymond made a very public [statement](#) when he switched from Red Hat to Ubuntu in February 2007. Dell announced in May 2007 that they were going to offer Ubuntu on several Dell computers, very big news indeed. The latest public numbers are that as of November 2006, Ubuntu had eight million users and was doubling every eight months.

Ubuntu comes from the African language Bantu and is a humanist ideology:

A person with ubuntu is open and available to others, affirming of others, does not feel threatened that others are able and good, for he or she has a proper self-assurance that comes from knowing that he or she belongs in a greater whole and is diminished when others are humiliated or diminished, when others are tortured or oppressed.

—Desmond Tutu

In short, Ubuntu means: “humanity towards others”, or “I am because we are.” Lovely. Shuttleworth, a South African himself, with \$10 million — a minuscule amount by the standards of the well-funded IT industry — and a copy of the Debian code, hired ten of its best volunteers to work full-time.

Ubuntu took Debian's solid base of packages and added a few innovations: they ship every six months, much more frequently than Debian's two to three-year release cycle, and they focus on ease of use and better support for proprietary drivers.

Perhaps for the first time, a Linux for human beings was created, Ubuntu's motto.



*Today, Linux (and computing in general) is mostly for male human beings.*

Periodically, Ubuntu designates stable releases for which they will provide paid support. This is a different model from that employed by Red Hat and Novell, which provide support only in their Enterprise versions that are separate products with separate life cycles.

With Ubuntu, the code is free, and there is all sorts of community support on forums and blogs, in short, a search engine is your friend. English majors can follow the steps to fix a problem, even if they don't fully understand them! In fact, it is often faster to type in a query to a search engine than to call someone who will usually ask many silly questions ("Is your printer on?")

Shuttleworth's success demonstrates that it is possible to foster a community of volunteers around a team of hard-core, full-time, paid developers. Ubuntu's full-timers spend their hours tracking down the hard kernel bugs and maintaining the larger, complicated packages, which often have more arcane gruntwork associated with

them. The volunteer community today focuses on maintaining the bug list, creating artwork, translating the user interface, and adding applications that Ubuntu is happy to provide on their servers but not necessarily fully support. With a large user base, if everyone just chips in a little, a lot of progress can be achieved.

Many Ubuntu users are passionate about the potential of free software and want to help, but they are unable to contribute back in any technical way. This is one of the reasons why donations of money should be used for free software, a topic I will discuss more in the next chapter.

Perhaps the biggest reason why money isn't donated to free software is that hardly anyone is asking. On Ubuntu's website, they encourage you to volunteer your time, but they do not ask for money.

Just \$1 per computer per year would cover Mark Shuttleworth's costs to run Ubuntu. Computers cost between \$500 and \$2,000, so five dollars for the five years you might keep your new computer is not very much to ask!

## Should Ubuntu Have Been Created?

I am both a Debian and an Ubuntu developer, and I'm sometimes amazed that Ubuntu discusses technical choices that were discussed (and solved) a few weeks earlier in Debian.

—Lucas Nussbaum

What many people don't understand about Linux development is that it's truly a team effort:  
Red Hat develops the kernel,  
Novell develops the applications,  
Debian does the packaging,  
and Ubuntu takes the credit!

—Joke found on a bathroom stall at LinuxWorld Boston 2005

The next chapter discusses challenges facing the free software community, but Debian/Ubuntu is a specific one to discuss here because it is a case study on the software vehicle that today is the most likely replacement for Windows and the Mac.

While the Linux community has benefited greatly from Ubuntu's investments and focus on the deficiencies of Debian, it is not clear why Shuttleworth needed to fork Ubuntu to improve Debian in the first place. Hiring volunteers to work full-time is a good way to speed up progress, but they could have done their work inside of Debian if Mark had told them to. In addition, there is an argument to be made that both Ubuntu and Debian are hurt by the split.

It is widely accepted in the free software community that Ubuntu and Debian have a special relationship. Ubuntu's website says that Debian is “the rock” that Ubuntu is built upon. Given that Debian is installed on millions of machines, has been around for 15 years, and has 1,000 developers, this analogy is apt.

While everyone agrees that Ubuntu's hurting Debian is bad for the free software movement, no one knows the extent of damage to Debian. There are no accepted and published metrics that geeks can use to help analyze the problem — like the number of Debian users who have switched to Ubuntu.<sup>5</sup> The Debian developer community is growing linearly, while Ubuntu and other free software efforts are growing exponentially.<sup>6</sup> There certainly must have been a slowdown at Debian around the time of Ubuntu's creation in early 2004.

We know the changes in Ubuntu could have been achieved by Debian because Shuttleworth hired ten of the best Debian volunteer developers, and they started work in a 100% Debian codebase. Debian is very highly respected within the Linux community, and the pre-Ubuntu consensus was that it was just missing a little polish and dynamism. This could have been easily fixed, especially with the shot in the arm of a few volunteers transitioning to full-time developers. Other computer companies have done their work directly in the Debian codebase, and Shuttleworth has never given justification as to why he couldn't adopt a similar strategy.<sup>7</sup>

Geoffrey Moore's recent book *Dealing with Darwin* talks about companies getting eaten up by “context”; irrelevant things not “core” or important to the business. With Ubuntu, Shuttleworth created yet another bug-tracking system, source control software, wikis, forums, and in general invested in a lot of infrastructure

---

5 Other good ones are the rate of growth of Debian users versus other, non-Ubuntu, distros. Also useful is the number of Debian developer-hours per person per week.

6 Based on a conversation with former Debian leader Anthony Towns, who said that the number of developers joining Debian has been constant over the last few years.

7 One of the biggest challenges would be for Debian to have two release cycles, one every six months, and one when “it is ready”, which is Debian's current modus operandi. This is non-trivial, but doable.

Former Debian leader Martin Michlmayr argues in his PhD thesis that Debian should switch to time-based releases. Debian believes they have, but it is an 18-month release cycle, and they still allow themselves to slip. I think a yearly release — perhaps on Debian's birthday — would be a good thing; the best reason for short release cycles is that it discourages procrastination.

Wider use of Debian Testing would be another possibility. Debian Testing contains the latest *tested* versions of all the applications all the time. New versions of the applications are pushed to Testing after sitting in the Unstable branch for a few days without any major bugs being found. The package manager even supports a way to install older versions of packages, so you are never stuck.

already in place at Debian. None of that work made Linux any more ready for human beings. Perhaps one-half of Shuttleworth's early team focused on this non-core work. Even assuming the Debian infrastructure needed work to make it ready for human beings, it is cheaper and better to improve an existing system than to build a new one just to add a few features.

New Linux users are joining the Ubuntu team and contributing to the Ubuntu codebase and community because that is the one set up for them. By analogy, it's as if someone took Wikipedia, made a few small changes, and this became the website people used rather than Wikipedia itself. If the changes were so good, why were they not made a part of Wikipedia, leveraging Wikipedia's expertise and processes? Putting the changes directly into Wikipedia would also be better because it would improve Wikipedia, which is what everyone is already using. Forking a codebase is primarily social engineering, with widespread impacts, and goes against the spirit of cooperation inherent in science and free software. In the Linux kernel, the good ideas are incorporated and the bad ones weeded out, but this is all done within the context of one codebase and team. Anyone who wants to improve on Linus's work can just e-mail him some code changes, there is no reason to create a separate, "competing" kernel.

There is evidence of the inefficiency in the separate organizations. Today, Debian developers and Ubuntu developers are working mostly on separate tracks. One of the best practices I learned at Microsoft was to give a developer full responsibility for a feature. The person adding the footnote code to Word would be responsible for the changes to the user interface, the file format, and the layout engine. This meant collaborating with other developers, but it also made just one person the feature expert, enabling him to make all decisions with a holistic view. Most importantly, it is efficient for this one person to fix any feature bug.

The hard part about doing something is learning *how* to do it. Making a gourmet dinner, landing the Space Shuttle, or performing open heart surgery is a few hours' work if you know what you're doing, but it is time-consuming and nerve-racking if you do not.

When a Ubuntu developer adds a feature, he designs and implements a feature in Ubuntu, but not in the Debian codebase. Now, the only person who understands the change is the Ubuntu developer who made it. Ubuntu publishes its source code on a website, but if a Debian developer grabs it, and runs into problems, he is not an

expert in this code yet because it was the Ubuntu developer who first made the change. Therefore, he will need to spend time getting up to speed.

The time to get up to speed is comparable to the time to do the work in the first place. In fact, the Debian developer who integrated a huge set of "X.Org" patches from Ubuntu told me that they were just a "starting point," unsurprisingly providing little more help than if he had done the work from scratch.

If a different codebase had never been created and all the Debian and Ubuntu developers were working in the same one, they would automatically work more efficiently. They wouldn't need to redo, and therefore re-learn, what someone else has just done. This would enforce a division of labor and would increase the pace of progress.

Having separate teams is inefficient, but it also hurts Ubuntu's quality. Whenever a Debian developer is re-learning about a software change first made in Ubuntu, he isn't using that time to move forward on new things.

Furthermore, Ubuntu isn't the beneficiary of Debian's greater expertise, which means their code is buggier than it could and should be. Debian and Ubuntu's buglist is one of the best metrics today for the set of obstacles preventing world domination. Ubuntu's user base has grown dramatically, but their small and young team has shown no ability to keep up with the new issues that have come piling in along with the new users. In May 2006, Ubuntu had 10,000 active bugs, and in February 2008, Ubuntu had 40,000.

I discuss more about the challenge of bugs in the next chapter, but the fact that Ubuntu has so many bugs means that there are unsatisfied users, and this is stunting Ubuntu's growth. Debian's much larger and more experienced team could provide great assistance, but because the team's release cycles and bug list are separate, there is no unified effort being made to resolve this challenge.

Because the Ubuntu team is smaller than the Debian team, they argue that they are too busy to take ownership of their work inside Debian. This idea is flawed because if someone else is redoing your work, then you aren't actually accomplishing anything. "Ya can't change the laws of physics, Captain Kirk!" If you are not accomplishing anything, it doesn't matter how busy you think you are. Smaller organizations should actually be more sensitive to wasted work because they have fewer employees.

Shuttleworth claims that Ubuntu and Debian are going after different markets, but he can give no examples of features Ubuntu

wants that Debian doesn't want. If you consider the areas in which Ubuntu has already made engineering investments: simple menus, 3-D graphics, faster startup, and educational software, it should be obvious that Debian wants all of these features as well.<sup>8</sup>

Many of the features that Ubuntu has added, like better suspend and resume for laptops, Debian is no longer motivated to add because almost any Debian user who wanted this feature is now using Ubuntu. Even if Debian does the work, they might not find the bugs because it doesn't have that many users testing out the feature. Debian is being consigned to servers and embedded, which has always been their strength, however, these are areas now being targeted by Ubuntu.

In a recent [blog post](#), Shuttleworth wrote that he admires Debian's goal of building a universal operating system, but he also said in the same post that he believes its objectives are unrealistic. Mark should trust his idealistic side and realize that because software is infinitely malleable, all of his software innovations can be put directly into Debian. There are strongly unified teams building Wikipedia and the Linux kernel, and their success stories can be applied here.

There is understandably a fair amount of bitterness around, which itself decreases the morale and productivity of the community. Debian has spent over a decade doing foundational work, but Ubuntu has made just a few improvements and grabbed all the excitement and new volunteers. I believe Debian has been terminally damaged by the split.

A separate user community is inefficient, but this is dwarfed by the inefficiency of the separate developer community. The greatest long-term threat to Debian is that they stop accumulating institutional knowledge. The best way to prevent this is to encourage Ubuntu users to join the Debian community as well. Debian is filled with many experienced programmers and is a great place to receive mentoring. And because changes are automatically propagated over to Ubuntu, work in Debian helps Ubuntu.<sup>9</sup>

---

8 Some argue that supporting as many processor platforms as Debian does is more work than supporting the 3 that Ubuntu supports, but there is very little architecture-specific code in Debian – most of it lives in the Linux kernel and the C compiler. Additionally, Debian has platform maintainers, who are constantly watching if anything breaks. Like with many things, Debian already has the infrastructure and is already doing the work.

9 There is a gaming team that recently decided to do all their work in Debian and just let the changes flow downstream. If all patches flowed in both directions, as everybody claims to want, and Debian and Ubuntu shipped on the same day, how would someone decide which distro to install?



I was honored to be a speaker and discuss some of these issues at the annual Debian developer conference in June 2007 in Scotland. A great thing about the free software community is how you can meet and talk with everyone. However, because Ubuntu is one of the most successful Linux distros and has been around for several years, the status quo is accepted. People who only look at the success of Ubuntu are ignoring the opportunity cost of doing things in a better way.

If Ubuntu and Debian were to combine their resources, it would eliminate animosity and the end result would be more innovative and reliable. This organization would be in a very good position to replace Windows. The first free software distribution with a community of 10,000 developers wins.

## One Linux Distro?

A free society is about voluntary communities cooperating through the division of labor.

—John Stossel

One of the big questions in the free software community is whether there should be just one Linux distribution tasked with taking on Windows. In theory, Windows provided a standard platform for developers. Windows '95 had a look and feel little different from Windows NT, and the differences were minor from a developer's perspective, and were mostly bugs.<sup>10</sup>

One distribution is an efficiency gain over many distributions, but the cost isn't as high with Linux. Most free software component's greatest dependency is on the kernel, compiler and other low-level tools, and these are the same across Linux distributions.<sup>11</sup>

The job of installing a software component and working through any compatibility issues is the responsibility of the Linux distribution. Because the cost for each additional distro isn't paid for by the component developer, it doesn't slow progress. In fact, the cost of today's many Linux distributions is relatively small in the grand scheme of this million-man movement.

---

10 The Unicode support in the US release of Windows '95 was minimal and buggy. Every non-trivial Windows application queries the version of the operating system to do different behavior.

11 If a distribution finds a bug in the Linux kernel, it will put a bug into the bug database. The fix usually goes into the next release of the software, but a distro can backport into their current version. The difference between the kernel in different distros is the version and the set of backports, which are usually not noticeable to applications.

The best way to increase efficiency is to increase the amount of code sharing between the distros. A good example of such an effort is FreeDesktop.org:

| FreeDesktop.org Projects  |  |
|---|--|
| <p><b>Avahi</b> is a multicast dns network service discovery library</p> <p><b>cairo</b> is a vector graphics library with cross-device output support.</p> <p><b>CJK-Unifonts</b> open source CJK unicode truetype fonts with additional support for Minnan and Hakka languages.</p> <p><b>Clipart</b> is an open source clipart repository.</p> <p><b>D-Bus</b> is a message bus system.</p> <p><b>Desktop VFS</b> is a Virtual File System aimed at message loop (gui) applications.</p> <p><b>desktop-file-utils</b> contains command line utilities for working with desktop entries and .menu files</p> <p><b>DRI</b> is a framework for allowing direct access to graphics hardware in a safe and efficient manner.</p> <p><b>Enchant</b> is a new cross-platform abstract layer to spellchecking.</p> <p><b>Enlightenment</b> is a desktop environment and application toolkit suite with lots of pretty pixels.</p> <p><b>Eventuality</b> is an "application automation meets cron" type DBUS based framework for creating a means to schedule arbitrary "actions" performed by conforming apps.</p> <p><b>Fontconfig</b> is a library for configuring and customizing font access.</p> <p><b>GNU FriBidi</b> is a library implementing the Unicode Bidirectional Algorithm and Arabic Joining/Shaping.</p> <p><b>Galago</b> is a desktop-neutral presence system.</p> <p><b>glitz</b> is an OpenGL 2D graphics library and a backend for gl output in cairo.</p> <p><b>GStreamer</b> is a streaming media framework.</p> <p><b>GTK-Qt Theme Engine</b> is a project to unify the GTK and Qt theming engines.</p> <p><b>HAL</b> is a specification and an implementation of a hardware abstraction layer.</p> <p><b>HarfBuzz</b> is the common <b>OpenType</b> Layout engine shared by Pango, Qt, and possibly others.</p> <p><b>Hieroglyph</b> is a PostScript rendering library.</p> <p><b>icon-slicer</b> is a utility for generating icon themes and libXcursor cursor themes.</p> <p><b>icon-theme</b> contains the standard and also references the default icon theme called hicolor.</p> <p><b>IMBUS</b> is a common tier-1 architecture of IM frameworks for connecting input method engine containers and client application libraries.</p> <p><b>immodule for Qt</b> is a modular, extensible input method subsystem for Qt.</p> <p><b>IPCF</b> is an inter-personal communication framework.</p> <p><b>LDTF - Linux Desktop Testing Project</b></p> <p><b>libburn</b> is an open source library suite for reading, mastering and writing optical discs.</p> <p><b>libmimetypes</b> is a simple implementation accessing the shared-mime-database included in <b>PCManFM</b>, a lightweight graphical file manager featuring speed, low resource usage, and tabbed-browsing. This small GPL'd lib can be used for mime-type handling as a lightweight replacement of xdgmime.</p> <p><b>liboil</b> is a library that makes it easier to develop and maintain code written for MMX/SSE/Altivec extensions.</p> | <p><b>libopenraw</b> is an open source library for Camera RAW file decoding and processing.</p> <p><b>libspectre</b> is a small library for rendering Postscript documents.</p> <p><b>Mesa</b> The Mesa 3D Graphics Library, an implementation of OpenGL.</p> <p><b>OpenSync</b> is a project to create a standardized synchronization framework.</p> <p><b>Oyranos</b> is a cross platform colour management system.</p> <p><b>poppler</b> is a PDF rendering library, forked from xpdf.</p> <p><b>Portland</b> provides a set of high level desktop-integration APIs</p> <p><b>SCIM</b> Smart Common Input Method platform, is a platform to develop input method services.</p> <p><b>Scratchbox2</b> is a cross-compilation tool.</p> <p><b>shared-mime-info</b> is a package containing a large number of common MIME types, created by converting the existing KDE and GNOME databases to the new format and merging them together, and software for updating the database based on the share-mime-info specification.</p> <p><b>startup-notification</b> is a sample implementation of startup notification (telling the desktop environment when an app is done starting up).</p> <p><b>Tracker</b> is a highly memory efficient file indexer and metadata harvester.</p> <p><b>uim</b> is a library to support input many languages.</p> <p><b>UTF-8</b> is a project to document and evangelize the use of UTF-8 encodings for open source projects.</p> <p><b>unicode-translation</b> aims to translate Unicode character names and other data into many languages.</p> <p><b>vaAPI</b> provides a decode only video acceleration API for all video formats. Currently in proposal stage.</p> <p><b>waimea</b> aims to be a standards compliant window manager for the X Window System making use of the <b>cairo</b> graphics library for all rendering.</p> <p><b>xdg-utils</b> is a set of command line utilities to simplify integration with a Free Desktop. It has simple functions for creating menus, opening files, setting mime types, and so on. It is part of the <b>Portland</b> project.</p> <p><b>xdg-user-dirs</b> is a tool to handle well known directories in the users homedir</p> <p><b>Xft</b> is a library for client-side rendering of fonts.</p> <p><b>xfullscreen</b> is a useful module for applications or window managers supporting fullscreen modes.</p> <p><b>Xgl</b> is an X server architecture layered on top of OpenGL.</p> <p><b>Xorg</b> is the XOrg Foundation's Public Implementation of the X Window System.</p> <p><b>xkeyboard-config</b> is a central project for keyboard configuration.</p> <p><b>xprint</b> is the X11 printing system.</p> <p><b>xssettings</b> is a reference implementation.</p> <p><b>X Testing</b> provides information on various software for testing X Servers and Clients.</p> <p><b>X Window information</b> is a window information utility for developers of applications, toolkits, and window managers.</p> |

When Linux standardizes font rasterizers and such, unification automatically occurs.

If we moved towards a truly universal PC operating system, what would happen to the long tail of Linux distributions? Eventually, in a

biological system, the biggest and best gain momentum and squeeze out the others. However, we are still in the early years of Linux, and so the natural selection process hasn't yet happened. Today every Linux distribution team is growing — when you have a worldwide market of only 3%, it is a target-rich environment. In addition, you can have subsets and other derivatives.

## Apple

After Woz hooked his haywire rig up to the living-room TV, he turned it on, and there on the screen I saw a crude Breakout game in full color! Now I was really amazed. This was much better than the crude color graphics from the Cromemco Dazzler. ... “How do you like that?” said Jobs, smiling. “We’re going to dump the Apple I and only work on the Apple II.” “Steve,” I said, “if you do that you will never sell another computer. You promised BASIC for the Apple I, and most dealers haven’t sold the boards they bought from you. If you come out with an improved Model II they will be stuck. Put it on the back burner until you deliver on your promises.”

—Stan Velt, former Editor-in-chief, *Computer Shopper*

Apple's iPod and iPhone may be sexy and profitable, but these small devices are specialized in function, so there isn't a lot to say about them. An iPod is busy when playing music, whereas when your computer plays music, it uses less than 1% of its computing power, which is not even noticeable.

For most of Apple's existence, they never really got the idea of the relationship between market share and a developer community. For example, Macs have historically not been allowed in enterprises because no one added the necessary features and applications — because it never got the requisite market share to make anyone want to bother.

Microsoft understood the virtuous cycle between users and developers, and knew that making it easy to build applications would make Microsoft's ecosystems successful. Bill Gates brags that Microsoft has ten times as many partners as Apple, and tools like Visual Basic and FrontPage were important reasons why.

This internal focus that has limited the Mac's potential market-share is now playing itself out with their new devices. Symptoms of this mindset are noticeable in the most basic scenarios: you cannot drag and drop music on and off an iPod, as you can with a digital camera. Even if you could copy over your files, unless it is in one of the few formats Apple can be bothered to support, you would still not be able to play it.

While Moore's law will push these new devices further up the computing value chain, it isn't clear Steve Jobs understands the value of having a developer community extend his platforms because he doesn't see past the potential complexity it creates. In a *New York Times* interview, Jobs said he would not allow third-party technology contributions to his new iPhone because he doesn't want the telephone network brought down by "rogue applications".<sup>12</sup>

Apple later reversed-course and now there are now third-party apps available for the iPhone, and Apple even runs ads touting a capability they initially shunned, but Apple's mindset of ignoring the outside world is still embedded into its culture. When the iPhone software development kit (SDK) was launched, third-party iPhone developers were: "legally banned from sharing programming tips, discussing code or asking questions of one another in forums or over e-mail." Note that an SDK only contains the public information necessary for a developer to write an app, so why they would be so restrictive is inexplicable.

After lots of complaints by developers who were unable to get their code working, and yet were not able to talk to each other, Apple in October 2008 reversed course here as well, so maybe they are *slowly* learning that not all good ideas originate in Cupertino.

One of the reasons Microsoft has won so many battles is that it knows the key to success is to just get early market share and let Metcalfe's law take over, with the ensuing profits. Steve Jobs is presumably satisfied with Macintosh's 3.5% worldwide marketshare and the state of the iPod today. It is this attitude that ultimately will ensure their failure, I believe. Apple's motto for a time was "Think Different" but a more apt one might be "Think Small".

## iPod and Digital Music

In stumbling upon the business of building fashionable music players, Apple has touched upon one of the important questions of the digital age: How do we acquire and archive our music?<sup>13</sup> If we

12 It should be not be possible for software to take down a network. There are also many ways to sandbox applications to limit their ability to do things.

13 Another important question that is not close to being answered: How does my music device plug into my car? A device should not expose itself as a set of files to your car radio because that requires your car to understand how to decode all the various formats. Even if the car does understand MP3, it won't understand audio bookmarks. My car radio wants to fast-forward through hour-long podcasts a couple of seconds at a time, whereas the RockBox OS on my iPod gets increasingly faster and is configurable in this regard. What we likely need is a streaming protocol: the iPod will implement functions like fast-forward, next song, show directory, search, etc.

A question: since you can understand voice that's 50% faster, should that play-

do it right, we can store our music in one format for decades, even forever.

A digital format is something not tied to a hardware medium in the way that the VHS format was tied to VHS tapes; everything digital can be copied to a hard drive or a USB key. (Iraqi terrorist Zarqawi's death was a big setback for al-Qaeda because we recovered a USB key containing his terrorist documents and music. One day, the only thing we will ever need to waterboard terrorists for is their passwords.)

We have been able to create basically one format for digital cameras — JPEG, which is free and efficient. We should have been able to do this for audio as well because the underlying math is similar!

Apple made the digital audio format problem worse by endorsing one that only they use, AAC. And, by adding DRM, they only allow you to play your music on their one device and in their one application. Apple has added hassles and created doubt about whether you will ever control, and therefore truly own, your music.

Steve Jobs is ecstatic that iTunes has sold 2.5 billion songs in five years, but when you consider that the music business is a \$40-billion-dollar per-year industry, and Apple has no serious competitors in digital music, that number is modest.

I've met many people who have told me that they won't buy an iPod again because of these and other issues. In fact, I run an alternative OS on my iPod, called Rockbox.<sup>14</sup> It supports more audio formats, lets me copy files back and forth like you can with a digital camera, it even sounds better because of a feature known as “cross-feed”. (When you listen to music on room speakers, each ear can hear music, slightly delayed, from both the left and right channel. However, this does not happen with headphones. Rockbox uses clever algorithms to simulate the sound from the opposite channel, slightly delayed, to make headphones sound more natural.)

Rockbox also comes with more software and other advantages. It is perfectly logical that the free software community can do a better job than Apple because Apple likely had 20 software developers writing code for the iPod device, whereas Rockbox has had more

---

back setting be available in the car radio UI, or be a setting of audio files? Suppose you wanted to override that behavior, would you be able to do that through the UI of the radio?

14 iPodLinux doesn't quite work yet. The installer repeatedly crashed, after installation I couldn't find a way to play music, I had to reboot into Apple's iPod OS to copy music over, etc. The momentum is with Rockbox.

than 300 contributors, and itself reuses a lot of other free code.<sup>15</sup> Even worse for iPod's future, I suspect that Apple scavenged some of the best people from their iPod team when staffing up iPhone — Microsoft would have done so.

Apple is now reversing course and providing more music DRM-free, but they are still **making** it difficult to put music onto your device, not letting other devices to play the songs in your music library, and are now preventing the installation of third-party software on their new hardware. It is only because Apple has such little overall marketshare that they can get away with this sort of behavior.

## Mac OS X Kernel



*Apple giving about as much attention to the iPod as the Mac is like Ford focusing their R&D on bling because they suddenly started making half of their profits on chrome rims.*

Compared to the task of decoding compressed music, the word processor I used to compose this book is 1,000 times more complicated. So, what about Apple's computers? The first relevant fact is that Apple is making half of its profits in the (currently) profitable portable device market which has caused Apple to lose interest in the computer as the most powerful tool ever created.

Apple's computers have an elegant hardware design, but the biggest difference between a Mac and a PC is that Apple's computers run a completely different pyramid of software, that is mostly proprietary like Microsoft's.

---

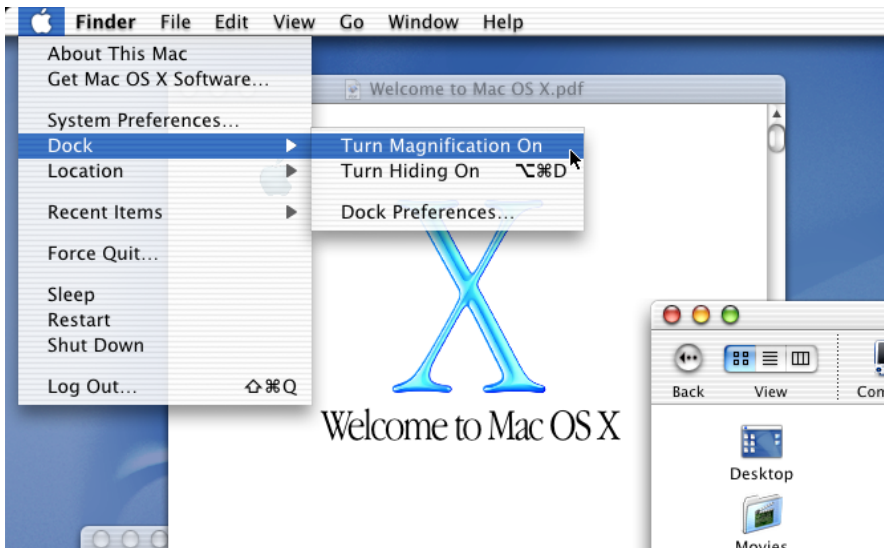
<sup>15</sup> My biggest complaint with Rockbox is that they haven't standardized the "back" button.

From its introduction in 1984 till 2001, the Macintosh ran a kernel built by Apple for its first Macintosh. It was written in Motorola's assembly language, and didn't have memory protection, preemptive multitasking, and many other features of a kernel you would read about in a college textbook. In the late 1990s, this caught up with Apple as Macintoshes were considered unreliable.

Apple eventually threw away their original kernel in creating Mac OS-X 10.0, but it took them a while to get going on it as they had four CEOs in the 1990s.



*Mac OS 9, the last release of the original Apple kernel and the official version until 2001.*



*Mac OS-X, Macintosh's first release with the BSD Unix kernel and the Cocoa user interface.*

Apple's second kernel wasn't built from scratch, but is based heavily on Berkeley Software Distribution (BSD) Unix code.<sup>16</sup> This code is a lot like Linux, but with a smaller development community and a non-copyleft license agreement. That Apple is depending on a smaller free kernel community, and yet doing just fine, does say something about free software's ability to deliver quality products. Apple's new kernel is certainly much better than the one they tossed away after 20 years of investments!

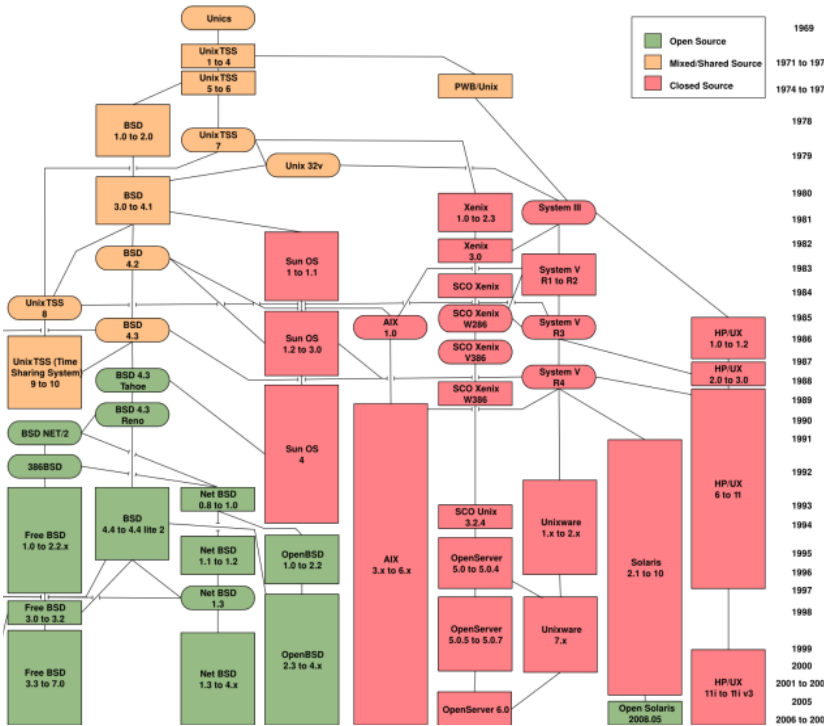
Unfortunately, in choosing this software, Apple gave life support to a group who should have folded their code and resources into Linux. The free software community can withstand such inefficiency because it is an army of millions, but from a global perspective, this choice by Apple slowed progress.<sup>17</sup>

---

16 Apple's kernel also contains pieces of the Mach kernel, but their OS contains much more code from BSD. In fact, I can see little to no benefit from using any Mach code as opposed to its equivalent from BSD. Some have described Apple's combined kernel as a software Frankenstein.

17 When Apple started shipping computers with four and more processors, BSD Unix's performance was worse than Linux's, and in 2003, BSD added scalability optimizations that Linux had added four years earlier. Linux kernel guru Greg Kroah-Hartman has said that Linux runs faster inside a virtual machine than the BSD-based Mac OS hosting it.





*Portion of the Unix family tree. The biggest reason why Unix and Linux hasn't beaten Windows yet is that the workstation vendors didn't work together on the software, and so kept re-implementing each other's features.*

Apple didn't publicly say why it chose BSD over Linux, but at the time it was widely surmised that one of the biggest reasons was that Apple feared the copyleft aspect of Linux's GPL license. Apple didn't understand that it is perfectly legal to ship proprietary code on top of Linux.

However, Apple should have asked itself why it chose a free kernel as the *foundation of its future software*, but not other free code. Apple has not embraced free software, even though it has come to depend on it, and Apple's software total less than 10% of revenues! Apple's hardware team should be happy to run Linux or Windows on its machines. With Macintosh at 3.5% marketshare, PCs are 30 times more an opportunity than a threat.

Supporting other OSES is no big feat of engineering; for a period of time, Linus did his Linux development on a Mac — after discarding all of Apple's software. If Apple did decide to go aggressively after the PC OS market, it would need to make just a few hardware changes: their keyboard is not quite suitable, and they have stub-

bornly resisted the move to two-button mice. Apple already produces a multitude of keyboards for different writing scripts, so they already understand the necessary work.

## Software

Nobody has ever had more contempt for customers than Steve Jobs.

—Eben Moglen



*In a better iWorld, Apple would have used Linux instead of BSD.*

With less of an internal focus, Apple could even have done a few things to make their OS more compatible with Windows, to increase sales and to better tempt Windows users into switching. There is plenty of free software out there to enable interoperation with Windows technologies. Out of the box, Windows Media is treated by the Mac like a text file; there is code out there to fix this, but Apple doesn't provide it. It's as if supporting Windows Media is a concession that weakens Apple. We aren't even talking about having music-creating software support that format — simply the ability to play these files!

Their iChat program doesn't support MSN messenger, though it does support AOL and Yahoo. Given the unnecessary hurdles Apple has created for Windows users, it is not surprising that their market share remains so low. There is even a free Win32 implementation known as WINE that would allow Windows software to run on the Mac; yet another way to storm Microsoft's beachfront that Apple hasn't adopted.

Apple creates proprietary software because they use it to bundle with their hardware, and to lock you into their platform. However, the threat of free software is already approaching Apple. The Linux community is currently focused on building “the Web and Office”, the minimum necessary to build a platform that competes with Windows. Once these are covered and more people start using Linux, then Linux contributors can focus on building video-editing software, and other features that Apple has leveraged to carve out its niche. In fact, supporting the few relevant Apple standards like QuickTime is a much simpler problem than supporting Microsoft's many complicated and popular standards.<sup>18</sup>

When I visit coffee shops, I increasingly notice students and computer geeks purchasing Macs. Students have limited budgets and so should gravitate towards free software. If Apple doesn't support free software, their position in the educational market is threatened.

Many computer geeks buy a Mac because of its Unix foundation. For example, in the terminal window of both the Mac and Linux, if you type “**ps -a**”, you see the list of processes. (Windows doesn't support the rich Unix command-line world except via an add-in that is little used by Microsoft or third-party developers.) Apple has good Unix compatibility only because their programmers never removed it. It was never a goal of the Mac OS-X to appeal to geeks — Apple just got lucky. Smartly, Apple now mentions Unix compatibility in their marketing material.

As a long-time Windows user, and a Linux convert, I tried out the Mac OS X for a couple of days. Here are some of my impressions:

- A Mac OS has more code than ever before, and a lot of it is based on free code, but it doesn't have a repository with thousands of applications like Linux. There are several third-party efforts to provide this feature, but none are blessed or supported by Apple. The Mac comes free with iPhoto, but they really want me to buy Aperture for \$159, which they tell me just added 100 new features! Apple ships a new OS every year, but you don't get free upgrades — it costs \$140 to upgrade from OS X 10.4 to 10.5.
- Many of the Mac's UI details like how to maximize windows, and shortcut keys, are dis-similar from Windows. Linux, by contrast, feels much more natural to a Windows user.

---

<sup>18</sup> WINE is one of the biggest pieces of Windows compatibility code out there, and it alone is much bigger than all the Apple compatibility code one would need to convert someone from a Mac to Linux.

Another example: when you double-click on a picture, it loads the iPreview application that stays around even after the window displaying the picture is closed. How about just creating a window, putting the picture in that window, and having it all disappear when I close the window? I tried to change the shortcuts to match the Windows keystrokes, but their software contains bugs because it didn't change it in all applications.

- The Mac feels like a lot of disparate pieces bolted together. The desktop widgets code has its own UI, and it doesn't integrate well into the OS desktop. The Spaces is a clone of an old Unix feature and doesn't implement it as well as Linux does. (For example, there is no easily discoverable way to move applications between spaces.)
- As mentioned above, the Mac doesn't support as many of the Microsoft standards as Linux does. One of the most obvious is WMA, but it also doesn't ship with any software that reads and writes DOC files, even though there is OpenOffice.org and other free software out there.
- It is less customizable. I cannot find a way to have the computer *not* go to sleep when the laptop screen is closed. The mouse speed seems too slow and you can only adjust the amount of acceleration, not the sensitivity. You neither resize the system menu bar, nor add applets like you can with Linux's Gnome.

Having used all three, it is my view that Linux is better than the Macintosh, which is better than Windows Vista. I hear that internally, Microsoft is very afraid of the Macintosh, but Linux presents a greater long-term threat.

Apple has been good for Linux because it has forced software vendors to think about writing cross-platform code. By using free software like the BSD Kernel, Apple is more of a part of the free software community than before, although not by much. Apple's web browser Safari is based on the engine WebKit, which Apple derived from the free KHTML, but they worked for a year on the fork before attempting to integrate their changes back, and KHTML developers said that their relationship with Apple was a "bitter failure."

Apple does use the free Apache, and the Linux printing system (CUPS), but they have their own office-type applications, and a vari-

ety of multimedia and other tools. They are neither using a fraction of the free software they could, nor releasing much of their code as free software.

## A Free Macintosh OS?



### *Can Apple's software stand alone?*

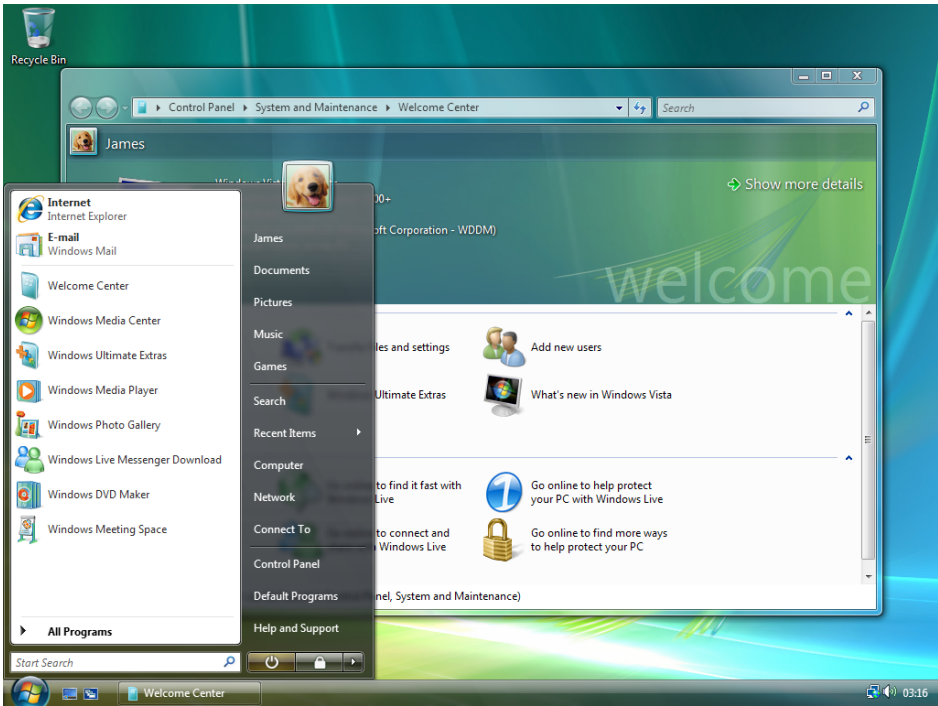
In general, other free software should be looked at as an opportunity for Apple. But what about making all of Apple's software free? If Apple's software were made free, Apple could work more closely with the global free software community and create a better product for their customers. However, the downside is that this software could get ported to Windows and Linux, and create less reasons to purchase a Mac.

However, if a Linux desktop takes off, Apple's OS will suffer the same fate as Windows and force Apple into being a hardware-only company like Dell and the rest, all the value they have built up will be gone. Once people are satisfied using free software, they don't usually go back to proprietary software. For example, free code to convert *from* proprietary formats has much more demand than code to convert *to* them.

If Apple's free software is used on other OSes and hardware, they risk becoming a company competing primarily on hardware features, but they have that risk already, and at least would be in control of their own destiny. Apple could be leading the free software movement! In addition, there are many ways to monetize users of

Apple's free software who are running it on other hardware platforms. It seems unlikely that Apple's free software would be widely used, but not their hardware.

# Windows Vista



*Screenshot of Windows Vista. It is five years of evolution beyond Windows XP, but has many of the same limitations. Whether it will still be prone to viruses and malware is one of the biggest questions.*

Microsoft's Windows Vista was released in December, 2006. The first major release of Windows since Windows XP in 2001, it is an evolutionary step forward in many ways. Every component was improved with new features; a glassy 3-D user interface, integrated search, improved kernel and a new version of Internet Explorer.

While Vista is the next generation of Windows XP, it still has the same fundamental limitations. It will have driver reliability issues because of its backward compatibility constraints and complicated code. It will become obsolete because new hardware is released continuously. Microsoft puts products into a maintenance mode after they ship, and places most engineering resources onto the future source of revenue.<sup>19</sup>

<sup>19</sup> Microsoft isn't able to just give you a new kernel with updated hardware support without giving you a whole new OS. Driver writers building code for the next OS don't want to support the old OS in addition, especially if that next version of Windows is "just around the corner." Microsoft has to threaten and plead with hardware vendors to get them to do all the work that MS creates for them. "In order to get the 'Designed for Windows

Beyond the reliability and interoperability issues, Vista's biggest limitation is that it doesn't contain any major functionality beyond what we all think of as Windows today. If you want an audio creation tool or educational software, Windows is not the answer.

By contrast, like the Linux kernel itself, a Linux distribution has all of its code in one big tree. This means that upon installation of a new version, all the applications work out of the box because they have been built and tested together. A distribution can fix any bug in any of its code because it has total access to it all. By contrast, Russ Cooper, a senior information security analyst at Cybertrust, has said that:

I'm going to Vista...when my VPN supplier tells me that they have drivers that work, and when my anti-virus vendor tells me that they have non-beta versions that work.

Microsoft has to spend much more on backward compatibility than Linux does, and the huge cost has imperfect results because Microsoft doesn't have the ability to fix code in third-party applications. In a free software world, bugs can be fixed anywhere, and therefore *in the proper place*.

In addition to the larger variety of applications in Linux, you find more choices all throughout the free software stack. The user interface is more customizable than Windows or the Mac, and there are communities around even the tiniest parts of a Linux operating system, like themes: every Linux desktop can be personalized to a much greater degree. I can even download a theme to make my computer have a glassy user interface like Vista, or a user interface like the Mac, just two of many choices.

A new version of Windows used to mean a lot because new hardware and software was made a part of the platform. However, because of the dearth and death of third-party Windows developers, and because most developers are building applications on the Internet, a new version of Windows matters a lot less these days.

Furthermore, because deployments of Windows take years, developers are left in a quandary: "If I write code that depends on a new Windows feature, what happens if my code is running on an old ver-

---

2008 Logo' you need to do *this*." I heard a Microsoft Windows Server evangelist say that their 32-bit operating system was going to be retired after Vista, which was why everyone should just go ahead and start work on the 64-bit drivers *right now* — this coming from a company that shipped remnants of 16-bit DOS and Windows 1.0 in Windows Me. A 64-bit Windows simply requires that all hardware vendors recompile their drivers, but because there is no unified tree, this is a hard problem involving a lot of coaxing.



sion of Windows?” Free software propagates with less friction, therefore developers can more easily take advantage of new features, and assume their existence.

A new IBM Thinkpad computer came with Vista Premium and I spent several hours with it. Here are my impressions:

- When I turned on the computer for the first time, it spent five minutes checking its performance before I could do anything; it's rather ironic that this task is slowing down my computer.
- Internet Explorer provided two textboxes to search the web, both using MSN's search. When I create a new tab using the shortcut key after I launch IE, it would say it could not connect, even though the computer had long-ago established an Internet connection. Apparently, the typing part of IE was initialized before the networking part.
- Windows Update wanted to install an Office 2003 service pack even though I could find no Office 2003 installed. Office 2007 was installed as demoware, even though I chose not to purchase it.
- There was a tool to resize the hard drive to make room for Linux, but it insisted on reserving 44 gigs of free space for Vista!
- The disk defragmentation tool said my hard drive contained 3055 fragmented files and 13,265 “excess fragments” and that the drive was “heavily fragmented”. I guess when Vista checked the computer's performance, it didn't notice the filesystem fragmentation issues. The defrag tool was nagware and kept encouraging me to purchase a better version. (In Linux, the file systems have much less fragmentation because the code is smarter.<sup>20</sup>)
- I found seven copies of system files like “i8042prt.sys”.
- I wanted to get Windows Media to play my OGG audio files. I told Windows Media I want to install plugins and it took me to a Microsoft-maintained web page with links to 3<sup>rd</sup> party plugins. I find one and install it. Once installed, I cannot see the plugin in the media player's list of installed plugins.
- I downloaded an OGG file in Internet Explorer, and Windows Media finds and plays the file, but I can't see where it was put. I use the search feature to look for it, but search doesn't

---

20 My ext3 user partition is 67% full and has 4% non-contiguous inodes. My system partition is 52% full and only 1.6%. You would expect the system partition, where not much writing happens, to not be very fragmented. In Windows, it was the system files that were fragmented because it had no user files on the computer yet.

find it. I try to refresh the index, and it warns me that performance will be slow because it is a background task, which is true because it doesn't even attempt to rebuild it.

- I checked on the Windows Media settings for ripping music. The default is to rip into 128 kbps WMA, which has never been good enough for archival purposes. It doesn't allow me to rip into anything higher than 192 kbps. I then discover that there are four types of Windows Media, including an option to use variable bit rate. (VBR encoding uses more or less bits depending on how much the music needs at that moment to maintain a certain level of quality.) The default is inexplicably not to use that clever feature. Windows Media allows me to rip into MP3, but I can only set four levels of quality, and I can't change any other MP3 encoding options like I can on Linux. (Microsoft got sued for billions of dollars for minimal MP3 support.)
- There are now three status icons for wireless Internet, one more than in Windows XP. Let no one say that Vista is not an upgrade: "Now with more 802.11 status icons!"
- After Vista Service Pack 1 is released, I go to Windows Update to try to download it. It isn't there, and shows me no new updates. I log-off, and it tells me it is installing a bunch of updates — ones that it just told me it didn't have.

I have pages of notes like this from just a few hours with the OS. A friend bought Vista on a new HP computer, and while he could see the Internet, he couldn't really transfer data because the Norton anti-virus nagware that came with the computer was incompatible with the Vista firewall. This former MS programmer needed to call HP to get help because the symptom of "sort-of" Internet access didn't lend itself to any obvious solutions. Removing the nagware solved the problem, but that process took 40 minutes — building a high-performance uninstallation program on Windows is apparently nontrivial, even for a company with their act together enough to build a virus scanner.

Someone remarked to me that Vista is "Different but not different." They've tweaked the icons and moved things around, but the same functionality is there. I remember the OS wars of the past, and it seems like we are in for another one. Grab popcorn!

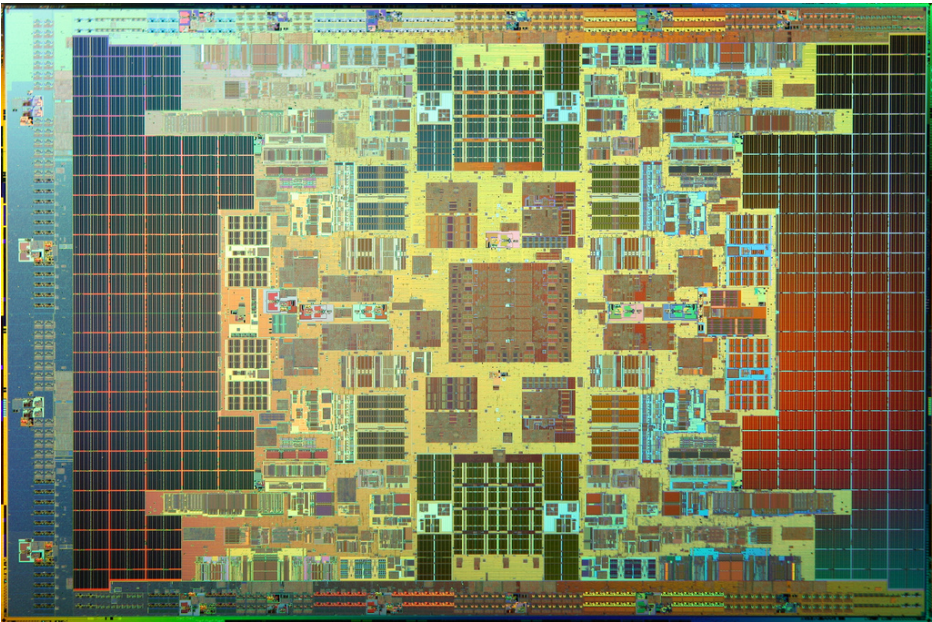
# TOOLS

You can tell a craftsman by his tools.

—Socrates

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

—Edsger Dijkstra, 1972



*Today's hardware is much more advanced than the software.*

The purpose of this chapter is to explore the biggest technical reason why people distrust computer technology today. Many of the problems that frustrate users, such as crashes, security violations, code bloat, and the slow pace of progress, are directly related to the software used to create our software. In fact, the last hardware bug anyone remembers was the Intel floating point division bug in 1994 — nearly all the rest are software bugs.<sup>1</sup>

---

<sup>1</sup> In fairness to Intel, that bug happened on average once in every nine million division operations, and would return results that were off by less than 0.000061. Testing a 64-bit processor's math capabilities involves  $2^{128}$ , or  $10^{38}$  test cases! Intel does continuously release errata lists about their processors. For example, initial-

The problem is this: the vast majority of today's code is written in C, a programming language created in the early 1970s, or C++, created in the late 1970s. Computers always execute machine language, but programming these 1s and 0s are inconvenient in the extreme, so programmers create high-level languages and compilers that convert meaningful statements to machine code. We are asking the same tools used to program the distant computer ancestors to also program our iPods, laptops, and supercomputers, none of which were even conceived of back then.

Imagine building a modern car with the tools of Henry Ford. A tool ultimately defines your ability to approach a problem. While the importance of cooperation in solving big problems is a major theme of this book, the current set of tools are stifling progress. A complete unified set of libraries will change computing even more than the worldwide adoption of the Linux kernel.

Metcalfe's law has unfortunately applied to C and C++; people use them because others use them. These languages became so dominant that we never moved on from our 1970s heritage. Programmers know many of the arguments for why newer languages are better, but paradoxically, they just don't think it makes sense for their codebase *right now*. They are so busy that they think they don't have time to invest in things that will make their work more productive!

Corporations today are struggling with the high costs of IT, and many feel pressure to outsource to places like India to reduce the costs of maintaining their applications. However, outsourcing doesn't decrease the manpower required to add a feature, it only reduces the cost, so this is merely a band-aid fix.

## Brief History of Programming

Q. Where did the names “C” and “C++” come from?

A. They were grades.

—Jerry Leichter

If C gives you enough rope to hang yourself, then C++ gives you enough rope to bind and gag your neighborhood, rig the sails on a small ship, and still have enough rope to hang yourself from the yardarm.

---

izing a computer is a very complicated process with plenty of room for ambiguity, yet because user data isn't even in memory yet, there is no risk of a hardware bug causing a newsworthy problem.

—Anonymous

For the first 60 years computers were programed, and input and output was done using punch cards.<sup>2</sup> Punch cards were invented for the 1890 census by Herman Hollerith, founder of the firm which was eventually to become IBM. Punch card machines operate on the same principles as modern computers but have one millionth the memory. Code reuse was impossible because one couldn't copy and paste punch cards.



*IBM's first computer, used for the 1890 census. Punch cards had been around for 110 years before the world heard of "pregnant chads".*

---

<sup>2</sup> This history ignores Fortran, Cobol, Algol, and other important languages, but this book is PC-focused.

The first programming languages which ran on devices we would recognize today as computers were known as assembly language, and were first created in the 1950s. Programmers almost had to comment explaining what each line of code did because it was inscrutable otherwise:

```

        .TITLE  HELLO WORLD
        .MCALL  .TTYOUT, .EXIT
HELLO:: MOV     #MSG,R1  ;STARTING ADDRESS OF STRING
1$:     MOVB    (R1)+,R0 ;FETCH NEXT CHARACTER
        BEQ     DONE    ;IF ZERO, EXIT LOOP
        .TTYOUT                ;OTHERWISE PRINT IT
        BR      1$         ;REPEAT LOOP
DONE:   .EXIT
MSG:    .ASCIZ  /HELLO, WORLD!/
        .END    HELLO

```

*Code to display "Hello, World!" in DEC's PDP assembly language.*

I worked with an engineer in Office named Murray Sargent who insisted that with practice, you could write in assembly language as quickly as you could write in C. However, this is someone who wrote a physics textbook, after creating a word processor to compose his textbook in, so I never took that statement very seriously. (Working at Microsoft was sometimes intimidating!)

As a college student, I took one assembly language class, which is the perfect amount for a budding computer scientist. I learned about the low-level concepts of computers: pointers, registers, the stack versus the heap, etc.

In addition to being cryptic, the biggest problem with assembly language instructions was that there were a lot of them, as each language contained details about the particular processor it was designed to run on. As a result, in order to run your software on a different computer, you were forced to alter nearly every line of code.

Therefore, one of the biggest advancements in computing was Bell Labs' creation of the programming language "C" in 1970. Compared to assembly language, code written in C is almost English:

```

#include <stdio.h>
int main(void)
{
    printf("Hello, World!\n");
    return 0;
}

```

*"Hello, World!" in C. This is much easier to read than assembly language, but still has weird things like "#include".*



C more comprehensible, but more importantly, by simply swapping the compiler, it enables software written in C to run on every computer in existence today with few changes required. In building the standard replacement for assembly language, Bell Labs changed computing.

Assembly language actually still exists in the nooks and crannies of modern computers: deep in the code of the operating system kernel, inside game engines, and in code to program specialized graphics or math processors. However, most processor-specific knowledge has moved out of the code a programmer writes and into the compiler, which is a huge step up.

While developers at Microsoft used assembly language in the very early years, when I joined in 1993, nearly all code was written in C or C++, and that is still true today.

*Bill Gates**Microsoft, 1978**Paul Allen*

*Bill Gates quit coding somewhere in this timeframe.*

Our most important codebases — Office and OpenOffice, Internet Explorer and Firefox, SQL Server and MySQL, IIS and Apache — are written in C or C++ today.

The world spent the first 1,000 man-years of software development primarily in assembly language, but most programs are written in C or C++. Man has spent 400,000 man-years in those two languages, the same amount of time that the Egyptians spent building the Great Pyramid of Giza:



*Man spent as much time programming in C and C++ as building the pyramids of Giza. Unfortunately, the software used to build our software looks as old and cracked as those pyramids do today.*

Today, the free software community and Microsoft are locked in a closely fought battle: Firefox is comparable, but not dramatically better than IE. This is because neither has yet adopted a productive new programming language for such a large and important code-base. They are warring armies of patient cavemen.

In fact, what both C and C++ are missing is an innovation known as “garbage collection” (GC), or automatic memory management, an idea so subtle and clever that it would be worthy of the first Nobel Prize in Computer Science.



# Lisp and Garbage Collection

Please don't assume Lisp is only useful for Animation and Graphics, AI, Bioinformatics, B2B and E-Commerce, Data Mining, EDA/Semiconductor applications, Expert Systems, Finance, Intelligent Agents, Knowledge Management, Mechanical CAD, Modeling and Simulation, Natural Language, Optimization, Research, Risk Analysis, Scheduling, Telecom, and Web Authoring just because these are the only things they happened to list.

—Kent Pitman, hacker

The future has already arrived; it's just not evenly distributed yet.

—William Gibson, science-fiction writer

Memory, handed out in contiguous chunks called buffers or arrays, is the scratch space for your processor. Your computer needs to read bitmaps and e-mails from the disk or network into memory in order to display or let you edit them. (Your operating system can actually cheat and pretend to give out lots of memory, and write to disk the infrequently used portions — virtual memory. If you come back to an application after not using it for several hours, you may find your hard drive is busy for several seconds because it is loading all of your data and code back into the real memory.)

Memory contains the current state of all the applications on your computer, and is therefore an extremely precious resource. Garbage collection is a system that transparently manages the computer's memory and automatically reclaims the unused “garbage” from the programmer when he is done using it. I will spend the rest of the chapter explaining how this technology changes programming, so sit tight if my explanation doesn't make sense yet.

John McCarthy created GC in 1959 when he created Lisp, a language invented ten years before C, but which never became accepted into the mainstream:

```
(defun finder (obj vec start end)
  (let ((range (- end start)))
    (if (zerop range)
        (if (eql obj (aref vec start))
            obj
            nil)
        (let ((mid (+ start (round (/ range 2)))))
          (let ((obj2 (aref vec mid)))
            (if (< obj obj2)
                (finder obj vec start (- mid 1))
                (if (> obj obj2)
                    (finder obj vec (+ mid 1) end)
                    obj))))))))
```

*The binary search function written in Lisp is a simple algorithm for quickly finding values in a sorted list. It runs in  $\log_2(n)$  time because at each step, it divides the size of the array in half, similar to how we look up words in a dictionary. There are many faster and more complicated algorithms, and search is a very interesting and large topic in computer science, but 99% of the time, ye olde binary search is good enough.*

C and C++ was based on BCPL and other languages before it, none of which had garbage collection. Lisp is a language built by mathematicians rather than operating systems hackers. Lisp pioneered GC, but also was clean and powerful, and had a number of innovations that even C# and Java don't have today.<sup>3</sup>

Wikipedia's web page doesn't explain why Lisp never became accepted for mainstream applications, but perhaps the biggest answer is performance.<sup>4</sup> So instead, people looked at other, more primitive, but compiled languages. The most expensive mistake in the history of computing is that the industry adopted the non-GC language C, rather than Lisp.

- 
- 3 Perhaps the next most important innovation of Lisp over C is functional programming. Functional programming is a focus on writing code which has no side effects; the behavior of a function depends only on its parameters, and the only result of a function is its return value. Nowadays, in object-oriented programming, people tend to create classes with lots of mutable states, so that the behavior of a function depends on so many things that it is very hard to write correct code, prove it is correct, support multiple processors manipulating that object at the same time, etc. Functional programming is a philosophy, and Lisp made it natural.
- 4 Most Lisp implementations ran 10x slower than C because it was interpreted rather than compiled to machine code. It is possible to compile Lisp, but unfortunately, almost no one bothered. If someone complained about Lisp performance, the standard answer was that they were considering putting the interpreter into hardware, i.e. a Lisp computer. This never happened because it would have sucked Lisp into the expensive silicon race.

While Lisp had many innovations, its most important was garbage collection. Garbage collection requires significant infrastructure on the part of the system and is a threshold test for an intelligent programming language.<sup>5</sup>

Because so few of the most important codebases today have adopted GC, I must explain how it improves software so my geek brethren start using it.

The six factors of software quality are: reliability, portability, efficiency, maintainability, functionality, and usability; I will discuss how GC affects all of these factors. The most important factor is reliability, the *sine qua non* of software.

---

5 Some argue that static type checking (declaring all types in the code so the compiler can flag mismatch errors) is an alternative way of making software more reliable, but while it can catch certain classes of bugs, it doesn't prevent memory leaks or buffer overruns.

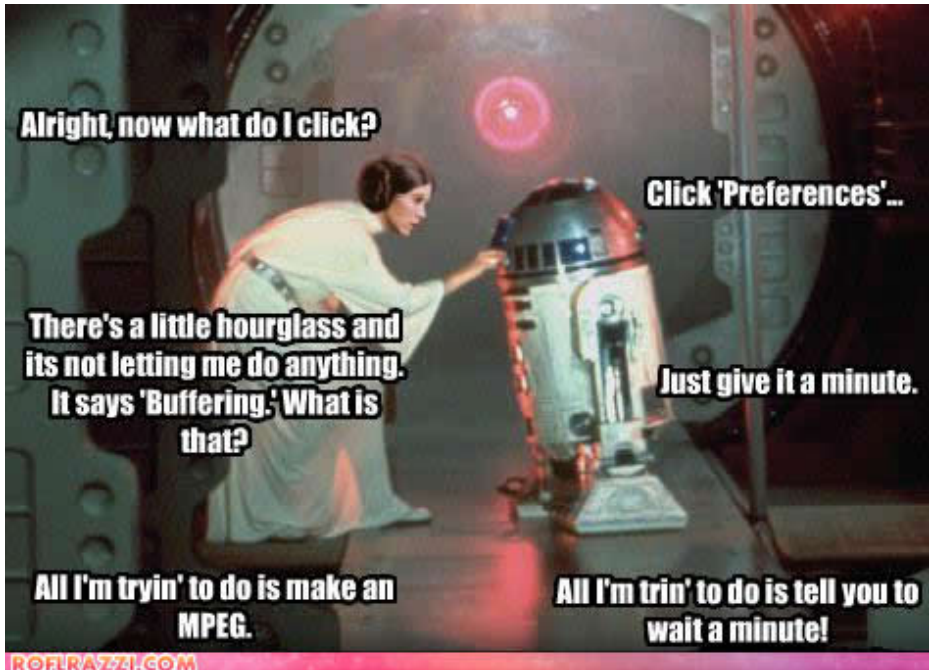
Likewise, there are "smart pointers" which can emulate some of the features of garbage collection, but it is not a standard part of languages and doesn't provide many of the benefits.

Apple's Objective C 2.0 has added support for GC in their language, but it is optional, and therefore doesn't provide many of the benefits of a fully GC language, like enabling reflection or preventing buffer overruns.

# Reliability

Therefore everyone who hears these words of mine and puts them into practice is like a wise man who built his house on the rock. The rain came down, the streams rose, and the winds blew and beat against that house; yet it did not fall, because it had its foundation on the rock. But everyone who hears these words of mine and does not put them into practice is like a foolish man who built his house on sand. The rain came down, the streams rose, and the winds blew and beat against that house, and it fell with a great crash.

—Matthew 7:24-27



*If the software of Star Wars was no more reliable than today's...*

Reliability, according to Wikipedia, is “the ability of a device or system to perform a required function under stated conditions for a specified period of time.” In software, the specified period of time is forever, because unlike metal and other things containing atoms, it doesn't wear out.

For example, reliability means that the computer does the same things for the same inputs every time. If you create a function to add two numbers, it should always work for all numbers. If your computer gets hacked and doesn't respond to your input, it isn't reliable. Even performance is related to reliability: if your computer is wasting CPU cycles and is no longer responsive, it isn't reliable.

Today, few would say that computers are completely reliable. Maybe your cable box crashes: "Comcast: On Demand, Sometimes." Maybe your laptop doesn't recognize the wireless network in one particular Internet cafe. I have trained my family to know that whenever their computer is misbehaving, they should reboot the system before calling me. This fixes any errors where the memory is in an unspecified state, which is the most common computer problem.

Reliability is important because software is built up as a series of layers. At one level is the code displaying a file open dialog box, and far below is a device driver reading the file you selected as a series of scattered data blocks on the hard drive. Every layer of software needs to be reliable because it is either being used by other software or by a human. Both humans and software need a reliable foundation.

A tiny bug in its software caused the crash of one of the European Space Agency's Ariane 5 rockets, costing \$370 million:<sup>6</sup>



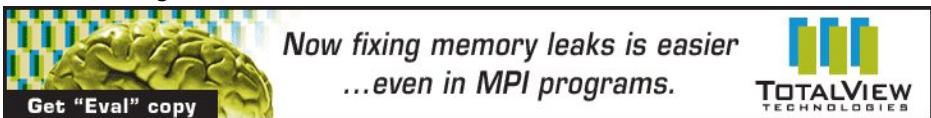
- 6 The rocket's software was written in Ada, an old language, but with many of the features of garbage collection. Code which converted a 64-bit integer to a 16-bit integer received a number too big to fit into 16 bits, and so the conversion code threw an exception. The code to handle this exception was disabled, and therefore the computer crashed. When this computer crashed, it started sending confusing diagnostic information to the flight control computer, causing it to fly in a crazy way and break apart, triggering the internal self-destruct mechanisms. Many blame management, but this was a simple design bug (you should be very careful when throwing away data). This was compounded because they were using a specialized embedded system with a non-mainstream programming language which allowed them the capability of disabling certain exceptions. This bug could have been caught in testing, but they didn't use accurate trajectory information in the simulations. Perhaps clumsy tools made it hard to modify test cases, and so they never got updated.

When your computer crashes, you can reboot it; when your rocket crashes, there is nothing to reboot. The Mars Spirit and Opportunity rovers had a file system bug which made the rovers unresponsive, nearly ending the project before they even landed!<sup>7</sup>

While it isn't usually the case that a software bug will cause a rocket to crash, it is typically the case that all of the software layers depending on that buggy code will also fail. Software reliability is even trickier than that because an error in one place can lead to failures far away — this is known in engineering as “cascading failures.” If an application gets confused and writes invalid data to the disk, other code which reads that info on startup will crash because it wasn't expecting invalid data. Now, your application is crashing on startup. In software, perhaps more than in any other type of intellectual property, a bug anywhere can cause problems everywhere, which is why reliability is the ultimate challenge for the software engineer.

Perfect reliability is extremely hard to achieve because software has to deal with the complexities of the real world. Ultimately, a key to reliable software is not to let complexity get out of hand. Languages cannot remove the complexity of the world we choose to model inside a computer. However, they can remove many classes of reliability issues. I'm going to talk about two of the most common and expensive reliability challenges of computers: memory leaks and buffer overruns, and how garbage collection prevents these from happening.

## Memory Leaks



*Web banner ad for a tool to find memory leaks. There is a cottage industry of tools to fix problems which exists only because the programming language is broken in the first place.*

One of the most common forms of memory corruption code is memory leaks — a common source of computer unreliability and frustration for users that worsens as our code becomes more complicated.<sup>8</sup>

<sup>7</sup> The rover file system used RAM for every file. The rovers created a lot of system logs on its trip from Earth to Mars, and so ran out of memory just as they arrived!

<sup>8</sup> The problem is even worse because every big C/C++ application has its own memory allocators. They grab the memory from the OS in large chunks and manage it themselves. Now if you are done with memory, you need to return to the person who gave it to you.

Losing the address of your memory is like the sign outside a Chinese dry-cleaner: “No tickie, no laundry.” To prevent leaks, memory should be kept track of carefully. Unfortunately, C and C++ do not provide this basic feature, as you can allocate and lose track of memory in two lines of code:

```
byte[] p = new byte[100]; // p points to 100 bytes of memory
p = NULL;                // p now points to NULL, reference
                           // to 100 bytes lost
```

*“New” returns the location of the newly allocated memory, stored into variable p. If you overwrite that variable, the address of your memory is lost, and you can't free it.*

Writing code in C or C++, in which all unused memory is returned to the operating system, is both complicated and tiresome:

```
Person* p = new Person("Linus", "Torvalds", 1969);
if (p == NULL) //Out of Memory or other error
    return;

Person* p2 = new Person("Richard", "Stallman", 1953);
if (p2 == NULL) //Out of Memory or other error
{
    delete (p); //Cleanup p because the p2 failed
    return;
}

MarriageLicense* pml = new MarriageLicense(p, p2)
if (pml == NULL) //Out of Memory or other error
{
    delete (p); //Cleanup p and p2
    delete (p2);
    return;
}
```

*Code in C or C++ to manually handle out-of-memory conditions and other errors is itself bug-prone, adds complexity, and slows performance.*

As programmers write increasingly complicated code, the work required to clean it up when things go wrong becomes more difficult. This small example has only three failure cases and the code to remedy these error conditions makes the code twice as complex as it would otherwise be. Furthermore, code which only executes in unlikely failure scenarios doesn't get executed very frequently, and is therefore likely to have bugs in it. A rule in software is: “If the code isn't executed, it probably doesn't work.” And if your error handling code doesn't work, your problems will accumulate faster.

Scale this dilemma into millions of lines of interdependent code written by different people and the complexities compound beyond



our ability to fix them. To date, there is no non-trivial codebase written in C or C++ which is able to solve all of these error conditions, and every codebase I saw at Microsoft had bugs which occurred when the computer ran out of memory.<sup>9</sup>

MySQL, an otherwise highly reliable database, which powers popular websites of Yahoo! and the Associated Press, still has several memory leaks (and buffer overruns.) Firefox's bug list contains several hundred, though most are obscure now.<sup>10</sup>

Let's take a look at why a memory leak can't happen when running on a GC language:

```
byte[] p = new byte[100]; // Variable "p" points to 100 bytes.
p = NULL;                // p now points to NULL.
                          // The system can deduce that no variables
                          // are referencing the memory, and therefore
                          // free it.
```

*You don't have to call "delete" because the system can infer what memory is in use.*

After the second line of code executes, "p" no longer references the 100 bytes. However, a GC system is smart, and it can take inventory of what memory is in use, and therefore discover that because these 100 bytes are not being referenced, they can be freed. Likewise, if the Chinese laundry knew you had lost your ticket, they would know to throw away your clothes.

*It is this simple innovation that changes programming.* Memory is such a critical part of computers that we need to have the system, not the programmer, keep track of it.<sup>11</sup>

9 Often near the end of a development cycle, after fixing our feature bugs, we would focus on some of the out-of-memory bugs. While we never fixed them all, we'd make it better and feel good about it.

It is true that when you run out of memory, it is hard to do anything for the user, but not causing a crash or a further memory leak is the goal.

10 Here is a link to all active MySQL bugs containing "leak":

<http://tinyurl.com/2v95vu>. Here is a link to all active Firefox bugs containing "memory leak": <http://tinyurl.com/2tt5fw>.

11 GC makes it easy for programmers to freely pass around objects that more than one piece of code is using at the same time, and the memory will be cleaned up only when every piece of code is finished with it. C and C++ do not enable this and many other common scenarios.

To write code which allows two pieces of software to share memory and to return it to the operating system only when both are finished is complicated and onerous. The simplest way to implement this feature in C/C++ is to do reference counting: have a count of the number of users of a piece of memory. COM, and the Linux and Windows kernels have reference counting. When the last user is finished, the count turns to zero and that last user is responsible for returning the memory to the OS. Unfortunately, this feature requires complicated nonstandard code (to handle multiple processors) and places additional burdens on the programmer because he now needs to keep track of what things are referencing each

It is also quite interesting that GC enables a bunch of infrastructure that Microsoft's OLE/COM component system tried to enable, but COM did it in a very complicated way because it built on top of C and C++, rather than adding the features directly into the language:

| COM Feature Name   | .Net & Java Feature Name  |
|--------------------|---------------------------|
| Reference Counting | Garbage Collection        |
| BSTR               | Unicode strings           |
| Type Libraries     | metadata + bytecodes / IL |
| IUnknown           | Everything is an Object   |
| IDispatch          | Reflection                |
| DCOM               | Remoting and Web Services |

*COM contains a lot of the same infrastructure that GC systems have, which suggests a deep similarity of some kind. Doing these features outside the language, however, made writing COM more tedious, difficult, and error-prone. ".Net" completely supersedes COM, and in much simpler packaging, so you will not hear Microsoft talk about COM again, but it will live on for many years in nearly every Microsoft codebase, and many external ones.*

## Buffer Overruns

As bad as memory leaks are because they often cause crashes, buffer overruns are worse because your machine can get hijacked! Buffer overruns are the most common type of security bug, and a major nemesis of the computer industry. Microsoft's code has fallen prey to a number of buffer overruns; the Code Red virus, which infected Microsoft's web server and caused rolling outages on the Internet, is estimated to have cost the industry two billion dollars. Free software is certainly not immune to this either; on a daily basis my Ubuntu operating system downloads fixes to newly discovered buffer overruns.<sup>12</sup>

Like with memory leaks, you can create and overrun a buffer with just two lines of code:

```
int* p = new int[50]; //Allocate 50 entries, referenced 0-49  
p[50] = 7;           //Write to 51st entry, "off by 1" bug
```

*C and C++ do not validate memory access, so a programmer can intentionally or unintentionally read or write to memory he shouldn't have access to.*

---

other. Handing out references, which should be simple, is now error-prone. Even so, reference counting is insufficient: if two objects point to each other, but no one points to them, they will keep each other alive.

12 One of the security fixes awaiting me right now is: "SECURITY UPDATE: arbitrary code execution via heap overflow," from CVE-2007-3106.

If hackers find a piece of code which doesn't validate a buffer length, they can send to that computer, not the usual data of an e-mail or picture, but a carefully crafted block of evil *code*, causing the system to start executing the hacker's software. A buffer overrun exploit is like a virus in the biological world: imagine cells which would unquestioningly incorporate any DNA they happened upon.

In addition to knowing what memory is in use, a GC system knows the size of every buffer, and so can validate all reads and writes to memory. The computer, not the error-prone programmer, makes sure memory doesn't ever leak or become corrupt. *Garbage collection is a necessary, if insufficient, precondition of the better code that we seek.*

While GC is necessary for reliability, it provides many other advantages.<sup>13</sup>

---

13 The Linux kernel is an example of reliable C code. Many might use this as proof that it is possible to write reliable code without garbage collection. However, there are several reasons why this lesson may not be valid: The kernel's primary job is to provide glue code to make the hardware work. Seventy-five percent of the kernel's code is hardware specific, and much of the code is tiny, simple components. Much of the remaining code implements decades-old operating system concepts like threads. This makes the feature requirements and design relatively stable over time. All of the things we consider smart software, like grammar checkers and speech and handwriting recognition, involve writing code which has no clear design and would never be part of a kernel. The kernel doesn't have the need to inter-operate with as much other software like applications do. Software written by many disparate and interdependent teams makes GC more important. The Linux kernel has a big team per line of code compared to other codebases. This gives them the luxury of using inefficient tools. The Linux kernel does have a few memory leaks even as I write this. (Bugs 5029, 6335, 8332, 8580) The kernel has a number of specialized requirements which make it harder to do garbage collection, but it would benefit from it, and it wouldn't surprise me if most of the kernel's code which is not written in an assembly language is written in a GC language one day. For now, we should focus on porting application code and leave the Linux kernel as the very last piece of code written in C. Once the world has collaborated on a new software stack, written in our new programming language, we can knock on Linus's door with our now mature tools and figure out what his requirements are.

# Portability

Fifteen years ago, Intel, with HP's help, decided to create a new processor architecture which would enable them to incorporate all they had learned in creating their x86 processors, first introduced in the 1970s. This chip, known as Itanium or IA-64, is a 64-bit chip which removed all of the previous compounded ugliness, leaving a simpler and therefore potentially faster design in its place. Microsoft was already working on porting its C tools and code to IA-64 when I joined in 1993, though the chip wasn't released until 2001:



*Beautiful new chip, but even though Intel included x86 compatibility hardware, it was still incompatible with existing C and C++.<sup>14</sup> Code written in a GC language is already portable to chips that have yet to be created.*

In spite of the years of R&D and Intel's enormous resources, the chip is hardly used today, even on servers. The reason this chip hasn't yet taken off is literally a billion-dollar question, causing Sun founder Scott McNealy to dub it the "Itanic".<sup>15</sup>

---

<sup>14</sup> Building emulation in hardware was not a good idea because you cannot mix 32-bit and 64-bit code because you cannot hand a 64-bit pointer to 32-bit code. You would have to boot the computer into a 32-bit mode where it never created a 64-bit memory address in order to let all code to run, and such a feature would defeat the purpose of building a 64-bit OS.

In 2006, Intel removed their x86 hardware interpreter, which ran the code 10 times slower than native, and replaced it with software that would dynamically recompile the x86 binary to Itanium instructions. The code still couldn't interoperate with true 64-bit code, but it could run at full speed, and it lowered the cost of building an Itanium. It is ironic that Intel had put into its hardware precisely what it was trying to get rid of!

<sup>15</sup> The two most common theories for the failure of IA-64 are: performance benchmarks which weren't significantly better than Intel's current processors, and

The biggest obstacle Intel faced was the fact that our pyramid of C and C++ code running on PCs today is compiled for the x86 processor. Such programs won't run without at least re-compiling the source for another processor, and it might even require changes to the software because it is easy to write non-portable C/C++. Consequently, the adoption of new hardware is significantly limited by the fact that you have to find every piece of code out there, and recompile it. What this ends up meaning is that while a lot of your stuff works, some of it doesn't. Itanium Linux had no Flash player or Adobe Reader till very recently, two significant stumbling blocks for desktop deployments, and even one obstacle can be too many.<sup>16</sup>

GC solves portability issues because programs written in languages such as Java, C#, Python, etc. are no longer compiled for any specific processor. By comparison a C/C++ executable program is just a blob of processor-specific code containing no information about what functions and other metadata are inside it. Its contents are completely opaque to the system, and the processor just starts blindly executing it. To a GC system, a blob of binary code is insufficient.

Like stores in the real world, GC systems in principle need to close to take inventory. However, unlike stores, they cannot just kick all the existing customers out, or wait for them to leave. So it does the equivalent of locking the doors, not letting any customers in or out (halts execution of code), and tabulating what is on the shelves and in the shopping carts (what memory is in use). Once it has an accurate account and it can then re-open the doors and let existing customers leave, and new ones enter (program execution resumes.)<sup>17</sup>

---

incremental mechanisms to add 64-bit computing to x86 without starting from scratch and creating something 100% incompatible.

16 If that code were free software, it would have been ported already by a geek just trying to get his hardware to work. However, if the code were written in a modern GC programming language, the issue of porting our pyramid of code wouldn't exist.

17 A more precise analogy is that it doesn't let any customers put new things into their cart until the inventory is done. It also needs to tag shopping carts as having been scanned so that it doesn't take inventory of a cart twice.

When GC pauses code, it needs to know what function the processor is currently executing, and even *where* in that function it is:

```
void ExampleFunction()
{
    int x = SquareNum(3); //If execution stops here, no memory
                          //allocated yet.

    object o = new object(); //This allocates memory into 'o'.
    DoStuffWithObject(o); //If execution stops here, 'o' is in use.

    int y = SquareNum(4); //If execution stops here, 'o' is no
                          //longer in use, and can be cleaned up.
}
```

*GC programs in principle need to know what your processor is doing at each moment in time to precisely inventory memory.*

The garbage collector needs to be able to know what objects are “live” in the system at every moment in time, but this depends on exactly what line of code the processor is executing when the inventory process is taking place. A C or C++ executable isn't required to have this sort of information available, but GC requires a rich understanding of the code in order to do its job.<sup>18</sup> Therefore, when a program is distributed for a GC language, it is delivered either in source form, or converted to a bytecode, a more efficient format to parse, but with very little loss of information compared to the original source.<sup>19</sup>

```
.assembly helloworld {}
.method public static void MyMain() cil managed
{
    .entrypoint
    ldstr "Hello, World!"
    call void [mscorlib]System.Console::WriteLine(string)
    ret
}
```

*“Hello, World!” in .Net's bytecode. This is similar to the original C#, though more verbose.*

In (all of the common) GC systems, the programmer ships source code or bytecode, not a machine-specific binary blob. If all code

<sup>18</sup> The Boehm GC doesn't require metadata, so it has to guess what is a pointer by walking the stack and heap and looking for bit patterns that look like pointers! Therefore, because Boehm isn't true GC, it doesn't enable things like reflection. Also, Boehm can't compact memory.

<sup>19</sup> That GC basically requires you to ship your source code seems like a hint from the Author of computer science that free software is a good thing.

written for the Macintosh was written in a GC programming language, it would have been zero work for Apple and third-parties to switch to the Intel processor once the GC runtime was ported!<sup>20</sup>

In fact, an application written in a GC programming language is automatically portable to chips that haven't even been created yet. We impede our freedom to create new processors when software is not written in portable languages.

Portability is one of the holy grails of computing, and while GC code doesn't completely solve cross-operating system portability, it does solve the situation of running the same code on different processors — itself an enormous step.<sup>21</sup>

With the source code or bytecode, the GC system has all the information it needs to figure out exactly what is going on when it stops execution. In fact, it also has a lot of information that enables other cool features like reflection, which allows code to query information about an object at runtime. These features create a more dynamic system.

We've discussed two advantages of GC: greater reliability and portability. The next topic is code performance, which is the biggest worry when using modern tools. I have had many discussions with smart geeks who insisted that languages such as C# simply weren't suitable for their “fast code.”

## Efficiency

It doesn't matter how fast your code runs if it doesn't give the correct result, but processing power is still an important resource. In fact, code efficiency is even less important than memory usage because if you waste memory, your computer will eventually crash, but if you waste processor cycles, your computer will just be sluggish and annoying. Microsoft has always focused on performance, and has often promoted it as a competitive advantage.<sup>22</sup>

If you walk up to a programmer on the street and ask them what they think of Java, one of your answers will be: “slow.” At one time,

---

20 Today, Mac users have to worry about whether a program is a PowerPC binary or an Intel binary. There is even a rumor that one day there will be four binaries, two for the 32-bit and 64-bit versions of both processors!

21 If we all use the same OS, then this OS cross-platform problem disappears :-)

22 GC code can give better performance because it has the exact hardware in front of it. C compilers are forced to generate generic code which will run on all models of a processor. New processors aren't just faster than previous versions, they add new instructions and other capabilities, which often go unused.

Netscape started working on a web browser written in Java, but they abandoned the effort after one year because of performance worries, and before the Java runtimes got fast.

Languages with garbage collection manage more details for you compared to “unmanaged” code, and this by definition adds a performance cost to your application. The need to pause a program while taking inventory of memory is one of the most common reasons cited for not using automatic memory management.<sup>23</sup> Add to this a bias: programmers, from their first class in computer science, are taught that analysis of algorithms revolves primarily around an analysis of code performance.

Both pieces of code below are less than 20 lines and look fairly similar, but the one on the left is significantly faster:

|  |   |
|--|---|
| <pre>static void quicksort(int[] a, int l, int r) {     int i, j;     char x, y;      i = l; j = r;     x = a[(l+r)/2];  do {     while((a[i] &lt; x) &amp;&amp; (i &lt; r)) i++;     while((x &lt; a[j]) &amp;&amp; (j &gt; l)) j--;          if(i &lt;= j) {             y = a[i];             a[i] = a[j];             a[j] = y;             i++; j--;         }     } while(i &lt;= j);      if(l &lt; j) quicksort(a, l, j);     if(i &lt; r) quicksort(a, i, r); } }</pre> | <pre>static void bubblesort(int[] items) {     int i;     int j;     int temp;     int x = items.Length;      for( i = (x - 1); i &gt;= 0; i-- )     {         for( j = 1; j &lt;= i; j++ )         {             if( items[j-1] &gt; a[j] )             {                 temp = items[j-1];                 items [j-1] = items[j];                 items [j] = temp;             }         }     } } }</pre> |
|--|---|

*Algorithm analysis teaches you that the code on the left should be about 50,000 times faster than the code on the right at sorting one million numbers. The speed of code, not the speed of the language is what matters.*<sup>24</sup>

23 You only need to *in principle* pause a program while doing GC. The good news is that in many types of applications, from word processors to web servers, a brief pause is acceptable. This number is proportional to the amount of memory in use by the application, and therefore obeys the “only pay for what you use” rule of engineering.

However, in certain situations, pausing is not acceptable, such as video games or code interacting with hardware. There are many solutions such as implementing GC via reference counting, or making it incremental and scheduling it proactively during idle moments, etc.

24 To sort an array of  $n$  numbers, Quicksort will do it in a time proportional to  $n * \log_2(n)$  and Bubblesort will do it in  $n^2$ . If you plug in one million for  $n$ , it means



Performance is rightfully important because the difference between fast code and slow code can mean the difference between milliseconds and hours. However, I spent years analyzing performance bottlenecks in various codebases at Microsoft, and every performance problem was caused by inefficient algorithms or bad design, not the speed of the language. Fast code written in C would be fast in a GC programming language, and slow code written in a GC programming language would also be slow in C.<sup>25</sup> The first lesson in the “Bible of Computer Science”, Donald Knuth's compendium of software algorithms, is that the speed of code is determined by the algorithms it incorporates.<sup>26</sup>

In the early days of C, you could ask an assembly language programmer what he thought of C, and he'd have said that it was “too slow.” No one says that today, however, because of C's 30 years of maturation, the dramatic progress of hardware, and because its significant other advantages overshadowed its early lackluster performance issues. My first group at Microsoft built FoxPro, which was for many years the fastest PC database, and they quit writing code in assembly language the day they re-wrote a bit of it in C and found it faster!

Java and C# have some performance issues today, but the primary reason is that most of the community is still expending energy continuing to optimize C/C++ tools, let alone exploring the many ways to GC systems unavailable to the older languages. Even so, let's assume that modern tools slow down code by 20%.<sup>27</sup> Taking a one-time 20% hit is worth it because our current software is ineffectively, or wastefully, using today's processing power. My pedestrian laptop is a now-standard dual-core computer, and it requires only

---

that the fast code will be about 50,000 times faster than the slow code.

I wrote some [code](#) to compare Mono's `Array.Sort()` to bubble sort, and found that the fast algorithm was 10,316 times faster than the slow one, which is the same order of magnitude as 50,000.

- 25 Firefox will get faster in certain ways when it is re-written in a modern programming language. One of the performance boosts will come when it throws away its Javascript interpreter. If Firefox ran Javascript at compiled native speed, it would be a significant performance boost.

Another example: Google's web spiders hit nearly all of my web pages nearly every day, even though I add new entries only on a monthly basis, and my articles do not change after they have been posted. The best way to make their servers use less processing power is not to do micro-optimizations, like making it 10% faster at parsing HTML, but to make it smarter about the big things, like not fetching and parsing unchanged web pages in the first place.

- 26 Knuth's *Art of Computer Programming* is about as hard to read as the King James Bible because it uses a made-up, primitive programming language.

- 27 That many focus on this 20% means they are focusing on a mere constant and ignoring the order of magnitude analysis which should be the primary focus.

one of the cores running at 2% usage when I type text as fast as I can into this book; even though I am working on a long document, it is redrawing the screen, autocorrecting, spell-checking, updating document statistics, etc. Sometimes CPU usage noticeably peaks in OpenOffice, but that's usually because it is redrawing the screen over and over, or doing some other useless task.

Today's software is primitive, i.e. it cannot do a processor-consuming task of intelligent analysis of my writing — rather, the computer is mostly just sitting there. It knows to correct “teh”, but not much more. The computer industry should focus less on not wasting a few CPU cycles and more on dreaming up new ways of using them to make our lives easier and better. Our stupid software still has a long way to go.

# Maintainability

The major incentive to productivity and efficiency are social and moral rather than financial.

—Peter Drucker

During the years we worked on Viaweb I read a lot of job descriptions. A new competitor seemed to emerge out of the woodwork every month or so. The first thing I would do, after checking to see if they had a live online demo, was look at their job listings. After a couple years of this I could tell which companies to worry about and which not to. The more of an IT flavor the job descriptions had, the less dangerous the company was. The safest kind were the ones that wanted Oracle experience. You never had to worry about those. You were also safe if they said they wanted C++ or Java developers. If they wanted Perl or Python programmers, that would be a bit frightening — that's starting to sound like a company where the technical side, at least, is run by real hackers.

We were always very secretive about our competitive advantage of Lisp. Robert Morris says that I needn't be because even if our competitors had known, they wouldn't have understood why: "If they were that smart they'd already be programming in Lisp."

—Paul Graham, *Hackers and Painters*

I worked on a team where one of our developers spent weeks integrating an HTTP stack written in C into our C codebase. In Java or C#, this basic functionality is already there, but even if it weren't, it would take mere hours to integrate such a library.<sup>28</sup>

For weeks after I started programming in C#, I would write code with a grin on my face, and my fellow co-workers also learning C# understood why. With a modern language, one finds that by having the system manage many of the details, more time is spent thinking about higher-level software issues.

Sun Microsystems researchers, in a paper called "FreeTTS – A Performance Case Study", analyzed this relationship between the productivity and performance of two speech synthesis engines — one written in Java, named FreeTTS, and one in C, named Flite:

When we started our study of the performance characteristics of a speech synthesis engine programmed in the Java programming language, our expectations were that it would hopefully be able to run nearly as fast as the native-C counterpart.

Through using some straightforward optimizations and relying

---

<sup>28</sup> It took so much time because C libraries often use their own string classes, synchronization primitives, error handling schemes, etc. Furthermore, modern languages do a better job at many little details like backward compatibility. If you reorder the elements of a data structure written in C, you need to recompile all code which uses it.

on the aggressive optimizations performed by the Java HotSpot compiler, we were pleased to find that FreeTTS runs two to four times faster than its native-C counterpart, Flite.

Clearly, it would be possible for us to roll some of these optimizations back into Flite with the likely result of improving Flite's performance to levels similar to FreeTTS. The lack of Java platform features such as garbage collection and high-performance collection utilities, however, makes performing these optimizations in Flite much more time consuming from a programming point of view.

The programmer's time is finite and is thus the limiting factor in software engineering. One could say that the only software quality that matters is maintainability, as that gives you the time to focus on every other aspect. We need garbage collection for, but the payoff is increased maintainability, and it will pay for the transition costs.

Many people in the software industry no longer believe that programming languages affect productivity. This misconception exists primarily because C++ didn't end up being significantly more productive than C. The debate is mostly between C and C++. Quoth Linus:

Quite frankly, even if the choice of C were to do nothing but keep the C++ programmers out, that in itself would be a huge reason to use C.

C++ leads to really really bad design choices. Developers invariably start using the 'nice' library features of the language like STL and Boost and other total and utter crap, that may 'help' you program, but they cause infinite amounts of pain when they don't work and inefficient abstracted programming models.

At the same time, few argue that assembly language had close to the productivity of C, so this contradiction is unresolved in the minds of many. Many computer geeks like to argue about C versus C++, but compared to a modern and elegant language like C#, this is like choosing between Britney and Paris.

While C++ added object orientation features to C, it had a fatal flaw: it was a superset of C. In fact, the early compilers just converted their code to C, which was a great way to bootstrap the new language to the many places where C was already used, but by being a superset it tied the designers to the baggage of C. While C++ added new object-oriented features, it also added significant complexity.<sup>29</sup> I used C++ for many years, and I liked some of the

---

<sup>29</sup> Another way to analyze the maintainability of code is to analyze the maintainability of the compilers for that code. One finds that C++ compilers are big, ugly and slow because the languages are big and complicated. If the tools for a language are complicated, it is also likely the code written in them will have maintainability

improvements over C, but the language is mind-numbingly complicated and generally provides many more ways to screw up than in C. That C++ didn't start with a clean slate is the second biggest mistake in the history of computing.

If you could double developer productivity at the cost of half of your current performance, would you take it? There isn't any universal agreement on the answer to this question among computer engineers today. However, Moore's law states that Intel's computers take merely 18 months to become twice as fast, and in ten years, computers will be another 100 times faster than they are today. A 20% drop in performance to enable garbage collection would take Intel hardware progress 4 months to counter-balance, and we would pay it only once. Anders Hejlsberg, architect of C#, has said it is the best use of Moore's law to come around in years.

Reliability issues also add a variability to the engineering cost of a project because developers spend an unpredictable amount of time fixing bugs. In adopting GC, developers would pay a fixed performance cost in exchange for decreased engineering costs and variability.

## Functionality and Usability

Functionality and usability are “soft” features — lower priority amongst the other software quality factors. If your computer corrupts your files while saving them to disk, you don't care how rich or usable the application was up to that point.

Many consumer devices today are often not very usable because the people building those devices write 100% custom code, often in assembly language. It takes a lot of effort merely to get the software working, and once it is, the device manufacturers become fearful of changing anything lest they break it.

One of my groups at Microsoft, called Spot, ran C# interpreted on a \$50 watch. The code was efficient because we focused on the high-level performance issues that you never seem to get to when you write in C or C++. In fact, the processor ran at just a few percent on average. Our code was so fast that it hardly ever ran, which was great for battery life! The watch failed in the marketplace, but it was reliable, rich, and easy to use. We were able to tweak the UI based on feedback right up until the end simply because we could

---

issues. It is perfectly possible to build a language which looks nothing like C, but whose compiler emits C. My favorite C++ gotcha is the need to make destructors virtual. One of the best description of C++'s flaws is at <http://yosefk.com/c++fqa/defective.html>.

without breaking things. Modern tools can go everywhere, even to devices constrained by cost and size, and their greater use will make consumer devices easier to use.

## Conclusion

We can only see a short distance ahead, but we can see plenty there that needs to be done.

—Alan Turing, father of modern computer science

Their policy on the problem of maintenance was but a game they seemed to be playing with a piece of rubber that could be stretched a little, then a little more.

—Ayn Rand, *Atlas Shrugged*

Phew — thanks for making it through all that! I had to explain GC because the fact that it is missing from so much code is the biggest reason why programs crash and people don't trust computers today. When software, the most fundamental building block of mankind's 21<sup>st</sup> century intellectual activity, protect themselves with copyleft, and use a standard GC programming language, mankind will enter a shining age. Perhaps we only need free software, but I don't think we can ever get there with C/C++, which are missing an important piece of technology invented 50 years ago.

Software, like math, is a pure world. Any aliens who have invented software will have created the same Quicksort algorithm we have. One of my mentors at Microsoft, Eliyezer Kohen, designed some of the best, and most complicated, code at Microsoft. He showed that code should be written as carefully as a mathematician constructs a proof. The compiler, the operating system, and everything else is simply the external machinery created to run your perfect software. His thoroughness left a strong impression on me, but better tools make it easier to write and share code, no matter what language you use. In fact, with better languages and runtimes, you can make your code “perfect” faster.

If Microsoft were to move its important code to C# and the free software world were to fail to move their code away from C/C++, Microsoft could leave them in the dust. In general, the task of converting the free software codebases to better tools is much easier than porting Microsoft's code because the free codebases are ten years younger, and much smaller and simpler. In fact, Microsoft's codebases are so big and old that I'm not sure they could port them, and if they did, they'd likely break backward compatibility so badly that users wouldn't necessarily switch to the new version. If they did

provide perfect compatibility, they'd build something almost as complicated as what they have today, and they wouldn't be able to divorce themselves from 20 years of legacy.

Switching to modern tools could be a huge competitive advantage for the free software community. The tremendous inefficiency of the free software community is its greatest weakness, and tools play an enormous part of that, and is a big part of how millions of free software programmers can continue to lose to Microsoft's thousands.

Upgrading to modern tools is a task that few of the important teams in free software are even considering. Every free software group needs to create a five-year plan and start today. We must do this for reliability reasons, but we will receive many other benefits. What language should they choose? That is the subject of the next chapter.

# THE JAVA MESS

I think everybody hates Java as a desktop thing. I see Java mentioned a lot lately, but all of the mentions within the last year have been of Java as a server language, not as a desktop language. If you go back a year and a half, everybody was talking about Java on the desktop. They aren't anymore. It's dead. And once you're dead on the desktop, my personal opinion is you're dead. If servers are everything you have, just forget it. Why do you think Sun, HP — everybody — is nervous about Microsoft? It's not because they make great servers. It's because they control the desktop. Once you control the desktop, you control the servers.

It's no longer something that will revolutionize the industry. It could have revolutionized the industry if it was on the desktop, but I don't see that happening anymore. I hope I'm wrong. Really. I just don't think I am.

—[Linus Torvalds](#), 1998

A company should build a process that systematically looks at every product, every service, every process, every policy, every market with the question, "If we weren't doing this already, knowing what we now know, would we start it, would we go into it?" If not, how quickly can we get out?

—Peter Drucker

```
class JavaProgram
{
    public static void main(String args[])
    {
        int counter = 1; //Add up all the integers from 1 thru 10
        int sum = 0;      //Store value in variable sum

        while (counter <= 10)
        {
            sum = sum + counter;
            counter = counter + 1;
        }
        System.out.println("Numbers 1-10 add to: " + sum);
    }
}
```

*Java is relatively elegant, and should have replaced C and C++. The highlighted portions show the very few places that would need changing to port to C#.*

**I**n 1995, Sun Microsystems created a next-generation programming language called Java, something that could have been the most significant part of their legacy, more important than their Sparc processor, Solaris (their flavor of Unix), or anything else. Java



had the potential to change the software industry by becoming the next generation C/C++, but now I think it is a dying language. (Java is similar in name to Javascript, the programming language of web browsers, and both are similar to C, but the languages and runtimes are incompatible and different.)

At first glance, Java looks quite like C, and very much like Microsoft's C#. Sun's Java adopted much of the syntax and semantics of C but added object orientation and many other language innovations, including garbage collection, and they made no effort to be 100% backwards compatible like C++ did.

While Java has many important technical advancements, it has achieved only a small fraction of the universal status it should have. Java should have replaced C and C++, languages desperate to be taken out back and shot! However, Java did not, and one of the biggest reasons why software is in shambles today is because Sun repeatedly screwed the pooch with Java.

In mid-2006, after Scott McNealy's departure as CEO, Sun began making Java free, attempting to solve some of the issues mentioned in this chapter, but Java grew up as proprietary and this has left indelible scars on it. I will discuss why making it free won't change things at the end of the chapter.

For those not in the software business, it is hard to describe the ire many programmers have felt towards Sun. This frustration is rooted in the special part tools play in the life of a software developer. A programmer is a tinkerer; he tinkers with his own tools as much as he concentrates on the job at hand. Software tools are continuously refashioned. If I need to process columns of data in a text file I receive from the Internet, it is almost as easy to fashion a tool to accomplish the task as it is to do it by hand. (A spreadsheet would be a useful tool, but there are other possible ones. A web server isn't going to load a spreadsheet to extract and process simple data from a text file.) A suitable tool might not exist, or might not be exactly suitable to the particular task until it is tinkered with and made into the proper tool for the job.

## Sun locked up the code

Brilliance is typically the act of an individual, but incredible stupidity can usually be traced to an organization.

—Jon Bentley

Sun's first mistake was that they failed to do what Bell Labs did with C and C++: create freely available compilers and runtimes for people to experiment with and extend. Instead, Sun locked up the Java codebase, letting few see it, and letting even fewer improve it, so that today, there exists only a small community of people, outside of Sun itself, improving Java.

For example, because Sun locked up their code, no one was able to port it to other processors. As I mentioned in the Linux chapter, Debian calls itself “The Universal Operating System” because it contains 18,000 software components that run on 15 different processor architectures:

**Intel x86 / IA-32**  
**AMD64**  
**Motorola 68k**  
**Sun SPARC**  
**Alpha**  
**Motorola/IBM PowerPC**  
**PowerPC 64-bit**  
**ARM**  
**MIPS CPUs**  
**HP PA-RISC**  
**IA-64**  
**S/390**  
**SuperH**  
**Big-Endian ARM**  
**Renesas's 32-bit RISC**

*Debian supports 15 processors, but Sun's Java web page lists just four.*

A wide variety of platforms allow the programmer to decide which processor is the best choice for his product. By contrast, on Sun's website, you can only download four architectures: x86, PowerPC, AMD64 and Sparc.

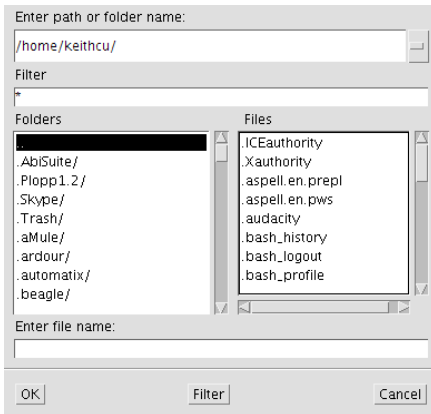
The fact that Java doesn't offer broad support on hardware platforms is a significant consideration. Why invest in a programming language that might not be usable with future hardware? C, and other free languages, provide tremendous hardware flexibility in their widespread adoption.

Flexibility and adaptability of programming tools is especially important for embedded devices because, unlike the PC environ-

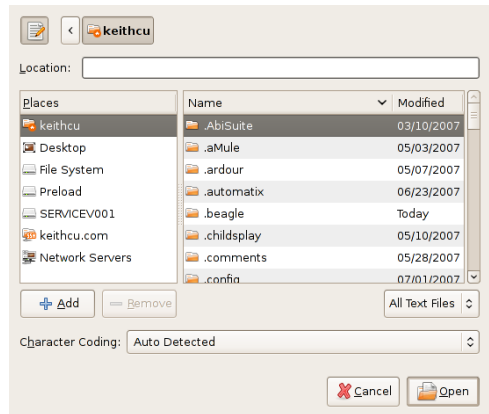
ment where processors and operating systems have similar functionality, the software must be customized for embedding on low-end hardware, something that Sun did not enable.

For its first five years, when everyone was seriously considering using it, Java ran ten times slower than C because it was interpreted rather than compiled, exactly like Lisp. This would have been fixed a lot faster if Sun had involved and encouraged the existing free compiler development community.

Even in 2008, Java programs on my computer don't look like they should. For example, when a program asks to display a file chooser dialog box to the user, below are the results for a Java and a native application:



*Java file chooser on Linux*



*Standard Linux file chooser*

*How does Java not look native in 2008? Let us count the ways.*

Java had many significant limitations for many years because all progress was held up by Sun.

## Sun obsessed over specs

A “spec” is close to useless. I have *never* seen a spec that was both big enough to be useful *and* accurate. And I have seen *lots* of total crap work that was based on specs. It's *the* single worst way to write software, because it by definition means that the software was written to match theory, not reality.

So there's two MAJOR reasons to avoid specs:

1. They're dangerously wrong. Reality is different, and anybody who thinks specs matter over reality should get out of kernel programming NOW. When reality and specs clash, the spec has zero meaning. Zilch. Nada. None.  
It's like real science: if you have a theory that doesn't match experiments, it doesn't matter *how* much you like that theory. It's wrong. You can use it as an approximation, but you **MUST** keep in mind that it's an approximation.
2. Specs have an inevitable tendency to try to introduce abstractions levels and wording and documentation policies that make sense for a written spec. Trying to implement actual code off the spec leads to the code looking and working like CRAP.

The classic example of this is the OSI network model protocols. Classic spec-design, which had absolutely *zero* relevance for the real world. We still talk about the seven layers model, because it's a convenient model for *discussion*, but that has absolutely zero to do with any real-life software engineering. In other words, it's a way to *talk* about things, not to implement them.

And that's important. Specs are a basis for *talking about* things. But they are *not* a basis for implementing software.

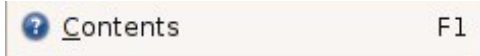
So please don't bother talking about specs. Real standards grow up *despite* specs, not thanks to them.

—Linus Torvalds

Imagine designing a car by a committee on a computer, and then sending the design directly to the assembly line without building a car first and letting anyone test-drive it.

Sun didn't let anyone read or tinker with their code but they wanted to be seen as a community-oriented effort, so, they created the Java Community Process where human-readable specifications describing Java were produced. Unfortunately, they focused on the specs and not on releasing software for people to try out so that the feedback came years later, or not at all. The biggest feedback

that Sun would have received, but did not, was how insanely complicated their Java specs were. As an example, here is what a menu item looks like on the screen:



and here are the 456 functions a Java MenuItem class implements:<sup>1</sup>

```
action, actionPerformed, add, addActionListener, addAncestorListener, addChangeListener, addComponentListener,
addContainerListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addImpl, addInputMethodListener,
addItemListener, addKeyListener, addMenuDragMouseListener, addMenuKeyListener, addMouseListener, addMouseMotionListener,
addMouseWheelListener, addNotify, addPropertyChangeListener, addVetoableChangeListener, applyComponentOrientation,
areFocusTraversalKeysSet, bounds, checkHorizontalKey, checkImage, checkVerticalKey, clone, coalesceEvents, computeVisibleRect,
configurePropertiesFromAction, contains, countComponents, createActionListener, createActionPropertyChangeListener,
createChangeListener, createImage, createMenuItem, createToolTip, createVolatileImage, deliverEvent, disable,
disableEvents, dispatchEvent, doClick, doLayout, enable, enableEvents, enableInputMethods, equals, finalize, findComponentAt,
fireActionPerformed, fireItemStateChanged, fireMenuDragMouseDragged, fireMenuDragMouseEntered, fireMenuDragMouseExited,
fireMenuKeyPressed, fireMenuKeyReleased, fireMenuKeyTyped, firePropertyChange, fireStateChanged, fireVetoableChange,
getAccelerator, getAccessibleContext, getAction, getActionCommand, getActionForKeyStroke, getActionListeners, getActionMap,
getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBackground, getBaseline, getBaselineResizeBehavior,
getBorder, getBorders, getChangeListener, getClass, getClientProperty, getColorModel, getComponent, getComponentAt,
getComponentCount, getComponentGraphics, getComponentListeners, getComponentOrientation, getComponentPopupMenu, getComponents,
getComponentZOrder, getConditionForKeyStroke, getContainerListeners, getCursor, getDebugGraphicsOptions, getDefaultLocale,
getDisabledIcon, getDisabledSelectedIcon, getDisplayedMnemonicIndex, getDropTarget, getDropTarget, getDropTarget, getFont, getFontMetrics,
getFocusListeners, getFocusTraversalKeys, getFocusTraversalKeysEnabled, getFocusTraversalPolicy, getFont, getFontMetrics,
getForeground, getGraphics, getGraphicsConfiguration, getHeight, getHideActionText, getHierarchyBoundsListeners,
getHierarchyListeners, getHorizontalAlignment, getHorizontalTextPosition, getIcon, getIconTextGap, getIgnoreRepaint,
getInheritsPopupMenu, getInputContext, getInputMap, getInputMethodListeners, getInputMethodRequests, getInputVerifier,
getRegisteredKeyStrokes, getRollOverIcon, getRollOverSelectedIcon, getRootPane, getSelectedIcon, getSelectedObjects, getSize,
getSubElements, getText, getToolkit, getToolTipLocation, getToolTipText, getTopLevelAncestor, getTransferHandler, getTreeLock,
getUI, getUIClassID, getVerifyInputWhenFocusTarget, getVerticalAlignment, getVerticalTextPosition, getVetoableChangeListeners,
getVisibleRect, getWidth, getX, getY, getYocus, grabFocus, handleEvent, hasFocus, hashCode, hide,
imageUpdate, fireMenuDragMouseEntered, isBorderPainted, init, insets, inside, invalidate, isAncestorOf, isArmed,
isBackgroundSet, isContentAreaFilled, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusCycleRoot,
isFocusOwner, isFocusPainted, isFocusTraversable, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, isFontSet,
isForegroundSet, isLightweight, isLightweightComponent, isManagingFocus, isMaximumSizeSet, isMinimumSizeSet, isOpaque,
isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isPreferredSizeSet, isRequestFocusEnabled, isRollOverEnabled,
isSelected, isShowing, isValid, isValidDateRoot, isVisible, keyDown, keyUp, layout, list, locate, location, lostFocus,
menuSelectionChanged, minimumSize, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, notify,
notifyAll, paint, paintAll, paintBorder, paintChildren, paintComponent, paintComponents, paintImmediately, paramString,
postEvent, preferredSize, prepareImage, print, printAll, printBorder, printChildren, printComponent, printComponents,
processComponentEvent, processComponentKeyEvent, processContainerEvent, processEvent, processFocusEvent,
processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent,
processMenuDragMouseEvent, processMenuKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent,
putClientProperty, registerKeyboardAction, remove, removeActionListener, removeAll, removeAncestorListener,
removeChangeListener, removeComponentListener, removeContainerListener, removeFocusListener, removeHierarchyBoundsListener,
removeHierarchyListener, removeInputMethodListener, removeItemListener, removeKeyListener, removeMenuDragMouseListener,
removeMenuKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removeNotify,
removePropertyChangeListener, removeVetoableChangeListener, repaint, requestDefaultFocus, requestFocus, requestFocusInWindow,
resetKeyboardActions, reshape, resize, revalidate, scrollRectToVisible, setAccelerator, setAction, setActionCommand,
setActionMap, setAlignmentX, setAlignmentY, setArmed, setAutoscrolls, setBackground, setBorder, setBorderPainted, setBounds,
setComponentOrientation, setComponentPopupMenu, setComponentZOrder, setContentAreaFilled, setCursor, setDebugGraphicsOptions,
setDefaultLocale, setDisabledIcon, setDisabledSelectedIcon, setDisplayedMnemonicIndex, setDoubleBuffered, setDropTarget,
setFocusable, setFocusCycleRoot, setFocusPainted, setFocusTraversalKeys, setFocusTraversalKeysEnabled,
setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont, setForeground, setHideActionText, setHorizontalAlignment,
setHorizontalTextPosition, setIcon, setIconTextGap, setIgnoreRepaint, setInheritsPopupMenu, setInputMap, setInputVerifier,
setLabel, setLayout, setLocale, setLocation, setMargin, setMaximumSize, setMinimumSize, setMnemonic, setModel,
setMultiClickThreshold, setName, setNextFocusableComponent, setOpaque, setPreferredSize, setPressedIcon,
setRequestFocusEnabled, setRollOverEnabled, setRollOverIcon, setRollOverSelectedIcon, setSelected, setSelectedIcon, setSize,
setText, setToolTipText, setTransferHandler, setUI, setVerifyInputWhenFocusTarget, setVerticalAlignment,
setVerticalTextPosition, setVisible, show, size, toString, transferFocus, transferFocusBackward, transferFocusDownCycle,
transferFocusUpCycle, unregisterKeyboardAction, update, updateUI, validate, validateTree, wait
```

The list of functions the Swing Java MenuItem class implements, with bizarre names like “isFocusTraversalPolicyProvider” and “addVetoableChangeListener”. Imagine if you needed to become familiar with 456 things to use your oven.

Such bloated code is hard to understand and even harder to make reliable. And this is just the Swing class library's opinion of a menu item, one of three popular implementations of Java widgets, and supposedly an efficient one!

<sup>1</sup> Other widget libraries, like the Gtk#’s MenuItem have hundreds functions as well, but many do not and Java is the worst. The WxWidgets MenuItem class has 50 members, and it uses native widgets on Mac, Windows and Linux. Apple’s NSMenuItem has 55.

Java is plagued by too much complexity as you can see, and releasing specs instead of code is a big cause of this problem. In fact, many of the Java specs aren't widely used because they do not meet developers' needs, or they became heavily amended after they finally got into customers' hands, which was often years after the specs were originally written.<sup>2</sup>

Java was not growing carefully and organically, as Linux has done, by following the “ship early, ship often” methodology. A lot of people who hate Java do so because of its pervasive extraneous complexity.

## Sun locked up the design

When it created Java, Sun didn't want to release the code because they were worried people would tweak their code, tweak it slightly to better meet their needs, and creating incompatibilities that would hurt the “brand” of Java — confusing a 99% friend with an enemy. This paranoia occurred because Sun made a rookie mistake in not understanding that a language needs to be extensible.

In the C and C++ world, there is even a convention that any non-standard keywords would be prefaced with two underscores, like this: “\_\_inline”. In some cases, a feature started out as a vendor-specific addition, was adopted by others, and eventually became added to the language in an official way. The creators of C and C++ didn't have the hubris to believe they had built a perfect language.

Even more importantly, the primary way to extend a computer language is by creating new functions and data types, which was also not allowed in any of the domains specified by Sun.

Sun got it backwards — people will only make minimal changes necessary to Java to make it work for them. That is what happened in C and C++. Sun didn't have faith in its customers and didn't understand the realities of how large bodies of systems-level code are created.

---

2 Two good examples of basically unused class libraries are:

1. JXTA, Java's peer-to-peer protocol that standardizes many aspects of P2P but is not used because it solves the wrong problems, and,
2. J2ME, Java's underpowered class library for mobile devices which has primitive widgets, is unable to access a phone's address book, calendar, camera, GPS, or web browser, and has no ability to make phone calls. J2ME's goal should be rich enough to build an entire phone user interface in Java, but it only has the ability to make toy applications, and even then, programmers often complain that the differences between the implementations make it a “write once, test everywhere” scenario.

## Sun fragmented Java

Because of fear, Sun decided to lock up their code to prevent it from being extended by anyone else. This short-sighted idea, but even worse, the law of unintended consequences came up and bit them in the tush. Because Sun made only the specs but not the code available, many third parties started producing their own Java runtimes so they could control their own destiny. Wikipedia documents 34 different third-party Java runtimes, and their list doesn't include any of the embedded Java implementations, which are harder to count.

Everyone who wanted to use but improve Java would have to start from scratch. This created much bigger areas of incompatibility because building a Java implementation from scratch requires a lot of work, and they could even unknowingly cut corners.

The biggest problem with so many runtimes, besides the wasted efforts, is that the imperfect specs became the reference rather than the code because there wasn't an agreed upon codebase containing the official Java design.

Backward compatibility is the greatest cause of complexity, and therefore unreliability, in computers today. As described in the Linux chapter, one of the Linux kernel's major innovations is that it avoids backward compatibility issues by encouraging all code into one tree. This makes it much easier to make changes to the design.

With an official source tree, Java could become whatever we need it to be, avoiding the mess of different codebases, and serving as a reference when the paper specs are ambiguous. It sounds chaotic, but things quickly converge, and on something with minimal excess complexity, because it can be removed.

While the Java specs were not good because feedback came years later, they became etched in stone because once 30 runtimes have a particular piece of functionality, it becomes very hard to change them all. Even if Sun let you alter their design, it might take years to make any changes. In contrast, if the Linux kernel is missing a feature, you can send some code to Linus and have it in the official codebase immediately.

# Sun sued Microsoft

Another Sun mistake was getting into a war with Microsoft over Java. Microsoft's tools division was an active participant in Java for its first several years, and PC Magazine wrote in 1997 that: "Microsoft's Java environment was the fastest and most compatible on our tests."

When Microsoft improved Java in certain ways that by definition were incompatible, Sun sued them. Microsoft's changes to Java were minor, the features justifiable, and the incompatibilities almost non-existent.<sup>3</sup> A public lawsuit, the exchange of \$20 million in damages, and much bitterness ensued.

Eventually, Microsoft quit supporting Java and went off to create C# instead. C# might be considered Java 2.0. While it is very similar to Java, it had five years of refinement and applied Microsoft's many years of experience as a dominant force in the tools business. C# created unnecessary chaos and confusion in the industry and allowed Microsoft to be 100% incompatible with Java rather than just .01% incompatible. Microsoft would not have created C# if Sun hadn't sued them.

## Java as GPL from Day 0

If Sun had created Java with a GPL license, the landscape of the software industry would look completely different today, and computers would be more reliable and smarter. Java would be more reliable, simpler, richer, and faster, and a community, like C and C++, that Microsoft tools developers would have been a part of.

Many more programmers would have contributed to Java and built something much better. These mistakes stunted Java's growth, but even worse, it caused many software programmers just to con-

---

3 One of the things Microsoft changed was to simplify and improve the performance of how a Java developer called in to native operating system functionality. Java has always allowed developers to write non-portable, operating system-specific code when the operating system provided a feature that a developer wanted access to but that Java did not support. By definition this code was not cross-platform, so one would imagine that Microsoft-specific syntax to access Windows-specific features would not have been a big deal. In fact, the way a Java programmer would call the native functionality on Windows had the same syntax between Microsoft's RNI and Sun's JNI. It was only the way the native method was declared that was different. Microsoft did make other changes, such as adding a keyword, "delegate", but if you search the web you can find many people who wished Sun had added that feature and code samples to enable that feature in Java.



tinue to plod along in their C or C++ codebases. It gave rise to C# and to other free languages like Javascript, PHP, Python, and Ruby. These were created to fill niches that Java didn't meet.

The success of C was in part due to its widespread support, which made it easier to share code than did its predecessor on the PC — assembly language. It is hard to share code across languages, and the proliferation of new ones has gotten worse since Java was created, when it should have gone in the opposite direction.<sup>4</sup> Even within Sun, there is a group of people trying to build another programming language called Fortress, which will be suitable for scientific computing. They too believe Java is not sufficient as it is.<sup>5</sup>

---

4 Sharing code across languages has historically been very difficult. The search engine Lucene is a recent example of something which started in Java but has been forked into versions in PHP and C#.

There are many reasons for it being so difficult to share code between different languages. They have different ways of deploying the code, different naming conventions, and different low-level details. Interoperating layers usually adds significant performance costs because everything from the strings on up the pyramid have to be marshaled, which usually involves making a copy of the data. If you can't agree on what a string is, then you will have difficulty sharing code. (Going from C# to C/C++ is fast because a copy of the data doesn't need to be created — a C Unicode string is a subset of a C# string.)

5 Fortress's syntax is arbitrarily different from Java in ways not at all related to high performance computing (HPC). A programming language is by definition extensible by creating new functions and classes, and it is also possible to alter the runtime, without changing the language, to optimize it for specialized hardware, etc. The big challenge in HPC is to robustly divide up the work amongst a farm of servers, pass messages and maintain and monitor the system — new software, not new language features.

# Pouring Java down the drain

The Lord said, "If as one people speaking one language they have begun to do this, then nothing they plan to do will be impossible for them."

Genesis 11:6<sup>6</sup>



*Scott McNealy, co-founder and CEO of Sun Microsystems from 1984 – 2006, and an impediment to free software — even though his company made most of their money on hardware.*

Sun's first major decision after co-founder Scott McNealy's departure in mid-2006 was a promise to make much of their software, including Java, free. This announcement was big news and some

---

<sup>6</sup> My bible says God scattered the Babylonians because of their "pride." I can't know what they thought, but I do believe that learning about ourselves, using technology to save and improve lives, and attempting to understand the true nature of the world, only gives us more appreciation for what He has created. I believe the creator of liberty and science wants us to understand these concepts, as long as we don't lose track of our relationship with the creator somehow in the process.

think it marks the potential for a new day in the life of Java. However, this effort is starting more than a decade after Java's creation, arguably too late to fix most of the problems.

Even with Sun's new GPL license, there is a lot of Java code out there that is not free and cannot be made so by Sun. Apple, for example, took Sun's PC implementation and worked extensively to port it to Macintosh. This code isn't free, and it doesn't seem likely that Apple will ever make it such. There is a company making Java run on mainframes, and their enhancements are proprietary as well. Even within the code Sun has recently made free, there are pieces Sun licensed from third parties, and they are having to re-write it because they can't get permission to change the license.

Even a free Java on its current trajectory will not absorb other communities. C# is considered better than Java, so those programmers are not making a switch. There are millions of web pages written in PHP, and I don't see them rushing to switch to Sun's Java either. Sun's code has been locked up for so long, there are few people in the outside world able to contribute. Why should the community fix a mess that was created only because Sun didn't work with the community in the first place?

C# and .Net is the dominant platform on Windows, PHP is the dominant language on the Web, and Python and Mono are the most popular GC runtimes on the Linux desktop. So while Java is taught in universities, used for custom applications in enterprises, and has some success on mobile phones, these niches aren't enough to ensure its long-term viability. Java has accumulated so much baggage, only some of which I have discussed here, that I think the software community should abandon it. This would also take a big step in lessening the problem of too many programming languages. In earlier drafts of my book, I proposed Sun create a next generation programming language, but I now believe there already are suitable codebases: Mono and Python.

# Let's Start Today

All economic activity is by definition “high risk.” And defending yesterday — that is, not innovating — is far more risky than making tomorrow.

—Peter Drucker

When there's a will to fail, obstacles can be found.

—John McCarthy

Sometimes the real hurdle to renewal is not a lack of options, but a lack of flexibility in resource allocation. All too often, legacy projects get richly funded year after year while new initiatives go begging. This, more than anything, is why companies regularly forfeit the future -- they over invest in “what is” at the expense of “what could be.”

New projects are deemed “untested”, “risky”, or a “diversion of resources.” Thus while senior execs may happily fund a billion-dollar acquisition, someone a few levels down who attempts to “borrow” a half-dozen talented individuals for a new project, or carve a few thousand dollars out of a legacy budget, is likely to find the task on par with a dental extraction.

The resource allocation model is typically biased against new ideas, since it demands a level of certainty about volumes, costs, timelines, and profits that simply can't be satisfied when an ideal is truly novel. While it's easy to predict the returns on a project that is a linear extension of an existing business, the payback on an unconventional idea will be harder to calculate.

Managers running established businesses seldom have to defend the strategic risk they take when they pour good money into a slowly decaying business model, or overfund an activity that is already producing diminishing returns.

How do you accelerate the redeployment of resources from legacy programs to future-focused initiatives?

—Gary Hamel, *The Future of Management*

When software, the most important building block of mankind's intellectual property, uses a standard GC programming language and protects itself with copyleft, mankind will enter a shining age. I think better runtimes are critical to the success of Linux on the desktop.

Unfortunately, fixing this problem is a big task because inertia is such a powerful force. The good news is that the difficulty of porting to a different language is not the same in every case. Porting from one modern language to another is ten times easier than moving from C to C#. Right now, much of our code and infrastructure lives in the dark ages, and this first port is the hardest, but it will provide big benefits.

During my tenure at Microsoft, I witnessed numerous re-writes. Unfortunately, most of them failed. Those that failed had at least one thing in common: they usually were re-writing C code in C, or C++, and because they did not use significantly better tools as part of their plan, they never ended up saving any time in the end. It doesn't make sense to re-build something from scratch unless you plan to make it *a lot* better. But if the teams out there move from C or C++ to a much more productive language, they will save time in the end and it is much more likely to succeed — assuming they start.

The key is to make a plan and break down the problem. For example, the OpenOffice word processor could use the existing layout engine written in C and replace the UI with one written in a GC language. This first step would give the team expertise to help them make the subsequent steps. They also might find that a bold vision will bring in new contributors, especially if it is easier and more enjoyable to work on the new codebase. Whatever choice makes sense for each team, the important teams like Firefox, OpenOffice, and Gnome should agree on a language because that will bring along the rest of the Linux desktop.

Today, the most popular GC runtimes on the Linux desktop today are Mono and Python. Both are built by a worldwide community, but each has certain advantages and disadvantages.

One of Mono's biggest advantages is that it is compiled. While performance is not the most important factor of software quality, any language which wants to become the standard for Linux needs a compiler. Even if only a small fraction of any piece of software is performance-critical, the language doesn't know what that part is, and so it should run all code fast. Another advantage of Mono is that it comes with a complete end-to-end solution for creating programs: a graphical editor and debugger, code browser with auto-complete, etc. With Python, you must piece together these components yourself, and there isn't a standard one with critical mass.

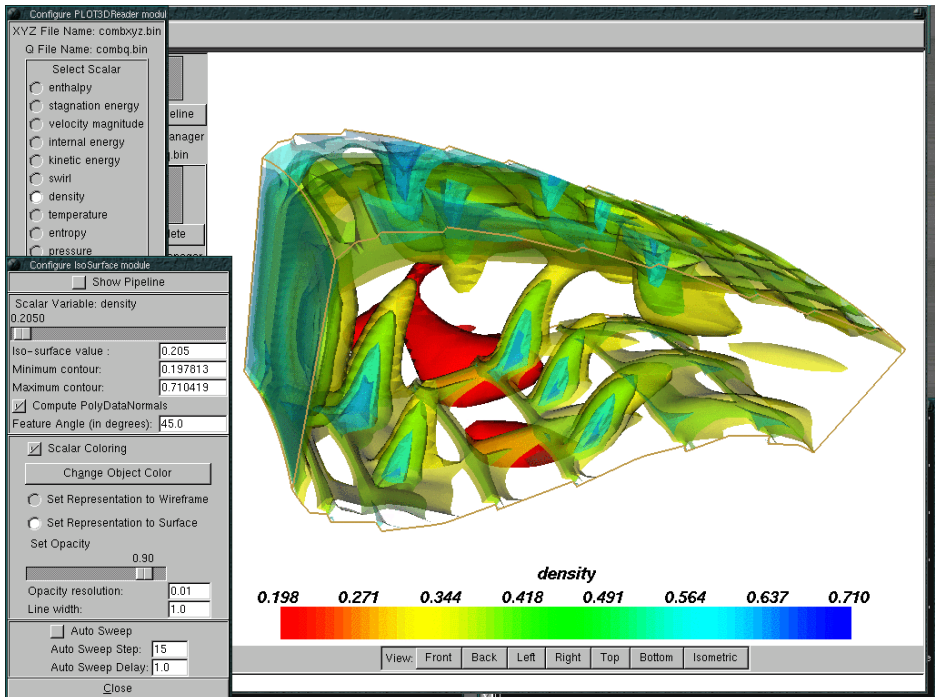
Mono's biggest disadvantage stems from the fact that it is an implementation of .Net which is a standard created by Microsoft.<sup>7</sup> The C# language and the core libraries are not created with much input from the worldwide community and the enhancements are

---

<sup>7</sup> There is also an enormous amount of FUD (fear, uncertainty and doubt) that surrounds Mono. Linux programmers are being irrational because the .Net specification is publicly available, and Microsoft has promised not to sue for implementations of .Net. This "cloud" has hung over Mono for eight years without any lawsuit and yet still the fear persists! Programmers should also realize that Mono extends the reach of .Net to other processors and operating systems, a common requirement of Microsoft's customers.

improved only on Microsoft's schedule. Microsoft has hired some of the best language designers on the planet, like Anders Hejlsberg, but he is not as smart as the combined knowledge of the rest of the languages designers on the planet. This has practical effects: C# is adding “dynamic language” features in the upcoming version 4.0 of C#, while Python has had these for many years.<sup>8</sup>

Python's advantage is that it is created with input from programmers all over the world, and has a larger set of libraries than any other GC language:



*A computational fluid dynamic (CFD) visualization of a combustion chamber. The Python community has created a wide variety of libraries for everything from gaming to scientific computing.*

Python's biggest disadvantage is performance: it does not compile and doesn't support multiple threads. These features are a requirement for any serious runtime and so one might one whether

<sup>8</sup> One other concern about Mono is that it doesn't use a copyleft license. Excerpt from the FAQ on the Mono website:

“When a developer contributes code to the C# compiler or the Mono runtime engine, we require that the author grants Novell the right to relicense his/her contribution under other licensing terms. This allows Novell to re-distribute the Mono source code to parties that might not want to use the GPL or LGPL versions of the code. Particularly: embedded system vendors would obtain grants to the Mono runtime engine and modify it for their own purposes without having to release those changes back.”

Python is still at a prototype phase of development. Even fans of Python very loudly complain that while it might be suitable for little scripts and application plugins, it is too slow for writing large applications.

These features are a big amount of work that are not even on Python's to-do list. And as the language has never been compiled there are even some questions as to whether it can be made fast.<sup>9</sup> In addition, Because the Python runtime is slow, the developers are forced to write a lot of support code in C, which further slows its development progress. Is it a shame that Python isn't fully using a tool which would have made their job easier — their own language. There are efforts such as the PyPy project which has built a Python compiler, in Python, which outputs C. Unfortunately, this piece of elegance is not yet the mainline codebase and is not considered for such.<sup>10</sup> And as mentioned earlier, the language doesn't have a standard graphical programming and debugging environment. So Python today has impediments for both casual programmers looking for an easy way to get into programming, and professionals who care about building high-performance applications.

There are other interesting languages and runtimes out there, but I believe the Linux desktop community should focus on these two. Some have proposed merging the Python language with the Mono runtime. Mono already supports other languages in addition to C#, some that even look like Python.

There are many good programming languages, in fact there are too many, but I also think they are mostly good enough. One could further tweak the letter of the English language to make it easier for your eyes to distinguish the letters, but it isn't necessary because what we have is good enough. Likewise, it is much more important to building a complete set of libraries for all aspects of computing, a Wikipedia of free code, than to worry that further language innovation is the gating factor towards future progress in software.

---

<sup>9</sup> In some performance tests, the IronPython implementation on .Net runs slower than the standard Python implementation. So even though IronPython has access to a compiler, it apparently isn't able to take advantage of it enough to be faster.

<sup>10</sup> One of the biggest missing features is that it doesn't support Python extension modules.

# CHALLENGES FOR FREE SOFTWARE

Justice officials argued that Microsoft's power was impregnable because consumers were so dependent on Windows. His voice rising, Gates exclaimed, "You give me any seat at the table: using Linux or Java I can blow away Microsoft!"

—*World War 3.0: Microsoft and its enemies*, Ken Auletta

The mode by which the inevitable is reached is effort.

—Felix Frankfurter, US Supreme Court Justice

**T**he last chapter described the state of the operating system market. This chapter talks about the challenges facing the free software community. Despite laboring since 1985, and its many successes with devices and servers, the free software community has yet to achieve a market share of more than 1% on the desktop, and this is the last and the biggest challenge.

As a side note, while Microsoft doesn't have a great reputation right now, especially because of Vista and malware, there is a lot of clever code and brilliant engineers inside the company. I was fortunate to learn from a bunch of great people in groups that fostered cultures of very high quality software.

When Microsoft got serious about Internet Explorer, Netscape's rag tag team of kids just out of college didn't have a chance. MS took some of their top engineers in text engines, networking, forms, internationalization, performance, and object models, and put together a large, world-class team. Microsoft's institutional knowledge of so many areas of software could be applied to any effort.

People forget all that, but this is why the reviewers consistently would say that Windows was better than OS/2, Microsoft Word was easier to use WordPerfect, IE was faster than Netscape, Excel was richer than Lotus 1-2-3 3, ad infinitum. PC Magazine wrote in 1997: "Microsoft's Java environment was the fastest and most compatible on our tests." Microsoft had built a better runtime than Sun, the creator of the language!

Many think Microsoft won all these battles merely because they somehow magically became a monopoly, but the atriums in the buildings at Microsoft have shelves overflowing with awards from independent reviewers. Microsoft succeeded in all these areas



because they used their financial success with DOS to hire an enormous army of smart programmers working in all areas of software, which could be applied to new areas, and passed on to new employees like me. And every time I switched groups, I'd take the knowledge and best practices I had learned in my previous groups.

I am not an apologist for Microsoft, and I think they are in deep trouble over the long-term, but the idea that Microsoft came to utterly dominate the computer industry and lay waste to countless companies was not mere chance. In fact, a big part of Microsoft's success was the strategic or technical incompetence of its competitors.

Free software has tremendous potential, but the community needs to execute better to win. In fact, until free software succeeds on the desktop, many will continue to question whether it is even viable.

## More Free Software

If you look at the magazines about the use of the GNU/Linux system, they're filled with ads for non-free software that you could run on top of the system. Those ads have a common message. They say: "Non-free software is good for you." They call these things "value-added packages," which makes a statement about their values. I call them "freedom-subtracted packages." Because if you have installed a free operating system, then now you are living in the free world. You enjoy the benefits of liberty that we worked for so many years to give you.

—Richard Stallman

One of the best ways to cure the problems of free software is simply to create more of it. If you talk to someone about their Linux experience, they might complain that they can't play DVDs out of the box, that some proprietary drivers are missing, or that iTunes or other proprietary software does not natively run on Linux.

Getting vendors to provide their proprietary software on Linux might sound like a great help, but building a completely free software stack should be the principal focus. Richard Stallman wrote:

Adding non-free software to the GNU/Linux system may increase the popularity, if by popularity we mean the number of people using some of GNU/Linux in combination with non-free software. But at the same time, it implicitly encourages the community to accept non-free software as a good thing, and forget the goal of freedom. It is no use driving faster if you can't stay on the road.

Some GNU/Linux operating system distributions add proprietary packages to the basic free system, and they invite users to consider this an advantage, rather than a step backwards from freedom.

Richard Stallman looks at free software as a struggle requiring constant vigilance, and he is correct. Linux is just like the real world in that respect: the parts that don't have freedom are the the worst parts. Ultimately, it is good news if the solution to your problem is to do more of what you're already doing as this means you are heading in the right direction, and success is possible.

In addition, proprietary software is generally an older, decrepit distraction. Microsoft's profitable products are simultaneously burdened by decrepit code, as is much of the industry's successful proprietary software. Adobe Photoshop is one of the most popular tools used by graphics designers today, and one of Adobe's [employees](#) wrote of Photoshop:

In “One Piece At a Time”, Johnny Cash tells the story of building a Cadillac from 20 years' worth of evolving, mismatched parts. I've gotta say, I know the feeling. Photoshop has been accreting power & users for the better part of two decades, and looking at some parts of Photoshop is like counting the rings in a tree: you can gauge when certain features arrived by the dimensions & style of the dialog. No one wants to work with — or work on — some shambling, bloated monster of a program.

I think proprietary software, if it becomes popular and long-lived, is destined to become a mess because it does everything by itself rather than leveraging free software components. Furthermore, it doesn't receive the constant tending that a garden the size of a city would require. If Adobe were to put a team of 100 developers on Photoshop to clean things up, it would no longer be the profitable product it is today, even with its \$1,700 price tag.

## Cash Donations

It is every man's obligation to put back into the world at least the equivalent of what he takes out of it.

—Albert Einstein

Drupal is a piece of free elegance to help manage the content of a web site. Like many software projects, it has a link where you can donate money via PayPal. While Drupal had 45,000 users in 2006, it received only \$1,000 in donations, for an average donation of 2 pennies per person. Free software, as Eric Raymond says, is about scratching an itch — fixing a problem that you care about. But what if you have the itch and some money but not the ability to scratch it?

The fact that much free software can flourish without money is amazing, but free software should have financial donations as a sustaining element, in addition to time.

Bounties, a collection of a large amount of money set aside towards a specific purpose, is another great idea that the free software community has explored but not widely adopted. If the eight million users of Ubuntu gave their one dollar towards bounties, and the bounties were set at \$5,000 each, this would create enough money for 1,600 projects. A major effort of Linux distributors could be to gather proposals, take donations from interested users, and manage the development. Doing this would motivate free software users to donate money because it would be applied directly towards projects users care about.

It is still early in the free software movement. The Linux kernel has thousands of contributors, but many other teams are not as well funded yet. An OpenOffice developer told me that this is the breakdown of developers from various companies:

Sun: 30

Novell: 15

RedHat: 1 to 2

Ubuntu: ~1

There are also RedFlag & IBM with large numbers of supposed contributors, but sadly with a large imaginary component ;-)

The people at Sun who are responsible for OpenOffice do not realize its importance to put such few developers on it. In spite of the fact that Sun has over 30,000 employees, Microsoft has twice as many programmers working on Internet Explorer alone as Sun has working on all of OpenOffice! I wouldn't work on something as big and complicated as OpenOffice without getting paid, which is why I donated money.

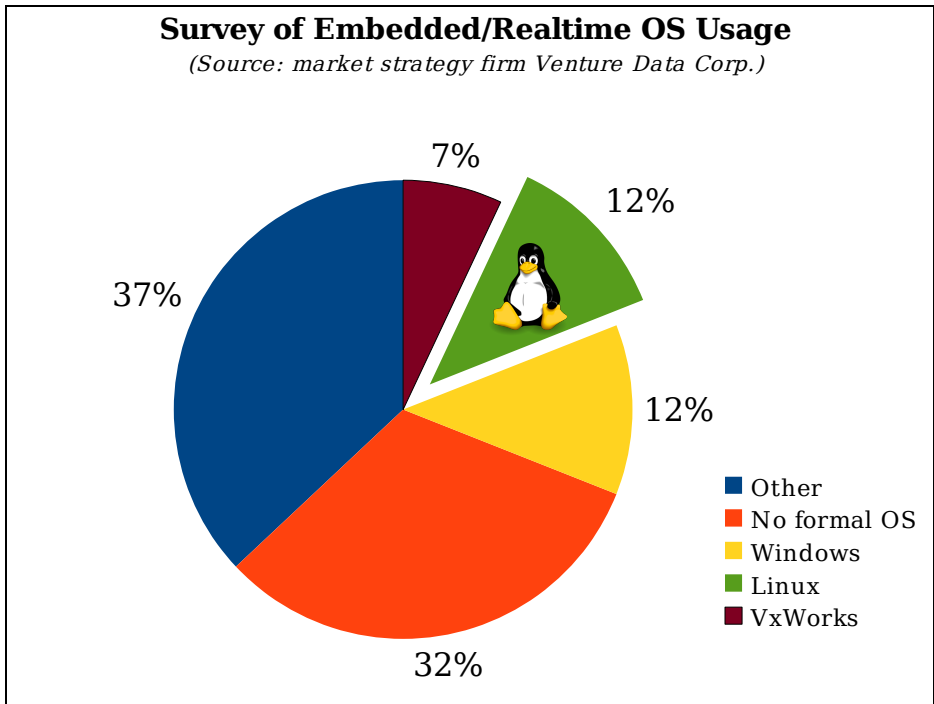
We are used to paying hundreds of dollars for proprietary software applications, and so we should be willing to invest smaller amounts for free software. I went on a guided tour of Mt. Saint Helens, and at the end the Forest Service Guide said that if you liked the tour, you should donate \$5, but if you didn't, you should donate \$20 — so they could improve the tour. That should be the attitude of the free software community: if you want more, contribute more. The nice thing about this model is that people can donate what they can afford. If you're a student or a business in India, the amount you can donate back for use of a free database is much less than if you are Amazon corporation and you have a farm of servers running the same software.

# Devices

The inside of a computer is dumb as hell but it goes like mad.

—Richard Feynman, 1984 (*Moore's law says this statement is 65,536 times more true today.*)

There are enormous markets for free software below the PC for which Microsoft has not built a dominant ecosystem, nor one with strong ties to the desktop. These markets should be much easier for free software developers to infiltrate and dominate:



*The fragmented state of embedded operating systems. This [survey](#) also showed that planned use of Linux is expected to increase 280% for the “next wave of projects.” However, that would still only give it 33% of the market.*

The proprietary OS Symbian has 70% of the high-end cellphone market but is not free software, and technically inferior to Linux, and therefore should be easy to replace. The good news is that ABI Research [forecasts](#) that by 2012, more than 127 million devices will be enabled with Linux, up from 8.1 million in 2007. Many of the proprietary embedded OSes enjoy their success primarily because of inertia.

Moore's law doesn't just apply to our computers, it will also ensure that small new devices will proliferate like mice in spring-time. And those mice will need software to make them interesting. One device I'm looking for is a robotic mouse for my cat:

home  
winningest  
kittens  
losingest kittens  
newest kittens  
add your kitten  
e-mail us



# kittenwar

Davis

Added to the site on May 23rd 2006



This is how Davis has performed in 5846 battles:

- ◆ Won: 2650 (45%)
- ◆ Lost: 2632 (45%)
- ◆ Drawn: 564 (10%)

But all our kittens are winners really...

[Start a new kitten battle!](#)

**Ads by Google**

**Kitten Ringtone**  
Send this ringtone to your phone right now!  
[RingRingMobile.com](http://RingRingMobile.com)

**2006 Cute Kid of The Year**  
Win Cash & Prizes - Free Membership  
Sign Up & Submit Your Photo Today!  
[www.TheCuteKid.com](http://www.TheCuteKid.com)

*Fierce soldier of cuteness on kittenwar.com.*

He's regal and mellow and while he happily enjoys lots of quality time on da cowsche and refrigerator, his hunting instincts are prolific.<sup>1</sup> I've purchased many toys for him to discover how he learns and plays, but while each one has provided a few minutes to a few hours of entertainment, none of the toys provides him the thrill and challenge of stalking and catching live prey.

One of Davis's favorite toys is a ping-pong ball. However, while it bounces around and can quickly disappear, it isn't alive and doesn't change direction under its own power. I can watch him calculate a path to pounce on a toy, realize it isn't a challenge for his prowess, sneer, and lose interest.

<sup>1</sup> As a bachelor who has spent too much time with computers, I've relied on Davis to teach me lessons about how to better interact with humanoids. When I come home at night, tired and ready to relax, my cat is just waking from his day-long slumber and is ready to play. If I ignore him, he will persistently meow at me, letting me know he is bored and wants to be entertained; living inside an apartment simply doesn't provide enough stimulation for him.

I'd like a robotic mouse that is quiet, agile and fast. Maybe when he catches it, the mouse could open a hatch and release some food. It would be nice to come home from work and be able to sit on the couch and commiserate with my cat about how exhausted we both are from earning our meal.

Better toys are in our future, and so are safer ones. My sister has a beautiful Husky, an interesting animal because, like my sister, it has no sense of awareness of its location. Huskies are bred for endurance and strength and can head in any given direction for great distances but are never able to retrace their steps. I'd like to be able to give him a collar with GPS and a cellphone-like transmitter that could report back on his location. Anyone who has lost a pet understands the frustration. A similar thing could also be used for young children. A GPS-device would allow you to sleep soundly at night knowing you can always find your vulnerable family members.

These are just two of the many innovative devices I can imagine, and these devices require little new hardware technology beyond what we already have at our fingertips. Building such devices would cost hundreds of dollars today, but the relentless march of Moore's law suggests that the price will keep halving every 18 months; this combined with free software will make the future very interesting!

## Reverse Engineering

Imagine trying to fix a car without having any of the documentation produced by the car manufacturer. Much technology in the computing world today is undocumented, which means that a lot of what free software developers have to do is reverse-engineering, a tedious and time-consuming task. I don't believe the Publisher format is publicly documented, so someone would need to create lots of little files and look through the on-disk binary to figure out the particulars of how formatting is stored on disk. It is sort of amazing that millions of people create files in formats that are not publicly described, yet this is the state of the industry today. Unfortunately, this means the user is enslaved to his software vendor, and arguably doesn't even truly own their documents.

If everyone used free standards, let alone free code, software progress would be faster. Today, if a standard is created, the community almost never gets any code to go with it. It might be best to have code be the official reference, rather than a written specifica-

tion, but today we often have neither. If the computing world ever breaks out of this vicious cycle, maybe we could focus our collective efforts on the truly hard challenges.

## PC Hardware

We love Linux, and we're doing our best to support the Linux community. we see the Linux desktop as a customer-driven activity. If customers want it, well, Dell will give it to them.

—Michael Dell

Michael Dell's quote demonstrates that he doesn't consider the situation where people aren't running Linux on his hardware because it doesn't work! I have a Dell Vostro laptop that doesn't have a Linux driver for wireless Internet. If I can't take the computer to a coffee shop and surf the web, it is useless. I thought about putting a bullet through the laptop and mailing it back.

PC hardware support in Linux is generally in good shape, and has improved a lot in the years since I first started using it. I believe it mostly requires that we continue to press on, working through the chicken and egg issue where hardware vendors are reluctant to support Linux until it has more users, but users won't run Linux if it doesn't fully support their hardware.

I singled out IBM in the OS chapter, but many other hardware companies are under-investing in free software even though public statements by their VIPs suggest that they believe in it. For example, Intel claims to be a strong support of Linux, but is doing only a decent job in its support of Linux drivers.<sup>2</sup> An Intel engineer told me at a Linux conference that their Linux efforts are just 1% of the manpower that their Windows efforts receive. Doubling their Linux development team would [cost](#) less than .1% of their total R&D. Intel is under-investing in Linux not because they can't afford to increase costs by .1%, but because they're suffering from Stockholm syndrome!<sup>3</sup>

---

2 My Intel wireless card resets and frequently re-associates or doesn't always resume properly. (These problems appear to now be fixed in 2009, but it took years.) In general, Intel's video drivers are considered slow, buggy, and behind: Intel added Linux driver support for TV-out years after the hardware was released. There are recent [news reports](#) that Intel's GMA 500 drivers are “a bloody mess.”

3 Wikipedia: “Stockholm syndrome is a psychological response sometimes seen in an abducted hostage, in which the hostage shows signs of loyalty to the hostage-taker, regardless of the danger in which they have been placed. Loyalty to a more powerful abuser — in spite of the danger that this loyalty puts the victim in — is common among victims of domestic abuse, battered partners and child abuse. In many instances the victims choose to remain loyal to their abuser, and choose not to leave him or her, even when they are offered a safe placement in foster homes.”

If Linux could recognize all of your hardware, all of your free software would run; if there is a bug anywhere in your hardware or device drivers, then it is quite possible that *no* software will run. Therefore, step one of World Domination by Linux is World Installation. The software incompatibilities will be better solved as soon as the hardware compatibilities become better solved. Therefore, it is the kernel that is currently holding up the PC revolution.<sup>4</sup>

I've installed Linux on a number of computers and found several problems: the fingerprint reader and other optional hardware often doesn't work, the computer doesn't always come out of sleep, sometimes the modem drivers aren't available, etc. None of my particular problems were a barrier, but others aren't so lucky. I recently tried to upgrade my dad's eMachines computer from Ubuntu 6.06 to 8.10 but the install failed because of APIC incompatibilities.<sup>5</sup> There are many long discussion [threads](#) on the Internet about Linux hardware incompatibilities.

Linus has overseen the design of an incredible platform, with support for more hardware than any other OS ever, and his kernel is the best piece of large free code ever written, but it needs further work. One of the biggest challenges I see for the kernel development community is a lack of respect for their buglist, the most important metric describing the state of the kernel.

---

4 The X Windows (video) subsystem is likely the next biggest obstacle. Video problems are especially tricky because it is hard to fix your computer when the display is not working! The external monitor stuff has been difficult for years and a UI for setting these options has only recently been added to Linux. Even the free drivers have some bugs, but the proprietary ones are years behind.

5 I even passed in "NOAPIC NOLAPIC" flags to the install program, but the install still did not work. I re-installed 6.06 and it worked fine, so this appears to be a kernel regression.



## Fix the F'ing Hardware Bugs!

Never, never, never, never give up.

If you are going through hell, keep going.

I am easily satisfied with the very best.

—Winston Churchill

At Microsoft, we had it beaten into our heads to fix bugs: a bug meant an unhappy customer, and a bug that affected just 1% of users meant that there were millions of unhappy customers!<sup>6</sup> Software that doesn't work is not worth anything.

As I write this, the Linux kernel has 1,400 active bugs, and the median age of those bugs is ten months — a number three times longer than its release schedule. In general, a bug, whenever it is found, should get a fix put in the next release. The Linux kernel is shipping frequently, but many bugs are taking three to four releases to get fixed.

Bugs get stale if they sit around for too long. If I file a bug against some device driver, and the owner of the code gets back to me one year later, I may not have all the necessary pieces set up to reproduce the bug again. Therefore, the bug must be resolved as not-reproduced and can get fixed only when the next unhappy customer finds it. Furthermore, the fact that a buglist already appears voluminous discourages people from adding to it.

One might believe that the Linux programmers are fixing bugs as fast as they can, but in fact they aren't. The Linux kernel development team has thousands of contributors and thousands of bugs, so fixing their bugs is a very manageable task — if they make it a priority.

The good news is that it is easier to fix bugs than to write code because writing code involves design, which in turn requires difficult choices to be made. (e.g. How do I add this feature?) In the bug-fixing phase of a software product cycle, most of the hard decisions have already been made, so it is mostly a matter of making small tweaks — the architectural equivalent of changing a blueprint to move a sink a few inches.

---

<sup>6</sup> You might not believe that, but I have the early bad reviews to prove it! Microsoft's software is considered unreliable today not because the developers don't care, but because they are burdened by old code, and have a development organization far too small for the vast scope of technology they release. They have perhaps 10,000 developers in total, whereas the Linux kernel alone has 3,000 developers.

Fixing these bugs might be tedious, especially when the developer doesn't have access to the hardware he is trying to debug, but everything about hardware is tedious so they may as well just do it now. PC hardware contains 10,000 devices, along with a tremendous amount of unnecessary complexity and undocumented designs, but the messiness of the PC has existed from the moment Linus first created his kernel. Linus, however, has the means to make all PC hardware work on Linux: lean on hardware vendors to support free software, and crack the whip on his kernel developers!

1,400 bugs for something as big and actively evolving as the Linux kernel is not a disaster at all. But if Linux kernel developers focused on the buglist for just a few months, the bug total could be brought down to less than 50. Linux's goal should be to fix 90% of new bugs in the next release, and 99% within two releases. This accomplishment would represent yet another breakthrough for a project of this size and complexity. Linux is a superior kernel to Windows today, but it needs a bit more work on the mess that is today's PC hardware.<sup>7</sup> The whip Linus must crack on his kernel developers is actually nothing more than a feather boa.

## Metrics

One of the great mistakes is to judge policies and programs by their intentions rather than their results.

—Milton Friedman

What's measured improves.

—Peter Drucker

Microsoft conducts yearly polls that give them statistically significant data on the state of the company. If they want to know whether developers were “happier” than they were five years before, they can get hard numbers on this. By gathering the right data and watching it over time, management can fix any organizational trends heading in the wrong direction.

This data was collected and managed by the Human Resources Department, but every other Microsoft team had metrics as well, and by far the most important was the bug count. The Microsoft buglist almost isn't even considered a metric because it is the driving force in the development process.

---

<sup>7</sup> In fact, if hardware companies did not finalize their hardware until their Linux driver was written, hardware itself would become more reliable. Most hardware today is designed before the software is written. When the the drivers finally get written, it can expose hardware bugs. Free drivers allow for better hardware and simpler software.

A lesson of success is the importance of metrics in driving change. Rudy Giuliani explained in his book *Leadership* that turning around New York City required a lot of hard work, but the key was to gather and publicize data about problems and track progress on them.

People might argue that data is a bureaucratic waste of time and rightly point out that by itself it doesn't change anything. What they are missing is that it gives people information which serves as motivation. When kernel developers see that the median age of Linux's bugs is ten months, they will realize that they should do better. This will motivate them to spend a bit more time working on the older bugs.

Every software team needs to find metrics that give good analysis of the state of their product over time. Metrics also help potential users of a piece of software; just knowing the total number of users and developers provides an excellent way to analyze the health and success of a component. The other nice thing about such a metric is that it is a self-reinforcing feedback loop that causes the fastest growing teams to grow faster.

## Volunteers Leading Volunteers

A manager's task is to make the strengths of people effective and their weakness irrelevant.

A manager sets objectives—A manager organizes—A manager motivates and communicates—A manager, by establishing yardsticks, measures—A manager develops people.

—Peter Drucker

Many of Debian's leaders were grad students hanging out in academia. One might wonder if someone working part-time can effectively lead thousands of people, but if you add up all the time corporate managers spend courting customers, creating budgets, and attending the required training on Corporate Values, one might find that very few spend a significant portion of their time thinking about the long-term big issues. It is very easy to get caught up in the day-to-day.

Because everything in volunteer organizations like Debians run on autopilot, the leader has the opportunity to focus on the big problems and be effective, even while working part-time. However, leading volunteers is a task as difficult as herding cats as you must coax each of them to do something.

There are benefits to a flat command structure: people are forced to convince others to do something based on the merits of their ideas. Because decisions are made on their merit, issues generally resolve themselves, with the person doing the work making the ultimate decision in a Do-ocracy. Everyone wants to do their best, and if they don't write correct code, bugs will come in, and developers will be forced to confront their mistakes.

In addition to metrics, task forces are a great tool of leadership. With small teams it is easier to get a consensus and produce a coherent document describing a problem and the proposed solution. Coming up with a proposal is easier than implementing it, but it can serve as a rallying point.

For example, Debian uses a lot of different tools to manage the history of the source code, with the number of tools increasing as Debian has grown. A task force could determine the recommended source control system. Such a recommendation wouldn't immediately change all the source control systems out there, but it would serve as a good default choice in the future, and it would push Debian, and the rest of the community, towards unification. Each task force needs to establish credibility and validity on its own to convince others in the loosely distributed meritocracy that is the free software movement.

## Must PC vendors ship Linux?

Many argue that hardware companies need to start shipping computers with Linux pre-installed before Linux will take off. While this is helpful, it is not necessary, and not sufficient.

The best first step for the hardware vendors would be to ensure that all of their hardware components have drivers in the Linux kernel tree. Once this has happened, it will mean that, in general, every distribution of Linux will run on every model of their machines. In many cases, different models use many of the same components, so supporting the 20<sup>th</sup> machine is often no more work than supporting the 19<sup>th</sup>.

Once the drivers are in the kernel tree, there is no need for a hardware vendor to pre-install Linux because the user can do it themselves. Windows is installed by hardware manufacturers because the retail version is missing drivers, but this need not be the case with Linux. (In fact, if your hardware is supported, installing Linux takes less than an hour.)

Eric Raymond, in his recent essay “World Domination 201”, argues that the free software community should only purchase PCs from vendors that ship Linux. However, it is much simpler just to convince hardware vendors to support Linux than to create new multinational corporations.

Even if vendors don't provide support for Linux yet, they could make sure it runs. Again, while people might debate whether free software is good for software companies, it is inarguably a benefit for hardware companies because it lowers the cost of a computer. In fact, PC hardware vendors don't even need to do any work other than demand Linux drivers from *their* component vendors. The power easily lies within the PC hardware vendors to apply the necessary pressure.

Shipping computers with Linux is not sufficient for world domination because hardware vendors don't have the means to solve some of Linux's problems. Dell cannot ensure that Apple's iTunes works on Linux, or that OpenOffice can read your Publisher files.

## The Desktop

Dell IdeaStorm is a website launched by Dell on February 16, 2007 to allow them to gauge which ideas are most important and most relevant to the public.

In the first 11 days, more than 83,000 users had requested that Linux should be provided on all Dell PCs. In a statement issued on its web site, Dell said it had “taken notice” of the suggestions.

—[Cnet.com article](#)

Free software developers have had a long row to hoe. As proprietary software has been the dominant model for decades, many geeks have had to live with one foot in each world; Linux programmers today are often forced to use Macs, Windows and other proprietary software.

The good news is that a Linux PC does a reasonably good job of interoperating with proprietary hardware and software, especially Microsoft technology. Today, Linux supports the Microsoft Office file format, Windows Media file formats and protocols, MSN Instant Messenger protocols, Microsoft file systems, C#, and much more. The implementations aren't perfect, but just the most popular 95% of a standard is good enough for most. In fact, by tackling the easy and more important portions, the codebases are smaller and simpler than their Microsoft counterparts.

In general, when a free software application supports a file type or protocol, the support is good, but there are still Microsoft formats such as Publisher, Money, and Access whose lack of support is problematic today. Having worked on Microsoft Publisher, I can say it is about 5,000 lines of code for OpenOffice to support reading these documents, which is not a big deal. The work can be done by an enterprising volunteer, or via a bounty.

As mentioned in the Apple section, one of Linux's most surprising capabilities is its ability to run Windows applications. Given a complete operating system, implementing the Win32 syntax for similar functionality is not difficult. Therefore, a backup strategy for supporting proprietary file formats is to run the proprietary Windows programs on Linux.

I was able to switch to using Linux 100% of the time, and cannot foresee a reason to switch back, but others are not so lucky. Work in these areas needs to continue.

## Approachability

The adversary she found herself forced to fight was not worth matching or beating; it was not a superior ability which she would have found honor in challenging; it was ineptitude — a gray spread of cotton that deemed soft and shapeless, that could offer no resistance to anything or anybody, yet managed to be a barrier in her way. She stood, disarmed, before the riddle of what made this possible. She could find no answer.

—Ayn Rand, *Atlas Shrugged*

I believe the free software community ought to make their top focus a fully functioning out of the box (OOB) experience. People need to be able to take a Linux CD, stick it into a new, or old, computer, have it recognize the hardware, install everything, and migrate the data. A key to Linux's success, and to the success of computers, is that it should work with minimal user input.

Installing Windows, as many people know, can take a full day.<sup>8</sup> To beat Windows, Linux will have to do better. Fortunately, even with all of its warts, Linux isn't that far off. I wrote a post on my blog in April, 2006 entitled “10,000 Bugs Away from World Domination.” In it, I wrote that if we could fix about 10,000 selected bugs in the free software stack (50 man-years of work, compared to Vista that had 5,000 man-years of effort), we could come very close to building an

---

<sup>8</sup> Assuming you don't run into a situation where you need a network card driver, but you can't download it until you have a functioning network card!

operating system that someone as computer unsavvy as my mom could install and maintain. Right now, setting up Linux can take ten steps, but any issue you run into is likely to be arcane to fix.

Even if we never get there for everyone, we can definitely get closer than we are now which is important because fixing computer problems is never easy, even on Windows. In fact, a perk of Microsoft VIPs was that they had personal IT staff for computer setup and maintenance! With their established IT infrastructures, large businesses should be able to move to Linux today, even with its warts.

In general, while the Linux kernel has a few remaining challenges, the biggest challenge for the next five years will be for the rest of the free software stack to get up to the same level of quality as the kernel. While Firefox and OpenOffice are rich and quite stable, few would argue that these popular and important applications, or any other free software for that matter, has the quality, reliability and elegance of the Linux kernel.

In some ways, applications are harder to build because unlike a hardware driver, which is by definition concrete and fully specified, a web browser and a word processor are applications whose full requirements are unknown and unlimited. What should a web browser do while disconnected? If speech recognition becomes the de facto input method, how does this change the user interface of a word processor? These are questions to which answers are elusive.

In the tools chapter, I discussed how modern tools will build more reliable applications, but the important point here is that better tools will make developers more productive, which will allow software to evolve more quickly as we better figure out the answers to some very hard questions.

It is important that free software programs, and their codebases, be approachable. Wikipedia's motto: "Don't bite the newcomers" is a great one for every free software organization. A big challenge for the next few years is to make the entire free software stack easier to use and more approachable for new users and programmers.

Many wonder whether Linux needs a killer application to beat Windows. Linux's status today of being mostly compatible with, and slightly more reliable than Windows might *not* be enough to overcome the inertia required for a worldwide switch, even to something that is free to acquire.

It is true that if Linux were to enable robust continuous speech recognition, or some other transformational feature, Linux could more quickly take over, but even that OS would also need to support

all your hardware and file formats. However, Linux already has what I think is a killer feature: the rich set of programs that come with it — although they could still use a bit more work. For example, the most popular high-end graphics editor in Linux, poorly named the “GIMP”, is comparable to Photoshop, suitable for professionals, and free! However, like Photoshop, it isn't particularly approachable for a new person to jump in and start using. The first time you use GIMP, it might take 20 minutes to figure out how to crop an image. It takes time to learn any powerful tool, but a lesson in engineering is: “You only pay for what you use.” Simple things should be simple to do, and today this is not always the case. While many of the free applications are good enough to convert people from Windows, and as good as competing proprietary products, they could still be dramatically better.

While the free software community is already doing good business, especially for websites and embedded applications, PC hardware and software are the biggest and final challenge. The minimal requirements, the Web and Office, are almost met today, yet its true potential, its Wikipedia-scale potential, is something much larger and several years of work away.

Once we start to coalesce into a few very good codebases, then the PhDs will be motivated to jump in. Nowadays, a graphics researcher might use GIMP to manipulate images, but he would definitely not use GIMP's source code as the basis for his research. Eventually computer scientists around the world will instinctively realize the place to contribute is in the free software code, just as there are linguists and computer scientists today using Wikipedia as the basis for their research. Once the PhDs get involved, software will get very interesting.

## Monoculture

Next to the problem of code written in old programming languages, the second biggest challenge the free software community faces is the amount of duplicate code that exists. There are several dictionaries in Linux, and so when you add a custom word to one, only some applications notice it. In Linux there are too many paint programs, media players, download managers, RSS readers, programming environments, source control systems, etc. Each piece of duplicate code is a missed opportunity for sharing and slows



progress. On the other hand, the free software community has gotten together on primarily one kernel, Linux, and one web server, Apache.

There is an interesting debate in the software community about the worry of a software monoculture. Like a monoculture in the biological world, one virus or other problem could destroy the entire ecosystem, and some argue that we run similar risks in the software world.

While a monoculture may be a risk, having different codebases doesn't necessarily help: any new, previously unforeseen exploit is able to cause damage to all codebases because none of them designed for it. A castle's walls may stop men, but they were not designed to stop cannonballs or helicopters.

While the little differences between codebases add an extra level of variability that makes building a virus harder, software is different from DNA (today at least!) because of our ability to infinitely recombine it. It is possible to make a product easy to use, and powerful, rich and reliable, fast and maintainable. Because software is infinitely malleable, all the best ideas can be incorporated into a single product.

In fact, the monoculture risk only applies to the proprietary software world because in that model we are unable to infinitely recombine ideas. There are hundreds of millions of Windows users, but there are only a few thousand people at Microsoft who can make changes to it.

If the monoculture risk doesn't apply to the free software world, we should focus more on working together. Progress in the free software world is slow where people are not working on the same codebases. As described in the tools chapter, switching to modern tools will bring a 2x productivity improvement, and consolidating various efforts into unified codebases will bring an additional 5x productivity jump. Unfortunately, agreeing to unified efforts is a much harder challenge because each group needs to reach out to other groups, and be willing to combine their efforts and potentially abandon their existing software investments.

SourceForge.net has 1.9 million programmers, but when you divide by 50 because they are volunteers, and then divide by 10 because of old tools and duplicate code, our 1.9 million-man army has only 3,800 people. However, it is amazing what free software has built when you realize that today it is on just a (relatively) small number of PCs. Even with that tiny user base, and their inefficien-

cies, they still have had the resources to do everything they have done. When you add to the 10x productivity improvement another 5x increase in users and contributors, free software in five years could have what seems like 50 times as many programmers as they do today.

# Linux Dev Tools

Simplicity is the ultimate sophistication.

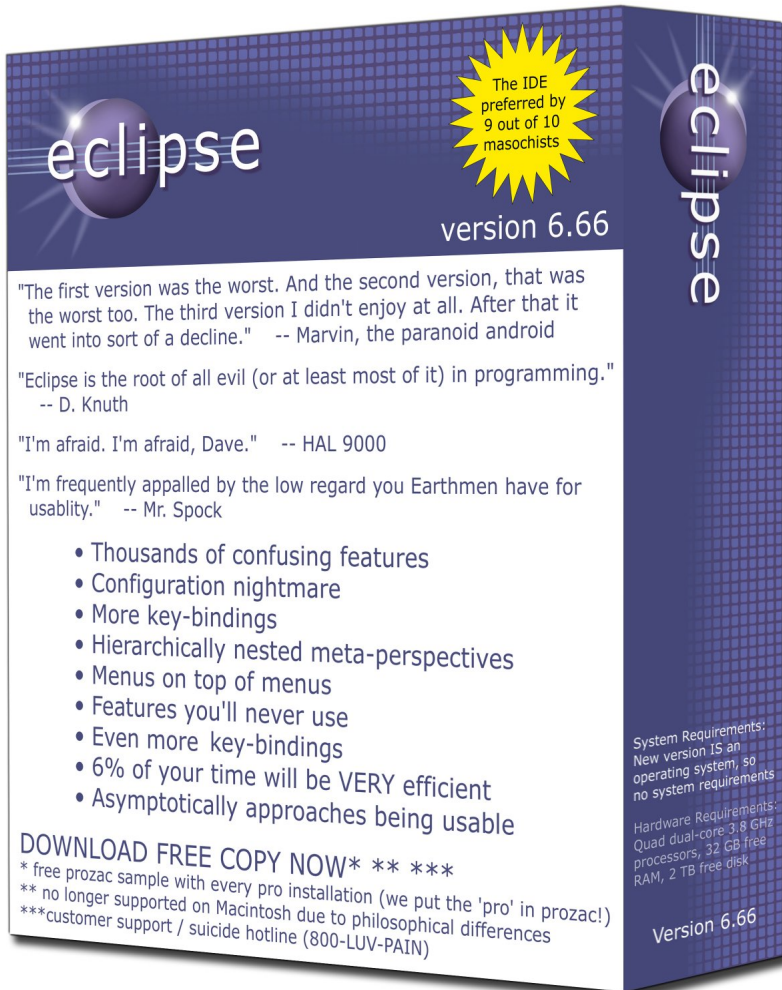
—Leonardo Da Vinci

Developers, developers, developers, developers, developers,  
developers!

Developers! Developers! Developers! Developers! Developers!

Developers! Developers! Developers! Developers! Yes!

—Steve Ballmer



*Parody of Eclipse packaging, the most popular free developer tool, from the funny folks at FarOutShirts.com. I took a few pages of notes of my frustrating experiences but decided that the picture above did a better job.*

Over the years Microsoft has invested a lot in the development of tools, and this generated much of its revenues in the early years. In fact, Microsoft was a tools company before it was an operating systems company.

Next to Windows, the most important product Microsoft produces today is Visual Studio. The tool has flaws, but it wins awards, especially for its ease of use. It is possible to create your first Windows application or web site just minutes after installing it.

The closest thing to Visual Studio in the free software world is Eclipse, which is despised by many for its excess complexity. Craig Mundie, Microsoft's Chief Research and Strategy Officer, [asserted](#) in a recent interview that the Linux server market share had plateaued. If this is the case, the state of Linux's tools has to be a big reason for this.

Eclipse is more powerful than Visual Studio, and can be used to develop all kinds of applications, but it has never had the same approachability. Eclipse has gotten better over the years, but it still has a long way to go. The community needs to focus on building powerful yet easy to use tools.

## Backward Compatibility

As described in the Linux chapter, all of the source for the entire software stack is publicly available so fixes can be made in the proper place. This is an important part of what makes free software simpler, more maintainable and more reliable over time.

The free software stack has traded simplicity for backward compatibility and so there is a downside. For example, Windows lets you download drivers off the web that will work on multiple versions of its OS, but Linux does not support this scenario because the internal kernel architecture is constantly changing. If hardware vendors did make Linux drivers available on the web, they would potentially have different drivers for every version of the kernel. The best way to get a new driver in Linux is to grab the latest kernel, which is where all the new drivers go.

It is possible that if you buy a new piece of hardware, your old kernel might not have support for the new driver. What then?

To be clear, this scenario is contrived in several ways:

- Anyone can add a driver to the Linux kernel, so the need to ever download code from a website is greatly reduced.

- The vast majority of the hardware you would buy in a store will have drivers that have been a part of the Linux kernel for several years. Even new hardware typically uses old drivers.
- Linux ships more frequently, and the code is free, so it is less likely you will be running an old version of the Linux kernel.

This is not a problem that has ever bitten me, but it is theoretically possible given its development model. In this situation, it could be that the simplest way to get your new hardware to work is to install a new kernel. Unfortunately, most Linux distributions do not allow you to just update the kernel as you must upgrade the entire OS. This is a notable downside of today's Linux.<sup>9</sup>

However, this is a trade-off in the development process that favors simplicity, reflects the dynamic nature of software, and so the need to upgrade software periodically is a fact of life that needs to be accepted rather than resisted. The alternative is a backward compatibility quagmire that Microsoft wallows in.

There are other backward compatibility issues. OpenOffice's native format is OpenDocument, but you can't use it until all whom you might share documents with are also using that software. And even when the documents are converted to the new format, the files are often hardcoded with font names owned by Microsoft. Proprietary software's biggest "advantage" is its switching costs. Oracle's proprietary database is powerful, yet unwieldy and old, and application vendors who have spent thousands of hours writing Oracle-specific code, have also created a situation where moving to any other product, even something free, is expensive. Fortunately, these are one-time costs.

---

9 One solution is to build an OS that continuously and smoothly upgrades itself to the latest software, a feature that is not offered on many distributions today. I think it would be great to always have the latest and greatest software on my computer without having to lift a finger. This is primarily a testing challenge.

# STANDARDS & WEB

**From:** Bill Gates  
**Sent:** Saturday, December 5, 1998  
**To:** Bob Muglia, Jon DeVann, Steven Sinofsky  
**Subject:** Office rendering

One thing we have got to change in our strategy - allowing Office documents to be rendered very well by other people's browsers is one of the most destructive things we could do to the company.

We have to stop putting any effort into this and make sure that Office documents very well depends on PROPRIETARY IE capabilities.

Anything else is suicide for our platform. This is a case where Office has to avoid doing something to destroy Windows.

I would be glad to explain at a greater length.

Likewise this love of the standard DAV in Office/Exchange is a huge problem. I would also like to make sure people understand this as well.

**T**o use the Internet, you need software that supports two big standards: TCP/HTTP and HTML. There is no "HTML" standard competing with an "HTMM" standard, as the idea is silly on its face, yet such redundancies exist in many other areas in the world of bits today. When you can't agree on a file format, your ability to exchange information goes from 1 to 0.

Free software has been a part of the Internet since the beginning. In fact, a website needs to send you software, in the form of HTML and JavaScript, in order for you to have something to look at and interact with. It is easy to learn how a website does its magic as the tags of an HTML document are self-describing and, on top of that, there is an organization called W3C whose job is to fully describe them.

In contrast, to display a Word or WordPerfect document, you had to reverse-engineer a complicated binary file format! This prevented their widespread use as the standard document format of the Internet.

## Digital Images

Unlike audio and video, in the realm of still images things are in good shape. JPEG is an efficient, free, and widely-supported standard for compression of images.<sup>1</sup> There might be a couple of standards better than JPEG out there, but the end is nigh. (There is a JPEG 2000 standard based on wavelets<sup>2</sup> which is 20% better than JPEG, but it has higher memory and processing demands. Sometimes to get a little more compression, you have to do a lot more work, and so you reach a point of diminishing returns.)

Microsoft in early 2007 announced a new Windows Media Photo format, which is also 20% better than JPEG, but without higher memory and processing demands like those required by JPEG 2000. The new format is based on JPEG, but with nine small tweaks. The spec is public, and there is even free public source code, but the license explicitly excludes it from being used in combination with copyleft licenses:

2. c. Distribution Restrictions. You may not ... modify or distribute the source code of any Distributable Code so that any part of it becomes subject to an Excluded License. An Excluded License is one that requires, as a condition of use, modification or distribution, that the code be disclosed or distributed in source code form; or others have the right to modify it.

Fortunately, reverse engineering is easily doable because the spec is public and if you start with existing JPEG code you are already very close — a tiny, hidden, speed bump Microsoft has created for the free software community.

## Digital Audio

It is the mess of proprietary standards and patent restrictions that are impeding the progress to digital audio. Proprietary software companies have been pushing their standards down our throat. If

- 
- 1 There is also PNG and GIF for lossless compression, but they are not suitable for real-world images with continuous gradients like clouds, etc. I took a high quality 1.9 MB JPEG and converted it to PNG and it became 2.9 times bigger. Interestingly, these lossless formats do a better job than JPEG for certain images like screen shots because JPEG doesn't handle the sharp transitions from black to white, etc. that you find on a computer screen. A JPEG of a screen shot is 2.2 times larger than an equivalent PNG. With a JPEG of the same size, you find it has added gray display artifacts at the black/white transitions. PNG was only created because after GIF became popular, Compuserve started suing.
  - 2 Wikipedia: "A wavelet is a kind of mathematical function used to divide a given function or continuous-time signal into different frequency components and study each component with a resolution that matches its scale."

you insert a music CD into a Windows computer, it wants to rip the audio into WMA, a proprietary format which the Mac OS doesn't support out of the box. If you insert that CD into a Mac, it rips the music into AAC, a format that Windows doesn't support by default.

I'm not sure what should be done about this colossal mess. We should keep trying to pick a standard format for audio, like JPEG has become for still images. The best candidates are OGG and MP3. MP3 is an old format and while it is not considered state of the art, it is efficient enough. MP3's primary problem is that there are a number of companies with patent claims against it. If the industry cannot agree to end the patent hassles, it should adopt OGG or some other fully free format and tell those MP3 license owners to take a long walk off a short pier.

If we could finally agree on a free digital music format, we could finally have digital music — which will also open up many possibilities of a richer connection between artist and consumer. Whatever format is chosen, we need to define the number of bits per second required to achieve transparency.<sup>3</sup> We also need to come up with a standard streaming protocol and video format, but I won't even get into that here; we must crawl before we can walk.

## The Next-Gen DVD Mess

Like many other things in the computer industry, the next generation format for high definition DVDs was a mess for several years because two standards were created: HD-DVD and Blu-Ray. Both formats are high quality and their discs look the same, but the playback hardware is inexplicably incompatible.

Fortunately, in early 2008, it looks like HD-DVD is going to be abandoned, although billions of dollars were wasted because the two camps bickered like twin sisters, and couldn't agree on a few minuscule technical details *before* they went to market. The bad news is that Blu-Ray adoption is slow. Why pay \$300 for a new player when all of the movies we already own are not in that format? Why should we pay full price merely to get a higher quality copy of something we already own? Didn't I really mean to purchase a high quality copy in the first place? The best way for faster adoption

---

<sup>3</sup> Transparency is defined as audio that is of such high quality it cannot be distinguished from a CD. Transparency *should* mean passing the aforementioned listening test, plus when you convert it to other transparent codecs and back, even 100 times, the quality is not diminished. To achieve that we do not need to go to lossless compression, which is five to seven times bigger.



would be to create a service where you mail in your old VHS or DVDs, and they mailed you back HD versions of them for a few dollars each. That could be a huge, if low-margin, business.

Consumers would adopt new standards faster if they could immediately enjoy everything they already own in that new format at something approaching the actual cost of producing a disc, instead of the full retail price. This is another area where shrinking the copyright expiration will help. It will make nearly free all those things we paid a license fee for, but which we now only have low-quality reproductions of.

At some point, the industry should remove the Interpol and DHS warnings about why we shouldn't have stolen what we are about to watch. When you stick in a disc, there should be two buttons that show up within 5 seconds: "Play" and "Menu." I have a boxed set of DVDs that display four minutes of introductory warnings and self-promotion that I cannot even skip through, before playing the actual content. I once put in the wrong disc and so had to repeat the hassle. This may seem petty, but these encroachments get worse. I have to accept the license agreement of my navigation system every time I turn on my car!<sup>4</sup> When people feel ripped off and treated like a sucker, a culture is created where people decide not to pay for what is sold. Respect is a two-way street.

---

4 Suppose I don't activate my nav system one day because I can't be bothered. Then I get into an accident and die because the system wasn't helping me. It could therefore be argued that the need to accept the license agreement played a factor in the cause of my death. The good news is that it would create standing to sue! Now, if only we could come up with a way for someone to die because they had to sit through all that stuff at the beginning of a video so we could get standing for that!

# MS's Support of Standards

| Ancient Egyptian 3000 BC |                          |        |        | proto-Sinaitic 2000 BC |             |        |            | Phoenician 1100 BC |       |        |        | Hebrew      |       |        |        | Greek 800-600 BC |        |          |        | Etruscan |        | Latin |  |
|--------------------------|--------------------------|--------|--------|------------------------|-------------|--------|------------|--------------------|-------|--------|--------|-------------|-------|--------|--------|------------------|--------|----------|--------|----------|--------|-------|--|
| word                     | means                    | symbol | symbol | word                   | means       | symbol | name       | word               | means | symbol | name   | word        | means | symbol | name   | early classic    | modern | name     | symbol | early    | modern |       |  |
| k3                       | ox                       |        | →      | /ʔ/ 'alep              | ox          | →      | /ʔ/ 'aleph | ox                 | →     | א      | aleph  | ox          | →     | א      | aleph  | →                | α      | alpha    | →      | →        | A      | a     |  |
| pr                       | /pr/ house               |        | →      | /b/ bayit              | house       | →      | /b/ beth   | house              | →     | ב      | beth   | house       | →     | ב      | beth   | →                | β      | beta     | →      | →        | B      | b     |  |
| m'3t                     | throw stick              |        | →      | /g/ gimel              | throw stick | →      | /g/ gimel  | throw stick        | →     | ג      | gimel  | throw stick | →     | ג      | gimel  | →                | γ      | gamma    | →      | →        | G      | g     |  |
| '3                       | door                     |        | →      | /d/ dalet              | door        | →      | /d/ dalet  | door               | →     | ד      | dalet  | door        | →     | ד      | dalet  | →                | Δ      | delta    | →      | →        | D      | d     |  |
|                          | (nervous)                |        | →      | /h/ he                 | he          | →      | /h/ he     | he                 | →     | ה      | he     | wall        | →     | ה      | he     | →                | Ε      | epsilon  | →      | →        | E      | e     |  |
|                          |                          |        | →      | /w/ wawwu              |             | →      | /w/ waw    |                    | →     | ו      | waw    |             | →     | ו      | waw    | →                | Ϝ      | digamma  | →      | →        | F      | f     |  |
| hct                      | /h/ wick                 |        | →      | /z/ zain               |             | →      | /z/ zayin  | sword?             | →     | ז      | zayin  | sword?      | →     | ז      | zayin  | →                | Ζ      | zeta     | →      | →        | Z      | z     |  |
|                          |                          |        | →      | /h/ heth               |             | →      | /ch/ heth  |                    | →     | ח      | heth   |             | →     | ח      | heth   | →                | Η      | eta      | →      | →        | H      | h     |  |
| ni                       | /ni/ arm, (to push away) |        | →      | /y/ yadu               | fist?       | →      | /y/ yodh   | hand               | →     | י      | yodh   | hand        | →     | י      | yodh   | →                | ι      | iota     | →      | →        | I      | i     |  |
|                          |                          |        | →      | /k/ kappu              |             | →      | /k/ kaph   | hand?              | →     | כ      | kaph   | hand?       | →     | כ      | kaph   | →                | Κ      | kappa    | →      | →        | K      | k     |  |
| net                      | /w/ water                |        | →      | /l/ lamdu              | ox goad     | →      | /l/ lamadh |                    | →     | ל      | lamadh |             | →     | ל      | lamadh | →                | λ      | lamda    | →      | →        | L      | l     |  |
| fy                       | /n/ water                |        | →      | /m/ mayim              | water       | →      | /m/ mem    | water              | →     | מ      | mem    | water       | →     | מ      | mem    | →                | μ      | mu       | →      | →        | M      | m     |  |
| net                      | /f/ horned               |        | →      | /n/ nahas              | snake       | →      | /n/ nun    |                    | →     | נ      | nun    |             | →     | נ      | nun    | →                | ν      | nu       | →      | →        | N      | n     |  |
| irt                      | viper                    |        | →      | /ʔ/ enu                | eye         | →      | /s/ samekh | fish?              | →     | ס      | samekh | fish?       | →     | ס      | samekh | →                | Ξ      | ksi      | →      | →        | S      | s     |  |
| db'                      | /f/ finger               |        | →      | /ʔ/ ayin               | eye         | →      | /ʔ/ ayin   | eye                | →     | ע      | ayin   | eye         | →     | ע      | ayin   | →                | Ο      | o mikron | →      | →        | O      | o     |  |
| ibh                      | /bh/ tooth               |        | →      | /p/ pe                 | mouth?      | →      | /p/ pe     | mouth?             | →     | פ      | pe     | mouth?      | →     | פ      | pe     | →                | Π      | pi       | →      | →        | P      | p     |  |
|                          |                          |        | →      | /s/ san                |             | →      | /ts/ sade  |                    | →     | צ      | sade   |             | →     | צ      | sade   | →                | Σ      | san      | →      | →        | S      | s     |  |
|                          | (elephant's tusk)        |        | →      | /q/ qoppa              |             | →      | /q/ qoph   | monkey             | →     | ק      | qoph   | monkey      | →     | ק      | qoph   | →                | Ϙ      | qoppa    | →      | →        | Q      | q     |  |
|                          |                          |        | →      | /r/ rashu              | head        | →      | /r/ resh   | head               | →     | ר      | resh   | head        | →     | ר      | resh   | →                | ρ      | rho      | →      | →        | R      | r     |  |
|                          |                          |        | →      | /sh/ shin              |             | →      | /sh/ shin  |                    | →     | ש      | shin   |             | →     | ש      | shin   | →                | Σ      | sigma    | →      | →        | S      | s     |  |
| nfr                      | /nfr/ perfect            |        | →      | /t/ tawwu              | cross       | →      | /t/ taw    | mark?              | →     | ת      | taw    | mark?       | →     | ת      | taw    | →                | Τ      | tau      | →      | →        | T      | t     |  |
|                          |                          |        | →      | /u/ u                  |             | →      | /u/ u      |                    | →     | י      | u      |             | →     | י      | u      | →                | Υ      | u        | →      | →        | U      | u     |  |
|                          |                          |        | →      | /v/ v                  |             | →      | /v/ v      |                    | →     | ו      | v      |             | →     | ו      | v      | →                | Ϝ      | v        | →      | →        | V      | v     |  |
|                          |                          |        | →      | /w/ w                  |             | →      | /w/ w      |                    | →     | ו      | w      |             | →     | ו      | w      | →                | Ϝ      | w        | →      | →        | W      | w     |  |
|                          |                          |        | →      | /x/ x                  |             | →      | /x/ x      |                    | →     | ח      | x      |             | →     | ח      | x      | →                | Χ      | chi      | →      | →        | X      | x     |  |
|                          |                          |        | →      | /y/ y                  |             | →      | /y/ y      |                    | →     | י      | y      |             | →     | י      | y      | →                | Υ      | y        | →      | →        | Y      | y     |  |
|                          |                          |        | →      | /z/ z                  |             | →      | /z/ z      |                    | →     | ז      | z      |             | →     | ז      | z      | →                | Ζ      | z        | →      | →        | Z      | z     |  |
|                          |                          |        | →      | /c/ c                  |             | →      | /c/ c      |                    | →     | צ      | c      |             | →     | צ      | c      | →                | Ϙ      | c        | →      | →        | C      | c     |  |
|                          |                          |        | →      | /g/ g                  |             | →      | /g/ g      |                    | →     | ג      | g      |             | →     | ג      | g      | →                | Γ      | g        | →      | →        | G      | g     |  |
|                          |                          |        | →      | /d/ d                  |             | →      | /d/ d      |                    | →     | ד      | d      |             | →     | ד      | d      | →                | Δ      | d        | →      | →        | D      | d     |  |
|                          |                          |        | →      | /h/ h                  |             | →      | /h/ h      |                    | →     | ה      | h      |             | →     | ה      | h      | →                | Η      | h        | →      | →        | H      | h     |  |
|                          |                          |        | →      | /y/ y                  |             | →      | /y/ y      |                    | →     | י      | y      |             | →     | י      | y      | →                | Ι      | y        | →      | →        | I      | i     |  |
|                          |                          |        | →      | /k/ k                  |             | →      | /k/ k      |                    | →     | כ      | k      |             | →     | כ      | k      | →                | Κ      | k        | →      | →        | K      | k     |  |
|                          |                          |        | →      | /l/ l                  |             | →      | /l/ l      |                    | →     | ל      | l      |             | →     | ל      | l      | →                | λ      | l        | →      | →        | L      | l     |  |
|                          |                          |        | →      | /m/ m                  |             | →      | /m/ m      |                    | →     | מ      | m      |             | →     | מ      | m      | →                | μ      | m        | →      | →        | M      | m     |  |
|                          |                          |        | →      | /n/ n                  |             | →      | /n/ n      |                    | →     | נ      | n      |             | →     | נ      | n      | →                | ν      | n        | →      | →        | N      | n     |  |
|                          |                          |        | →      | /s/ s                  |             | →      | /s/ s      |                    | →     | ס      | s      |             | →     | ס      | s      | →                | Ξ      | ksi      | →      | →        | S      | s     |  |
|                          |                          |        | →      | /o/ o                  |             | →      | /o/ o      |                    | →     | ע      | o      |             | →     | ע      | o      | →                | Ο      | o        | →      | →        | O      | o     |  |
|                          |                          |        | →      | /p/ p                  |             | →      | /p/ p      |                    | →     | פ      | p      |             | →     | פ      | p      | →                | Π      | pi       | →      | →        | P      | p     |  |
|                          |                          |        | →      | /q/ q                  |             | →      | /q/ q      |                    | →     | ק      | q      |             | →     | ק      | q      | →                | Ϙ      | qoppa    | →      | →        | Q      | q     |  |
|                          |                          |        | →      | /r/ r                  |             | →      | /r/ r      |                    | →     | ר      | r      |             | →     | ר      | r      | →                | ρ      | rho      | →      | →        | R      | r     |  |
|                          |                          |        | →      | /sh/ sh                |             | →      | /sh/ sh    |                    | →     | ש      | sh     |             | →     | ש      | sh     | →                | Σ      | sigma    | →      | →        | S      | s     |  |
|                          |                          |        | →      | /t/ t                  |             | →      | /t/ t      |                    | →     | ת      | t      |             | →     | ת      | t      | →                | Τ      | tau      | →      | →        | T      | t     |  |
|                          |                          |        | →      | /u/ u                  |             | →      | /u/ u      |                    | →     | י      | u      |             | →     | י      | u      | →                | Υ      | u        | →      | →        | U      | u     |  |
|                          |                          |        | →      | /v/ v                  |             | →      | /v/ v      |                    | →     | ו      | v      |             | →     | ו      | v      | →                | Ϝ      | v        | →      | →        | V      | v     |  |
|                          |                          |        | →      | /w/ w                  |             | →      | /w/ w      |                    | →     | ו      | w      |             | →     | ו      | w      | →                | Ϝ      | w        | →      | →        | W      | w     |  |
|                          |                          |        | →      | /x/ x                  |             | →      | /x/ x      |                    | →     | ח      | x      |             | →     | ח      | x      | →                | Χ      | chi      | →      | →        | X      | x     |  |
|                          |                          |        | →      | /y/ y                  |             | →      | /y/ y      |                    | →     | י      | y      |             | →     | י      | y      | →                | Υ      | y        | →      | →        | Y      | y     |  |
|                          |                          |        | →      | /z/ z                  |             | →      | /z/ z      |                    | →     | ז      | z      |             | →     | ז      | z      | →                | Ζ      | z        | →      | →        | Z      | z     |  |
|                          |                          |        | →      | /c/ c                  |             | →      | /c/ c      |                    | →     | צ      | c      |             | →     | צ      | c      | →                | Ϙ      | c        | →      | →        | C      | c     |  |
|                          |                          |        | →      | /g/ g                  |             | →      | /g/ g      |                    | →     | ג      | g      |             | →     | ג      | g      | →                | Γ      | g        | →      | →        | G      | g     |  |
|                          |                          |        | →      | /d/ d                  |             | →      | /d/ d      |                    | →     | ד      | d      |             | →     | ד      | d      | →                | Δ      | d        | →      | →        | D      | d     |  |
|                          |                          |        | →      | /h/ h                  |             | →      | /h/ h      |                    | →     | ה      | h      |             | →     | ה      | h      | →                | Η      | h        | →      | →        | H      | h     |  |
|                          |                          |        | →      | /y/ y                  |             | →      | /y/ y      |                    | →     | י      | y      |             | →     | י      | y      | →                | Ι      | y        | →      | →        | I      | i     |  |
|                          |                          |        | →      | /k/ k                  |             | →      | /k/ k      |                    | →     | כ      | k      |             | →     | כ      | k      | →                | Κ      | k        | →      | →        | K      | k     |  |
|                          |                          |        | →      | /l/ l                  |             | →      | /l/ l      |                    | →     | ל      | l      |             | →     | ל      | l      | →                | λ      | l        | →      | →        | L      | l     |  |
|                          |                          |        | →      | /m/ m                  |             |        |            |                    |       |        |        |             |       |        |        |                  |        |          |        |          |        |       |  |

working on Mac and Unix? Proprietary protocols are a special type of bad standard because they tie you to two pieces of someone's technology. Microsoft's Outlook uses a proprietary protocol to fetch e-mail from Exchange; this protocol ties you to client and server.

The free software guys do not, and should not, have any compunction about implementing closed, undocumented or poorly documented standards, as long as the standard is popular. An open standard is always better, but it is the source code that is the true value of intellectual property, not whether the details are sanctioned by a particular organization.

Free software supports more standards than proprietary software. If a standard is popular, it will get implemented. Linux by default rips music into the free OGG, but it is easy to change it. The Linux media player supports Microsoft's and Apple's media formats, plus free standards. Linux's instant messaging applet supports MSN, Yahoo, AIM, ICQ, GroupWise, Jabber, and others. In fact, proprietary software is holding up the development and adoption of new standards. The BBC has created a free wavelet-based video codec known as Dirac, but only Linux supports it out of the box today.

Since it is built by users who add features as necessary, free software gives you more choices than any proprietary vendor can motivate themselves to provide. Not only is there better support for standards, but also you will never be locked into a corner because support for a standard disappeared for "strategic" reasons. With open formats and free software, you are building a future-proof platform.

The web is the most important standard for information exchange, but one that is almost as important is the standard for office documents.

# OpenDocument Format (ODF)

**From:** Bill Gates  
**Sent:** Thursday, August 5, 1999  
**To:** Bob Muglia

Why would the Office Group be giving out the Office 2000 format to competitors? To me this sounds crazy.



*Microsoft Office 2007 promotional screenshot (Got Blue?) Office includes Word, Excel, PowerPoint, Outlook, Access, FrontPage, Visio, Project, OneNote, web services, and tools to build Office extensions, an incredible package of interconnected proprietary technologies.*

Microsoft's relationship with document creators is older and more entrenched than its relation with Windows users; I've been using Windows since 1990, but have been using Word and Excel since 1986. For hundreds of millions of information workers, students, writers, and government employees throughout the world, the primary tools for producing their intellectual property is inside Microsoft Office, and there are billions of documents out there. I have heard that every corporate purchase in the UK involves the creation of an Excel spreadsheet at a stage of the process. I have also learned that creating PowerPoint documents is a required skill for the upper echelons of the US military.

Microsoft Office is key to Microsoft's profits, and the best reason to install Windows. Today, about 50 million people use the free OpenOffice, while the rest of the world has paid \$200 or \$0 for a copy of Office. (The name "OpenOffice" is trademarked by someone else, so their "official name" is actually "OpenOffice.org".)

When you realize how much intellectual effort is expended inside productivity tools, you realize how important it is that the file formats be documented. A company or a government would like to know that they can keep these files around for decades like they can with paper. This is, of course, a much harder challenge because on a piece of paper all the digitized words, text layout rules, and other formatting information are lost — imagine if someone replaced all your Word documents with screenshots of those documents; the information would be readable, but not editable.

The Microsoft Word team didn't try to build a file format they would be happy with for 20 years because they knew that engineering Word in 1993 to read the files of Word 2013 was an impossible task. For the first 10 years, Word  $n$  could never even read the files of Word  $n + 1$ . This is because for many years Office used binary formats, and it would have crashed if it tried. Inside a binary Word file, you might find the following data:

|             |           |
|-------------|-----------|
| 05          | 01        |
| <b>bold</b> | <b>on</b> |

*'0501' means turning on bold*

|             |            |
|-------------|------------|
| 05          | 00         |
| <b>bold</b> | <b>off</b> |

*'0500' means turning off bold*

|    |    |
|----|----|
| 06 | ?? |
| ?! | ?! |

*What if an old Word doesn't understand '06' ? What next?*

If you don't know what '06' means, you don't know what is coming next, and so you can't continue and must abort. Even if you wanted to skip over it, you can't because you don't know how far to advance.

Microsoft did not create binary formats to lock out other vendors. Formats by all the word processors were binary for many years because it is efficient and because a better solution hadn't been invented.

The answer to this conundrum, which has plagued computing since the beginning, is for documents to be self-describing, and that is what eXtensible Markup Language (XML) is all about. XML defines how to create a document which any software can read and write without crashing, even if it doesn't completely understand it.

I can guarantee that the following XML (fragment, trimmed for clarity) will be readable by a word processor in 20 years:<sup>5</sup>

```
<?xml version="1.0" encoding="UTF-8"?>
<office:document-content office:version="1.0">
  <office:body>
    <office:text>
      <text:p text:style-name="Standard">Hello, 2028!</text:p>
    </office:text>
  </office:body>
</office:document-content>
```

*"Hello, 2028!" in Open Document Format (ODF)*

XML is built upon Unicode, the standardization of all the world's characters, and adds to it <brackets> which allow you to find the end of every element, even those you don't understand.</brackets>

As tiny as it sounds, the key to being able to read old files years later, or new files with old code, is simply to make the data self-describing, allowing you to figure out a piece of data's name and length. The brackets, and the exact XML format, isn't important, what is important is the fact that all types of information you'd like to represent are doable in a way that is possible for humans to read and for computers to manipulate. When every computer uses a standardized, self-describing format, we will have taken a big first step in being able to exchange documents without causing a crash.<sup>6</sup>

XML is about having a standard, self-describing file format and is one of the most important standards in the history of computing, and *just one* of its uses will be the standard schema to represent the billions of office documents.

- 
- 5 Even if the schema does change in an incompatible way, it should be possible to write a tiny program to update to the new schema – try doing that with a binary format. XML also allows you to keep but ignore information you don't understand, whereas code that supports a binary format usually throws away things it doesn't understand.
  - 6 There could be other formats than XML which is not particularly efficient for computers to parse. Efficiency was never a part of the design, as the code was created after the spec. The XML people say that they can come up with more efficient binary formats and I suggest we take them up on their boast. Perhaps proprietary software is holding this up because a binary XML would require changes to many XML codebases.

The Office binary formats were not documented for many years, and the license agreement for the documentation today says that you can only use the information for products that “complement Microsoft Office.” Is *supplant* the same thing as *complement*?!

Microsoft is not particularly interested in building an open standard because it will never perfectly represent their features, and because an open standard makes it easy to switch tools. Right now everyone buys Office because that is what you need to read the documents you receive today. The adoption of an open format for productivity tools is a mortal threat to Microsoft's Office profit margin.

Regrettably, there is a battle going on in the XML office document space. Microsoft has for many years ignored and then resisted the ISO standard called OpenDocument Format (ODF), and now they have created their own competing standard called Office OpenXML (OOXML). However, the whole point of a standard is to not have two of them.

XML provides the structure for your files and guarantees that applications should be able to parse everything, even parts they don't understand, without crashing. Given that baseline, it should be possible to create a format that can represent the features of office productivity tools. Microsoft's OOXML specification, which provides 100% compatibility with Microsoft Office, is 6,000 pages, while the ODF specification is only 1,000 pages because it doesn't re-use many existing standards like SVG, SMIL, MathML and XForms.<sup>7</sup>

OpenXML is also filled with legacy bloat. At the top of 600 pages of the VML specification is the following text:

Note: The VML format is a legacy format originally introduced with Office 2000 and is included and fully defined in this Standard for backwards compatibility reasons. The DrawingML format is a newer and richer format created with the goal of eventually replacing any uses of VML in the Office Open XML formats. VML should be considered a deprecated format included in Office Open XML for legacy reasons only and new applications that need a file format for drawings are strongly encouraged to use preferentially DrawingML.

---

<sup>7</sup> Stéphane Rodriguez documents a number of defects in the OpenXML standard: <http://ooxmlisdefectivebydesign.blogspot.com/>.

Microsoft has since moved to deprecate this spec in favor of the DrawingML format which is equally as proprietary, but Word 2007 generates VML, so deprecating the spec doesn't make the work go away for apps that want to interop with Microsoft's. [Google wrote](#) in their analysis of OpenXML:

Although OOXML may formally comply with Ecma, it was clearly not designed with an “open” spirit. Comparing the current with the future situation, interoperability is likely to become more difficult instead of easier. The implementation of a fully compatible ODF importer (the current efforts regarding .doc and .xls) is not an easy task, but it is dwarfed by the implementation of a fully compatible OOXML importer, which we estimate to take something between 50 – 500 person years, or even longer. Therefore, although it is theoretically possible to generate an OOXML document, this document will probably only use a very small subset of the standard.

In sum, OOXML can be compared to Microsoft giving access to a labyrinth to which it alone owns a map; moreover, certain tunnels within this labyrinth are not accessible without a key that only Microsoft has, and that third parties would need to replicate first. (And, in doing so, these third parties would not know whether they would violate any rights that exposes them to litigation).

All things being equal, leveraging existing standards is better than re-inventing them, and in choosing between two standards, the one which is smaller, because it re-uses other standards, is going to be a much better choice for the industry. A standard which is hard to support will be adopted slowly, or have buggy implementations. Today, OpenXML is mostly supported by Microsoft, unlike ODF, which has broad industry support from companies like Red Hat, Adobe, Computer Associates, Corel, Nokia, Intel, Oracle, Novell, Google, IBM and Sun.

OpenXML is a proprietary Microsoft format wrapped in XML. This is not a standard suitable for use by many different types of tools over many years. If you start on Day 1 with a lot of baggage, you are doomed. Microsoft has aggressively lobbied for OpenXML support, recognizing that adoption of ODF could make it easy to switch away from Office.

It is important to recognize that Microsoft Office is the most full-featured and popular productivity tools suite on the planet, and therefore any open format should support important Office features. My impression from reading through the the specification is that the OpenDocument guys have bent over backwards to ensure good Microsoft Office compatibility, and they should be commended for



their open-mindedness. For example, I was amazed, and slightly dismayed, to find references inside the spec to DDE, an obscure and now mostly dead Microsoft-only technology. However, this technology became a part of the Microsoft OLE “monikor” format, which specified how documents would embed portions of spreadsheets, and became an important part of Microsoft's documents, thus ODF supports it.

A robust, standard, self-describing file format for productivity tools will allow people to archive their documents, confident that the format will be readable many years into the future.<sup>8</sup> In addition, like everything we build in computers, standards can become platforms for other standards. When ODF incorporates scenarios to encrypt and digitally sign documents, to notarize and transmit legal documents, and support cross-company workflow, e-forms and e-government, it could become a lingua franca in a way that Microsoft Word's DOC and PDF *combined* have never been.

“Competing standards” is a misnomer in my opinion, so perhaps it would be best if everyone were to adopt ODF. Sun is building extensions to Office to support this format, although if people start using the free OpenOffice, which uses this format as its native format, there is no real need for Office. This book was written using OpenOffice, and while the application is far from perfect, it is far beyond good enough for most users.

The state of Massachusetts was forward-looking in *nearly* adopting ODF, but after tons of lobbying by Microsoft, it reversed course and now endorses either OpenXML or ODF. They either caved on the idea of creating a standard, or they didn't really understand the issue. Microsoft has attempted to confuse many on the importance

---

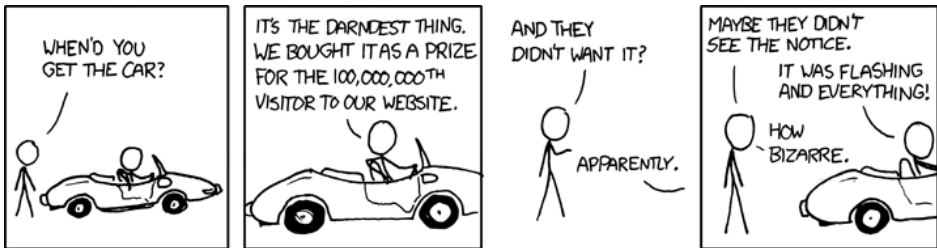
8 The tricky thing about building a standard is that new requirements can cause ripples throughout the design of a system. Imagine two people sitting in different countries collaborating on one file. The challenge is that the ODF file format was not made to be incremental, it was meant to represent an entire document. Do you send a new copy of the document over every time the user makes a change? This is very inefficient and yet doesn't tell the user what changed. Another solution is to use the undo stack, but it doesn't look like OpenDocument stores an undo stack with the document. They could just send around XML diffs, but the XML is not usually the in-memory representation of a document, in which case XML isn't easily usable! One solution is to have an object model (with functions like CreateTable) on top of the file format, and can be sent between computers, but the OpenDocument committee has not attacked this yet. I look forward to seeing how they solve this problem, or whether they decide that while it is a doable feature, it is too hard and outside the scope of the OpenDocument standard. Software is infinitely malleable, but that doesn't mean you'll like what requirements have forced upon you!

of standards as if saying: “There are many standards out there, and code should be able to work with all of them.” It is true that there are lots of standards, but each one should serve a different purpose.

## Web

The first message ever to be sent over the ARPANET occurred at 10:30 PM on October 29, 1969. The message itself was simply the word “login”. The “l” and the “o” transmitted without problem but then the system crashed. Hence, the first message on the ARPANET was “lo”. They were able to do the full login about one hour later.

—[Wikipedia article on Internet precursor ARPANET](#)



Cartoon from [xkcd.com](#)

One could write an entire book about the web, but I did want to include some ideas here.

From a technical perspective, HTML has always been lacking as a text-formatting standard. It is important because of two reasons:

1. It is widely used: it's famous because it's famous.
2. It has an easy deployment model.

HTML was not designed in a rigorous way by text processing experts and so it never had the sort of awe-inspiring respect that Donald Knuth's TeX typesetting system has had. HTML was mature before it even added support for the concept of a page, which is why printing does not work well yet. Text processing is a hard problem, but not incorporating basic features like styles for seven years demonstrates that those guys were in over their heads and they shouldn't have reinvented the wheel.

In addition, the uptake of the JavaScript language outside of the web has been minimal. There is nothing web-specific about the language, and so it didn't even need to be created. It has just made the programming language tower of Babel situation worse.

In spite of its limitations, HTML today is the best cross-platform widget set and is supported by a vast array of tools, so enterprises should be using it for as many corporate applications as possible. A

Boeing airplane could even have its entire cockpit UI be a web site. (You can build something simple, reliable and pretty enough with a web page if you pick the right subset of HTML. Right now an airplane has a mess of buttons and knobs because each subsystem of the airplane has its own set; the guys who build the flaps don't want to share any buttons with those who control the overhead lights.)

The web today is still far from being an appropriate tool for building rich applications, and nearly everything about the web is harder than building an equivalent functionality in a rich client application. Even Google has created a number of applications that presumably couldn't have been built using the web, such as Google Earth, Picasa, and Google Desktop. Google Docs is an impressive web-based engineering effort with nice collaboration features, but it is slow, clumsy, feature-limited, doesn't work offline, has its own authentication mechanism, is hamstrung by the web's limited printing capabilities, and poses no threat to Microsoft's Office business any time soon.

The missing features of HTML are reasons why Adobe's Flash has become so popular. Flash started off as a failed client-based programming environment, but had a rebirth as a web plugin as a way to fix limitations in HTML.

## Adobe Flash

Flash is a primitive GC runtime whose primary advantage is that it is cross-platform like the web. Its programming language is known as ActionScript, which is based on the standard EcmaScript, and which is similar but incompatible with Javascript and JScript. (It is a mess.)

Flash is interpreted, somewhat buggy and not a free standard, nor is it available as free code. We should minimize the use of Flash because it is a big black box to the web server, web browser, search engines, and all existing HTML tools for building and managing websites. Flash also doesn't enforce UI standards. Every flash site works and looks different and sometimes I can't even tell what is a clickable button! Creators of the website, who myopically live in their application, don't notice it, but users visiting the website for the first time do.

Many websites built by non-technical people hire programmers to write a Flash website because of the UI candy it allows, but then they don't update their site for years because it would require hiring a programmer again. It is possible to build pretty, interactive web

pages using only HTML and Javascript, as modern mapping websites have demonstrated. If you want a pretty website, use pretty pictures!

Limiting Flash to specific portions of specific pages, as YouTube does with its video player, is quite reasonable given certain limitations of the web and the mess of video standards, but building whole websites in Flash is a mistake and a threat to the web.

## Merging the Rich Client and the Web

In addition to its limitations, HTML has been stagnant since the release of 4.01 in 1999. We need to continue to evolve HTML, and push tools to keep up with the latest standard. Free software lowers the friction of code distribution and makes this easier to do. I worry that Internet Explorer will hold up progress on HTML because it is so popular, yet it is years behind in certain standards already, and Microsoft has disbanded the team several times.

XHTML and HTML 5 are nascent efforts to improve the web, although they are still fundamentally limited because as standards they define the limits of what a web application can be. My computer has thousands of software components that web applications cannot call into because they are not part of the HTML standard.<sup>9</sup>

The holy grail of computing is to find a way to merge the best of the rich client and the web. Java tried but failed to build a solid cross-platform set of widgets in an extensible runtime. Maybe the programming language that replaces C/C++ on the Linux desktop will revive this possibility. This is one of the interesting remaining challenges in computing.

## Web Etc.

The web needs more personalized content. I live in Seattle, and I am a huge fan of the Seahawks football team, but this is the news article ESPN is offering to me on a website today: "[Bengals decline big Redskins offer for WR Johnson](#)." If they want me to click, they are going to have to do better than that.

Amazon.com has music samples I can listen to but that take 15 seconds to buffer before I can hear them. Music files on the web are often recorded below 192 kbps MP3, which means they sound worse than a CD and we are therefore going backwards from 1982 technology.

---

<sup>9</sup> Even if they could call into them, they couldn't install them if they weren't already on your computer.

Pictures should be big, and scale down on small screens. Drug-store.com has pictures that are at best 300 x 300 pixels:



*Can you read the text on this box, like you could in a real store?*

This is an actual “picture” of Mt. Saint Helens found on a website:



*This picture is not nearly as good as being there.*

An immersive reading experience is missing one thing: 200 dpi monitors. Reading is tiring because it takes more work for your eyes to recognize letters when they are jaggy or blurry. I saw one in 2002, and it was so pretty I couldn't pull my eyes away, but they have inexplicably been taken off the market.<sup>10</sup>

The web needs to continue to integrate with other media like TV. The best way to implement interactive TV is to visually overlay HTML on top of the TV signal. The nice thing about this is that you could turn off the distracting crawling text along the bottom of the screen if you wanted. If the TV industry wanted a few features to make it look prettier than a web page, that would be very easy to do. A French company called Free is pioneering this, using free software as part of this effort. (A cable provider is a good example of a company which doesn't want to maintain a bunch of proprietary software.)

Everyone who produces television shows should also create an XML schema which has information like who the guests are, etc. This extra information allows for a more personalized experience

<sup>10</sup> I noticed that its high-resolution broke websites which work in pixels, and so screw up when pixels get 4 times smaller, by doing things like displaying only 3 words per line.

A much smaller advance in monitor technology is to use LEDs as the backlight for LCD monitors rather than fluorescent bulbs. LEDs are more durable and more efficient than today's fluorescent bulbs. LEDs will eventually replace many uses of incandescent and fluorescent light because they are 45 times more efficient than incandescent, and 7 times more than fluorescent.

similar to what I get with the web: I could set my cable box to record whenever comedian Dennis Miller is on any channel. If a show runs over, which often happens with sports, the cable box would be smart enough to not stop recording; it needs just a tiny bit extra information to do this. Interactive TV is simply waiting for someone to create, and everyone to get behind, two simple standards, each of which would be less than 50 pages if they re-used HTML and XML. (It would also be nice to be able to watch the Seahawks no matter what city I live in. Comcast only offers me 4 football games per week.)

## Hardware

Lots of people worry about running out of Internet bandwidth, but they are just being nattering nabobs of negativism. Nippon Telephone and Telegraph of Japan demonstrated sending 14 trillion bits per second down a single strand of fiber — or 2,660 music CDs in one second. We are not running up against the limits of the laws of physics!

Metering data sounds logical, although it would seem that the distance and the number of hops a packet takes would provide a better measure. However, I counted the number of router hops required to get data from my home in Seattle to various destinations:

| Seattle to:                    | Router hops |
|--------------------------------|-------------|
| google.com (Mountain View, CA) | 23          |
| msn.co.jp (Japan)              | 23          |
| www.tmobile.de (Germany)       | 30          |
| www.latviatourism.lv (Latvia)  | 17          |
| www.google.co.uk (UK)          | 15          |
| www.gws.com.tw (Taiwan)        | 23          |

In other words, the distance to the destination was not correlated with the amount of work required to route the packet to the destination. Therefore, routing based on the number of hops might not make sense. However, if we are going to meter based on bandwidth used, how does ten cents per gigabyte sound? For \$20, my Internet service provider gives me 200 gigabytes of data transfer, an IP address, a virtualization instance, ten gigabytes of redundant disk storage, and 360 MB of RAM. I get ten cents per gigabyte, and all that other stuff for free.

1 gigabyte for ten cents allows you to send two copies of the Encyclopedia Britannica, which would be all the web traffic many would need for a whole month.

Note that the cost of transmitting packets follows Moore's law because a router is just a specialized computer. Therefore, the cost of sending a packet is dropping exponentially. In 18 months, we should ask for five cents a gigabyte. The nice thing about moving to such a model is that it would create an incentive for people to use the bandwidth. Cable companies would offer more HD content if they were getting paid more for the traffic.

# DA FUTURE

## Phase II of Bill Gates' Career

The royalty paid to us, minus our expenses of the manual, the tape and the overhead make Microsoft a break-even operation today.

—Bill Gates, Open Letter to Hobbyists, 1976



*Bill to Steve: "It was fun while it lasted."*

**I**n June 2006, we learned that Bill Gates would step down from Microsoft in June 2008. This gave plenty of warning for the markets, but it also means that Steve Ballmer will remain as CEO for a decade or more, the rumors of his demise having been greatly exaggerated.

One can presume that Bill Gates doesn't believe his legacy is furthered by spending any more time at Microsoft. Given the twin threats of free software and Google, history may judge Bill Gates more as an Andrew Carnegie than a Michelangelo.

Microsoft succeeded because it was the company that exploited Metcalfe's law to its greatest advantage. Microsoft got everyone



using MS-DOS, which it used to suck customers into Windows, Office, and everything else. Few other companies had this strategy, or the resources.

Bill Gates has provided a lot of leadership to the computing industry over the last few decades. In areas from graphical user interfaces to integrated productivity tools, to software as a service, to the web lifestyle, so his stepping down could be a greater loss to the industry than even Microsoft's demise. Who else can provide such vision, something that rallies the entire industry?

That said, Bill hasn't provided too much leadership in recent years. The last time he was on the cover of Time Magazine, he was talking about the XBox 360.



*Why is Bill not smiling? Perhaps because the XBox 360 is a PC minus a keyboard, web browser and lots of other software.*

There may be several reasons why Bill hasn't remained an iconic figure for the industry over the last few years. Part of it is that Microsoft doesn't seem to have anything new to say. HTML has changed very little in the last six years, and so Microsoft doesn't have anything to talk about regarding the web.

Secondly, what happens with Microsoft doesn't matter as much to the computer industry. Lots of people building websites are building them in PHP, a free programming language. MySQL is the second most popular database in North America, so all of those users don't care about what's included in the latest Microsoft SQL Server, or how it integrates better than ever with the latest Windows. Vista was a major release, but it has no must-have features, and so the excitement was muted. (Windows 95's major innovation was 32-bit computing.) OpenDocument format is a specification almost as important as HTML, but Microsoft doesn't support it, so Bill can't talk about it.

Bill Gates signed the ultimate Faustian bargain in the history of business: proprietary software made Microsoft the most valuable company ever created, but it was destined to fail because it didn't adopt an expansive licensing agreement that let its users contribute back to the system.

If Windows NT had adopted GPL, there would have been no reason to invent Linux. Unix predates Microsoft DOS, and so perhaps DOS and Windows wouldn't have been invented if those various Unixes had been GPL from the beginning.

A lot of software has been proprietary since computers were invented, but it is interesting to wonder where we'd be today if GPL had been the standard license agreement from the very beginning. The industry would be very different, and certainly a lot further along.

Even though free software obviates the need for Microsoft's existence, history may remember Bill Gates for many other things separate from his role in the company. The Gates Foundation has \$80 billion to spend which is enough to hire 20,000 skilled workers for 40 years. Spending \$80 billion is much harder than it sounds, especially if it is invested in projects which create value, and in turn generate revenues! Perhaps Bill could get involved in space exploration, which gives me a convenient excuse to discuss that subject.

# Space, or How Man Got His Groove Back

Midnight, July 20, 1969; a chiaroscuro of harsh contrasts appears on the television screen. One of the shadows moves. It is the leg of astronaut Edwin Aldrin, photographed by Neil Armstrong. Men are walking on the moon. We watch spellbound. The earth watches.

Seven hundred million people are riveted to their radios and television screens on that July night in 1969. What can you do with the moon? No one knew. Still, a feeling in the gut told us that this was the greatest moment in the history of life. We were leaving the planet. Our feet had stirred the dust of an alien world.

—Robert Jastrow, *Journey to the Stars*

Management is doing things right, Leadership is doing the right things!

—Peter Drucker



*SpaceShipOne was the first privately funded aircraft to go into space, and it set a number of important “firsts”, including being the first privately funded aircraft to exceed Mach 2 and Mach 3, the first privately funded manned spacecraft to exceed 100km altitude, and the first privately funded reusable spacecraft. The project is estimated to have cost \$25 million dollars and was built by 25 people. It now hangs in the Smithsonian because it serves no commercial purpose, and because getting into space has never been the challenge — it has always been the expense.*

In the 21<sup>st</sup> century, more cooperation, better software, and nanotechnology will bring profound benefits to our world, and we will put the Baby Boomers to shame. I'd like to explore one more big idea: how we can colonize space. Space, perhaps more than any other endeavor, has the ability to harness our imagination and give everyone hope for the future. When man is exploring new horizons, there is a swagger in his step.

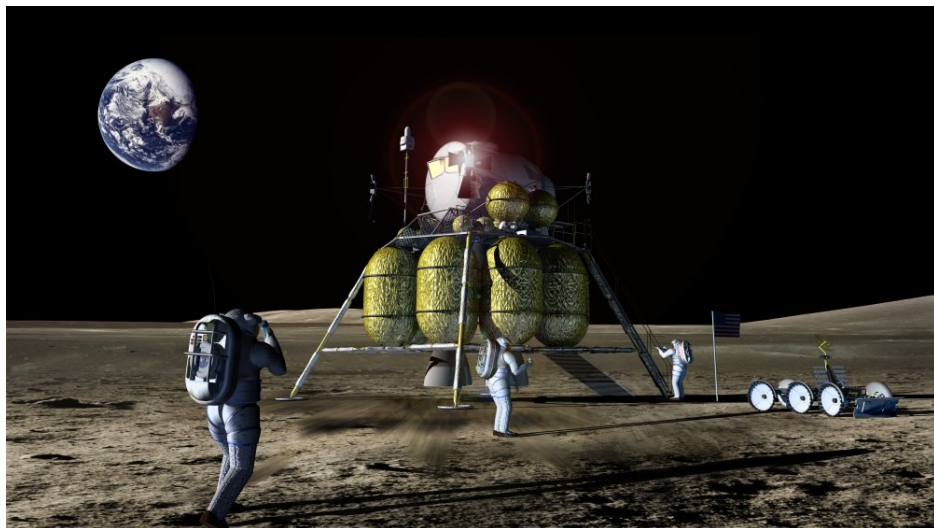
Colonizing space will change man's perspective. Hoarding is a very natural instinct. If you give a well-fed dog a bone, he will bury it to save it for a leaner day. Every animal hoards. Humans hoard money, jewelry, clothes, friends, art, credit, books, music, movies, stamps, beer bottles, baseball statistics, etc. We become very attached to these hoards. Whether fighting over \$5,000 or \$5,000,000 the emotions have the exact same intensity.

When we feel crammed onto this pale blue dot, we forget that any resource we could possibly want is out there in incomparably big numbers. If we allocate the resources merely of our solar system to all 6 billion people equally, then this is what we each get:

| Resource                | Amount                         |
|-------------------------|--------------------------------|
| Hydrogen                | 34,000 billion Tons            |
| Iron                    | 834 billion Tons               |
| Silicates (sand, glass) | 834 billion Tons               |
| Oxygen                  | 34 billion Tons                |
| Carbon                  | 34 billion Tons                |
| Energy production       | 64 trillion Kilowatts per hour |

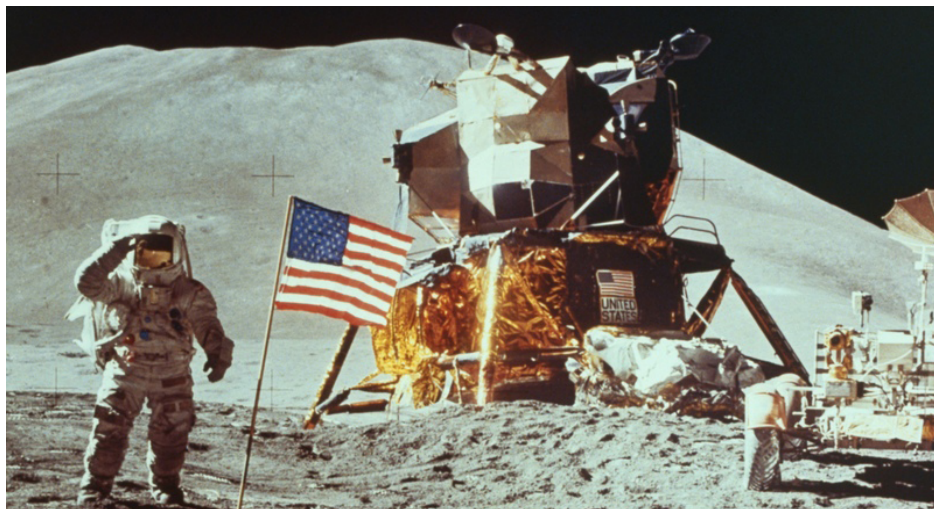
Even if we confine ourselves only to the resources of this planet, we have far more than we could ever need. This simple understanding is a prerequisite for a more optimistic and charitable society, which has characterized eras of great progress. Unfortunately, NASA's current plans are far from adding that swagger.

If NASA follows through on its 2004 vision to retire the Space Shuttle and go back to rockets, and go to the moon again, this is NASA's own imagery of what we will be looking at on DrudgeReport.com in 2020.



*Our astronauts will still be pissing in their space suits in 2020.*

According to NASA, the above is what we will see in 2020, but if you squint your eyes, it looks just like 1969:



*All this was done without things we would call computers.*

Only a government bureaucracy can make such little progress in 50 years and consider it business as usual. There are many documented cases of large government organizations plagued by failures of imagination, yet no one considers that the rocket-scientist-bureaucrats at NASA might *also* be plagued by this affliction. This is especially ironic because the current NASA Administrator, Michael Griffin, has admitted that many of its past efforts were failures:

- The Space Shuttle, designed in the 1970s, is considered a failure because it is unreliable, expensive, and small. It costs \$20,000 per pound of payload to put into low-earth orbit (LEO), a mere few hundred miles up.
- The International Space Station (ISS) is small, and only 200 miles away, where gravity is 88% of that at sea-level. It is not self-sustaining and doesn't get us any closer to putting people on the moon or Mars. (By moving at 17,000 miles per hour, it falls fast enough to stay in the same orbit.) America alone spent \$100 billion on this boondoggle.

The key to any organization's ultimate success, from NASA to any private enterprise, is that there are leaders at the top with vision. NASA's mistakes were not that it was built by the government, but that the leaders placed the wrong bets. Microsoft, by contrast, succeeded because Bill Gates made many smart bets. NASA's current goal is "flags and footprints", but their goal should be to make it easy to do those things, a completely different objective.<sup>1</sup>

I don't support redesigning the Space Shuttle, but I also don't believe that anyone at NASA has seriously considered building a next-generation reusable spacecraft. NASA is basing its decision to move back to rockets primarily on the failures of the first Space Shuttle, an idea similar to looking at the first car ever built and concluding that cars won't work.

Unfortunately, NASA is now going back to technology even more primitive than the Space Shuttle. The "consensus" in the aerospace industry today is that rockets are the future. Rockets might be in our future, but they are also in the past. The state-of-the-art in rocket research is to make them 15% more efficient. Rocket

---

1 The Europeans aren't providing great leadership either. One of the big investments of their Space agencies, besides the ISS, is to build a duplicate GPS satellite constellation, which they are doing primarily because of anti-Americanism! Too bad they don't realize that their emotions are causing them to re-implement 35 year-old technology, instead of spending that \$5 Billion on a truly new advancement. Cloning GPS in 2013: Quite an achievement, Europe!

research is incremental today because the fundamental chemistry and physics hasn't changed since their first launches in the mid-20<sup>th</sup> century.

Chemical rockets are a mistake because the fuel which propels them upward is inefficient. They have a low “specific impulse”, which means it takes lots of fuel to accelerate the payload, and even more more fuel to accelerate that fuel! As you can see from the impressive scenes of shuttle launches, the current technology is not at all efficient; rockets typically contain 6% payload and 94% overhead. (Jet engines don't work without oxygen but are 15 times more efficient than rockets.)

If you want to know why we have not been back to the moon for decades, here is an [analogy](#):

| What would taking delivery of this car cost you?  |
|---|
| A Californian buys a car made in Japan.<br>The car is shipped in its own car carrier.<br>The car is off-loaded in the port of Los Angeles.<br>The freighter is then sunk. |

The latest in propulsion technology is electrical ion drives which accelerate atoms 20 times faster than chemical rockets, which mean you need *much* less fuel. The inefficiency of our current chemical rockets is what is preventing man from colonizing space. Our simple modern rockets might be cheaper than our complicated old Space Shuttle, but it will still cost thousands of dollars per pound to get to LEO, a fancy acronym for 200 miles away. Working on chemical rockets today is the technological equivalent of polishing a dusty turd, yet this is what our esteemed NASA is doing.



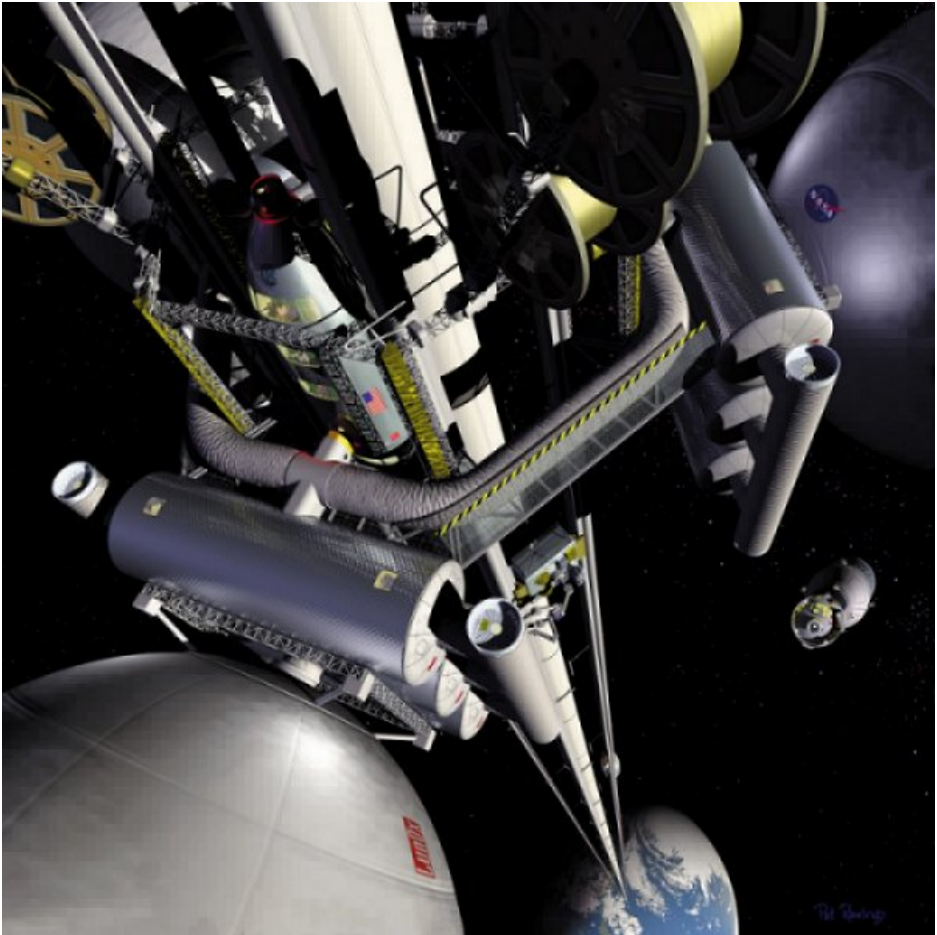
# The Space Elevator

When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is very probably wrong.

—Arthur C. Clarke RIP, 1962

The best way to predict the future is to invent it. The future is not laid out on a track. It is something that we can decide, and to the extent that we do not violate any known laws of the universe, we can probably make it work the way that we want to.

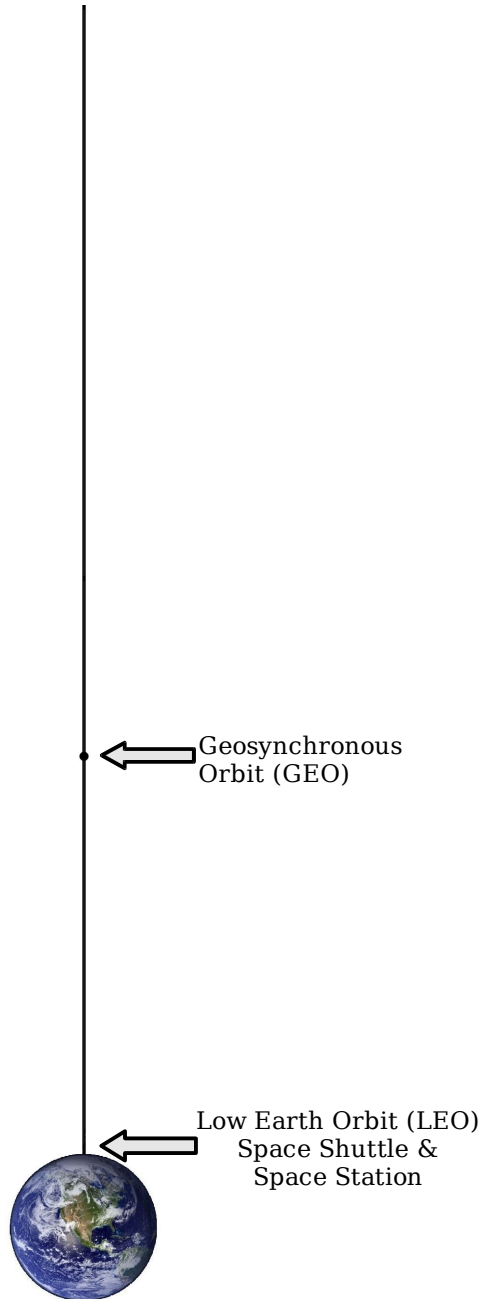
—Alan Kay



*A NASA depiction of the space elevator. A space elevator will make it hundreds of times cheaper to put a pound into space. It is an efficiency difference comparable to that between the horse and the locomotive.*



One of the best ways to cheaply get back into space is kicking around NASA's research labs:



*Scale picture of the space elevator relative to the size of Earth. The moon is 30 Earth-diameters away, but once you are at GEO, it requires relatively little energy to get to the moon, or anywhere else.*

A space elevator is a 65,000-mile tether upon which we can launch things into space in a slow, safe, and cheap way.

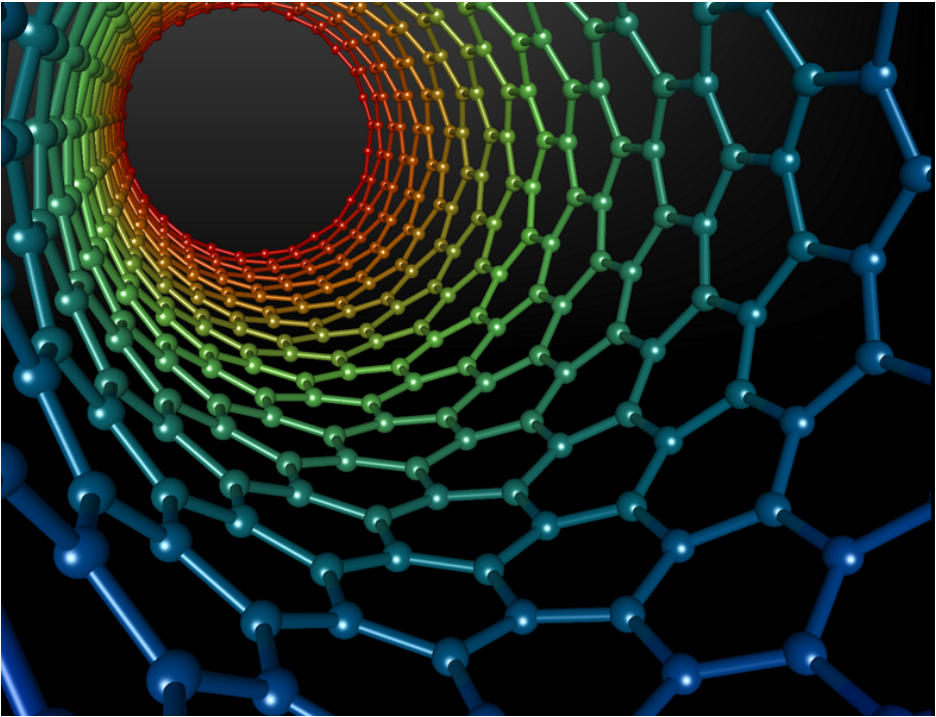
And these climber don't even need to carry their energy as you can use solar panels to provide the energy for the climbers. All this means you need *much* less fuel. Everything is fully reusable, so when you have built such a system, it is easy to have daily launches.

The first elevator's climbers will travel into space at just a few hundred miles per hour — a very safe speed. Building a device which can survive the acceleration and jostling is a large part of the expense of putting things into space today. This technology will make it hundreds, and eventually thousands of times cheaper to put things, and eventually people, into space.

A space elevator might sound like science fiction, but like many of the ideas of science fiction, it is a fantasy that makes economic sense. While you needn't trust my opinion on whether a space elevator is feasible, NASA has never officially weighed in on the topic — they haven't given it enough serious consideration.

This all may sound like science fiction, but compared to the technology of the 1960s, when mankind first embarked on a trip to the moon, a space elevator is simple for our modern world to build. In fact, if you took a cellphone back to the Apollo scientists, they'd treat it like a supercomputer and have teams of engineers huddled over it 24 hours a day. With only the addition of the computing technology of one cellphone, we might have shaved a year off the date of the first moon landing.

## Carbon Nanotubes



*Nanotubes are Carbon atoms in the shape of a hexagon. Graphic created by Michael Ströck.*

We have every technological capability necessary to build a space elevator with one exception: carbon nanotubes (CNT). To adapt a line from Thomas Edison, a space elevator is 1% inspiration, and 99% perspiration.

Carbon nanotubes are extremely strong and light, with a theoretical strength of three million kilograms per square centimeter; a bundle the size of a few hairs can lift a car. The theoretical strength of nanotubes is far greater than what we would need for our space elevator; current baseline designs specify a paper-thin, 3-foot-wide ribbon. These seemingly flimsy dimensions would be strong enough to support their own weight, and the 10-ton climbers using the elevator.

Nanotechnology will bring many changes to our world in the next few decades. Materials sciences will be one of the biggest challenges occupying our minds in the 21<sup>st</sup> century. The nanotubes we need for our space elevator are the perfect place to start because,

unlike biological nanotechnology research, which uses hundreds of different atoms in extremely complicated structures, nanotubes have a rather trivial design.

The best way to attack a big problem like nanotechnology is to first attack a small part of it, like carbon nanotubes. A “Manhattan Project” on general nanotechnology does not make sense because it is too unfocused a problem, but such an effort might make sense for nanotubes. Or, it might simply require the existing industrial expertise of a company like Intel. Intel is already experimenting with nanotubes inside computer chips because metal loses the ability to conduct electricity at very small diameters. But no one has asked them if they could build ropes made of nanotubes.

The US government has increased investments in nanotechnology recently, but we aren't seeing many results. From space elevator expert Brad Edwards:

There's what's called the National Nanotechnology Initiative. When I looked into it, the budget was a billion dollars. But when you look closer at it, it is split up between a dozen agencies, and within each agency it's split again into a dozen different areas, much of it ends up as \$100,000 grants. We looked into it with regards to carbon nanotube composites, and it appeared that about thirty million dollars was going into high-strength materials — and a lot of that was being spent internally in a lot of the agencies; in the end there's only a couple of million dollars out of the billion-dollar budget going into something that would be useful to us.

The money doesn't have focus, and it's spread out to include everything. You get a little bit of effort in a thousand different places. A lot of the budget is spent on one entity trying to play catch-up with whoever is leading. Instead of funding the leader, they're funding someone else internally to catch up.

Again, here is a problem similar to the one we find in software today: people playing catchup rather than working together. I don't know what nanotechnology scientists do every day, but it sounds like they would do well to follow in the footsteps of our free software pioneers and start cooperating.

The widespread production of nanotubes could be the start of a nanotechnology revolution. And the space elevator, the killer app of nanotubes, will enable the colonization of space.

## Why?

William Bradford, speaking in 1630 of the founding of the Plymouth Bay Colony, said that all great and honorable actions are accompanied with great difficulties, and both must be enterprised and overcome with answerable courage.

There is no strife, no prejudice, no national conflict in outer space as yet. Its hazards are hostile to us all. Its conquest deserves the best of all mankind, and its opportunity for peaceful cooperation may never come again. But why, some say, the moon? Why choose this as our goal? And they may well ask why climb the highest mountain? Why, 35 years ago, fly the Atlantic? Why does Rice play Texas?

We choose to go to the moon. We choose to go to the moon in this decade and do the other things, not because they are easy, but because they are hard, because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we intend to win, and the others, too.

It is for these reasons that I regard the decision last year to shift our efforts in space from low to high gear as among the most important decisions that will be made during my incumbency in the office of the Presidency.

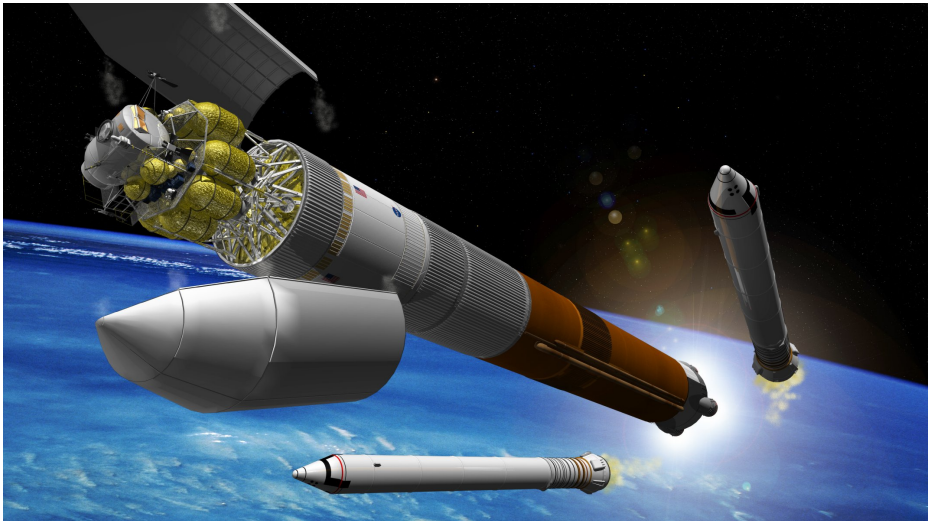
In the last 24 hours we have seen facilities now being created for the greatest and most complex exploration in man's history. We have felt the ground shake and the air shattered by the testing of a Saturn C-1 booster rocket, many times as powerful as the Atlas which launched John Glenn, generating power equivalent to 10,000 automobiles with their accelerators on the floor. We have seen the site where five F-1 rocket engines, each one as powerful as all eight engines of the Saturn combined, will be clustered together to make the advanced Saturn missile, assembled in a new building to be built at Cape Canaveral as tall as a 48 story structure, as wide as a city block, and as long as two lengths of this field.

The growth of our science and education will be enriched by new knowledge of our universe and environment, by new techniques of learning and mapping and observation, by new tools and computers for industry, medicine, the home as well as the school.

I do not say the we should or will go unprotected against the hostile misuse of space any more than we go unprotected against the hostile use of land or sea, but I do say that space can be explored and mastered without feeding the fires of war, without repeating the mistakes that man has made in extending his writ around this globe of ours.

We have given this program a high national priority — even though I realize that this is in some measure an act of faith and vision, for we do not now know what benefits await us. But if I were to say, my fellow citizens, that we shall send to the moon, 240,000 miles away from the control station in Houston, a giant rocket more than 300 feet tall, the length of this football field, made of new metal alloys, some of which have not yet been invented, capable of standing heat and stresses several times more than have ever been experienced, fitted together with a precision better than the finest watch, carrying all the equipment needed for propulsion, guidance, control, communications, food and survival, on an untried mission, to an unknown celestial body, and then return it safely to earth, re-entering the atmosphere at speeds of over 25,000 miles per hour, causing heat about half that of the temperature of the sun — almost as hot as it is here today — and do all this, and do it right, and do it first before this decade is out — then we must be bold.

—[John F. Kennedy](#), September 12, 1962



*Lunar Lander at the top of a rocket. Rockets are expensive and impose significant design constraints on space-faring cargo.*

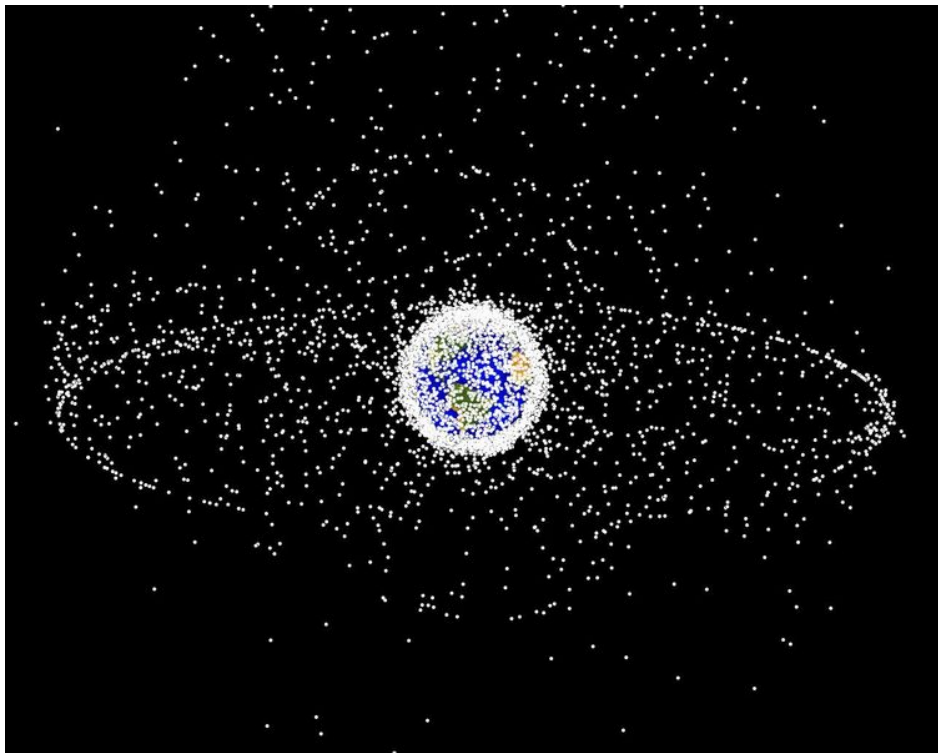
NASA has 18,000 employees and a \$16-billion-dollar budget. Even with a fraction of those resources, their ability to oversee the design, handle mission control, and work with many partners is more than equal to this task.

If NASA doesn't build the space elevator, someone else might, and it would change almost everything about how NASA does things today. NASA's tiny (15-foot-wide) new Orion spacecraft, which was built to return us to the moon, was designed to fit atop a rocket and return the astronauts to Earth with a 25,000-mph thud, just like in the Apollo days. Without the constraints a rocket imposes, NASA's

spaceship to get us back to the moon would have a very different design. NASA would need to throw away a lot of the R&D they are now doing if a space elevator were built.

Another reason the space elevator makes sense is that it would get the various scientists at NASA to work together on a big, shared goal. NASA has recently sent robots to Mars to dig two-inch holes in the dirt. That type of experience is similar to the skills necessary to build the robotic climbers that would climb the elevator, putting those scientists to use on a greater purpose.

Space debris is a looming hazard, and a threat to the ribbon:



*Map of space debris. The US Strategic Command monitors 10,000 large objects to prevent them from being misinterpreted as a hostile missile. China blew up a satellite in January, 2007 which created 35,000 pieces of debris larger than 1 centimeter.*

The space elevator provides both a motive, and a means to launch things into space to remove the debris. (The first elevator will need to be designed with an ability to move around to avoid debris!)

Once you have built your first space elevator, the cost of building the second one drops dramatically. A space elevator will eventually

make it \$10 per pound to put something into space. This will open many doors for scientists and engineers around the globe: bigger and better observatories, a spaceport at GEO, and so forth.

Surprisingly, one of the biggest incentives for space exploration is likely to be tourism. From Hawaii to Africa to Las Vegas, the primary revenue in many exotic places is tourism. We will go to the stars because man is driven to explore and see new things.

Space is an extremely harsh place, which is why it is such a miracle that there is life on Earth to begin with. The moon is too small to have an atmosphere, but we can terraform Mars to create one, and make it safe from radiation and pleasant to visit. This will also teach us a lot about climate change, and in fact, until we have terraformed Mars, I am going to assume the global warming alarmists don't really know what they are talking about yet.<sup>2</sup> One of the lessons in engineering is that you don't know how something works until you've done it once.

Terraforming Mars may sound like a silly idea today, but it is simply another engineering task.<sup>3</sup> I worked in several different groups at Microsoft, and even though the set of algorithms surrounding databases are completely different from those for text engines, they are all engineering problems and the approach is the same: break a problem down and analyze each piece. (One of the interesting lessons I learned at Microsoft was the difference between real life and standardized tests. In a standardized test, if a question looks hard, you should skip it and move on so as not to waste precious time. At Microsoft, we would skip past the easy problems and focus our time on the hard ones.)

Engineering teaches you that there are an infinite number of ways to attack a problem, each with various trade-offs; it might take 1,000 years to terraform Mars if we were to send one ton of material, but only 20 years if we could send 1,000 tons of material. Whatever we finally end up doing, the first humans to visit Mars will be happy that we turned it green for them. This is another way our generation can make its mark.

---

2 Carbon is not a pollutant and is valuable. It is 18% of the mass of the human body, but only .03% of the mass of the Earth. If Carbon were more widespread, diamonds would be cheaper. Driving very fast cars is the best way to unlock the carbon we need. Anyone who thinks are running out of energy doesn't understand the algebra in  $E = mc^2$ .

3 Mars' moon, Phobos, is only 3,700 miles above Mars, and if we create an atmosphere, it will slow down and crash. We will need to find a place to crash the fragments, I suggest in one of the largest canyons we can find; we could put them next to a cross dipped in urine and call it the largest man-made art.



A space elevator is a doable mega-project, but there is no progress beyond a few books and conferences because the very small number of people on this planet who are capable of initiating this project are not aware of the feasibility of the technology.

Brad Edwards, one of the world's experts on the space elevator, has a PhD and a decade of experience designing satellites at Los Alamos National Labs, and yet he has told me that he is unable to get into the doors of leadership at NASA, or the Gates Foundation, etc. Setting aside the nanotechnology, we might need only 5,000 man-years of work (NASA alone has 50,000 employees) to accomplish this task, but no one who has the authority to organize this understands that a space elevator is doable.

Glenn Reynolds has blogged about the space elevator on his very influential Instapundit.com, yet a national dialog about this topic has not yet happened, and NASA is just marching ahead with its expensive, dim ideas. My book is an additional plea: one more time, and with feeling!

## How and When

It does not follow from the separation of planning and doing in the analysis of work that the planner and the doer should be two different people. It does not follow that the industrial world should be divided into two classes of people: a few who decide what is to be done, design the job, set the pace, rhythm and motions, and order others about; and the many who do what and as they are told.

—Peter Drucker

There are a many interesting details surrounding a space elevator, and for those interested in further details, I recommend *The Space Elevator*, co-authored by Brad Edwards.

The size of the first elevator is one of biggest questions to resolve. If you were going to lay fiber optic cables across the Atlantic ocean, you'd set aside a ton of bandwidth capacity. Likewise, the most important metric for our first space elevator is its size.

The one other limitation with current designs is that they assume climbers which travel hundreds of miles per hour. This is a fine speed for cargo, but it means that it will take days to get into orbit. If we want to send humans into space in an elevator, we need to build climbers which can travel at 10,000 miles per hour. While this seems ridiculously fast, if you accelerate to this speed over a period of minutes, it will not be jarring. Perhaps this should be the challenge for version two if they can't get it done the first time.

The conventional wisdom amongst those who think it is even possible is that it will take between 20 and 50 years to build a space elevator. However, anyone who makes such predictions doesn't understand that engineering is a fungible commodity. Two people will, in general, accomplish something twice as fast as one person.<sup>4</sup> How can you say something will unequivocally take a certain amount of time when you don't specify how many resources it will require or how many people you plan to assign to the task?

Efficiency drops as teams get larger, but with the Internet, a superb tool of communication and collaboration, thousands of people can work together efficiently. Manufacturers are using the Internet to reinvent how they interact with their suppliers during the design process – shaving years off the design. Boeing's newest 787 “Dreamliner” airplane will go from project start to take-off in just four years, which is half the time it took them to design the 777. (Why Boeing offered their 30-year old 767 instead of the 787 to the US Military, as the workhorse of their new refueling fleet, was an insanely backward-looking mistake. If Boeing doesn't think their improvements are worthwhile, why are they making them?)

Furthermore, predictions are usually way off. If you asked someone how long it would take unpaid volunteers to make Wikipedia as big as the Encyclopedia Britannica, no one would have guessed the correct answer of two and a half years. From creating a space elevator to world domination by Linux, anything can happen in far less time than we think is possible if everyone simply steps up to play their part. The way to be a part of the future is to invent it, by unleashing our scientific and creative energy towards big, shared goals. Wikipedia, as our encyclopedia, was an inspiration to millions of people, and so the resources have come piling in. The way to get help is to create a vision that inspires people.

In a period of 75 years, man went from using horses and wagons to landing on the moon. Why should it take 30 years to build something that is 99% doable today?

Many of the components of a space elevator are simple enough that college kids are building prototype elevators in their free time. The Elevator:2010 contest is sponsored by NASA, but while these

---

4 Fred Brooks' *The Mythical Man-Month* argues that adding engineers late to a project makes a project later, but ramp-up time is just noise in the management of an engineering project. Also, wikis, search engines, and other technologies invented since his book have lowered the overhead of collaboration.

contests have generated excitement and interest in the press, they are building toys, much like a radio-controlled airplane is a toy compared to a Boeing airliner.

I believe we could have a space elevator built in 7 years. If you divvy up three to four years of work per person, and add in some time to ramp up and test, you can see how seven years is quite reasonable. Man landed on the moon 7 years after Kennedy's speech, exactly as he ordained, because dates can be self-fulfilling prophecies. It allows everyone to measure themselves against their goals, and determine if they need additional resources.

If the design of the hardware and the software were done in a public fashion, others could take the intermediate efforts and test them and improve them, therefore saving further engineering time. Perhaps NASA could come up with hundreds of truly useful research projects for college kids to help out on instead of encouraging them to build toys.

The Unknown Unknown is the nanotubes, but nearly all the other pieces can be built without having any access to them. We will only need them wound into a big spool on the launch date.

I can imagine that any effort like this would get caught up in a tremendous amount of international political wrangling that could easily add years on to the project. We should not let this happen, and we should remind each other that the space elevator is just the railroad car to space — the exciting stuff is the cargo inside and the possibilities out there. A space elevator is not a zero sum endeavor: it would enable lots of other big projects that are totally unfeasible currently. A space elevator would enable various international space agencies that have money, but no great purpose, to work together on a large, shared goal. And as a side effect it would strengthen international relations.<sup>5</sup>

---

5 Perhaps the Europeans could build the station at GEO. Russia could build the shuttle craft to move cargo between the space elevator and the moon. The Middle East could provide an electrical grid for the moon. China could take on the problem of cleaning up the orbital space debris and build the first moon base. Africa could attack the problem of terraforming Mars, etc.

## 21<sup>st</sup> Century Renaissance

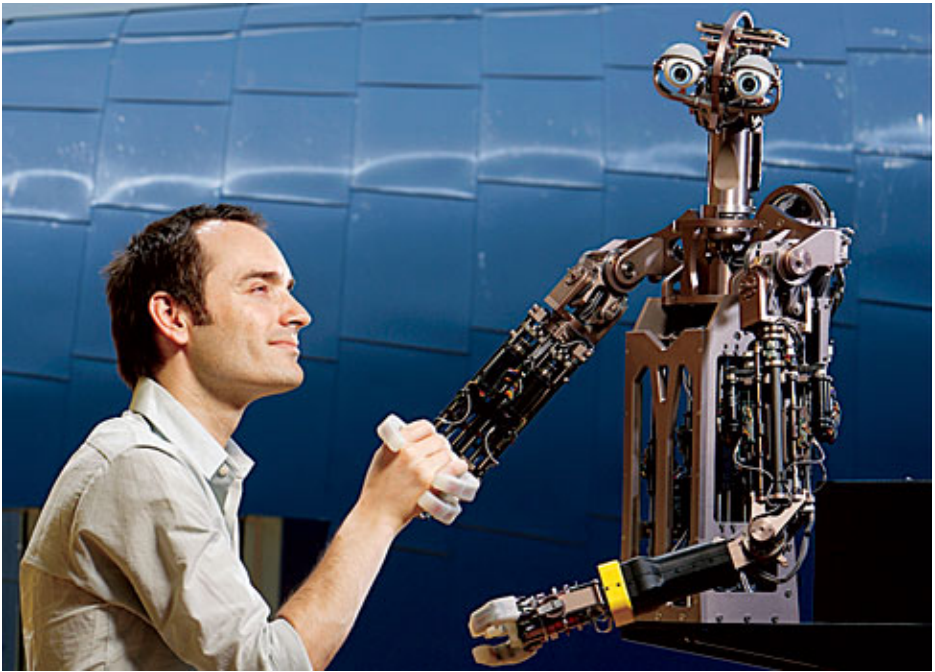
The great achievements of western capitalism have rebounded primarily to the benefit of the ordinary person.

—Milton Friedman

If we have learned one thing from the history of invention and discovery, it is that, in the long run, and often in the short one, the most daring prophecies seem laughably conservative.

—Arthur C. Clarke, 1951

As I see it, I still live in the 20<sup>th</sup> century. History will remember the 21<sup>st</sup> century as the time when man entered a new Renaissance, and when this was not just a photo shoot.



*If we could replace journalists with robots, the issue of media bias would disappear, and they would become better looking.*

Even in our still-primitive world of today, I would rather be making \$30,000 a year than be promised \$100 million dollars if I walked through a door back into 1986, and I say that not just because of the big hair. The Internet was still seven years away from its first web page in 1986. We might not be able to go back in time, but in general we wouldn't want to.

Many people don't appreciate how much faster the world is moving every day with the creation of the Internet and other modern

technologies. Some say: "The only constant is change", but this is wrong: the only constant is acceleration, an increasing rate of change. People don't feel the acceleration yet because the world is moving so slowly. Energy prices are high because we haven't built a nuclear power plant in 30 years (blame the Democrats), and building nuclear power plants supposedly takes 15 years (blame the bureaucrats). It took decades to roll out HDTV. Genetically modified foods are treated as Frankensteins of nature. Drug companies hire as many lawyers as researchers, and then we wonder why drug costs are so high, and why it takes years to release new ones. Devices are disintegrated and stupid. Waiting in line is considered a normal part of life.

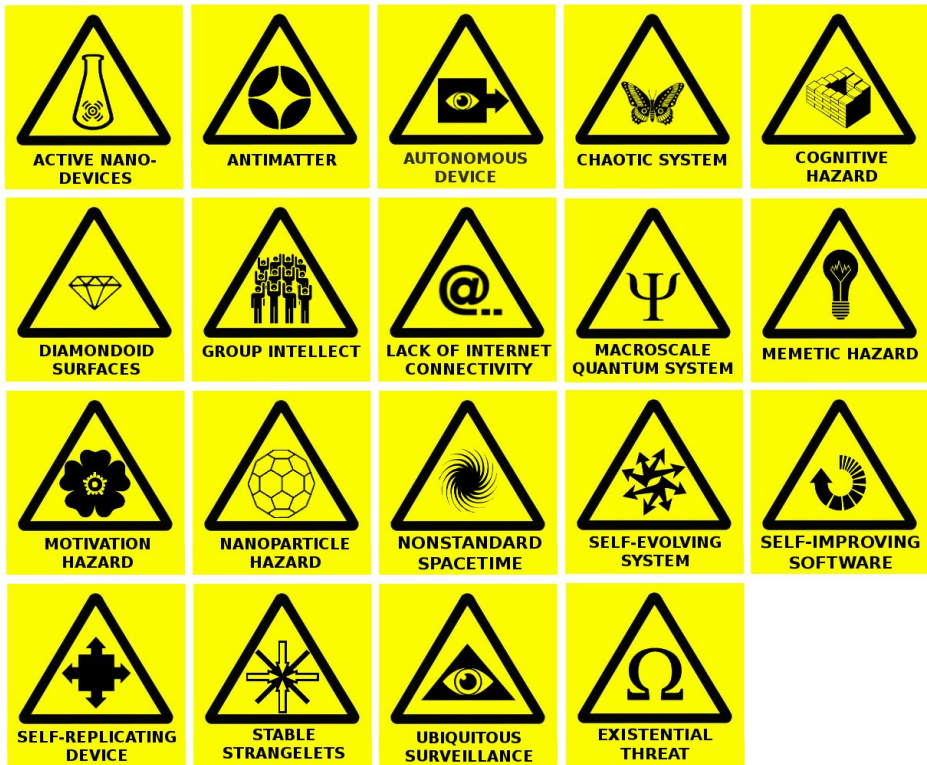
The gap between the rich and the poor doesn't cause social instability because Bill Gates doesn't live materially better than anyone else. His contact lenses are no better than mine, and neither of us owns a robot-driven car.

With better cooperation and better tools, the 21<sup>st</sup> century Renaissance is waiting for us. The world there will be characterized by a gift economy, so resist the urge to hoard. Once we cross that line, we will get our sanity and happiness back. Let's hurry!

Do not fear the future; man was created to solve problems and gain enjoyment from that process. The question as to whether the United States will be relevant in the 21<sup>st</sup> century depends on the responses to these simple questions: will it have the most scientists, and are they learning from each other, and therefore working together?

The End

# Warning Signs From the Future



*This is our scary future, and unfortunately mankind is crawling towards it.<sup>6</sup>*

6 From <http://lifeboat.com/ex/warning.signs.for.tomorrow>, created by Anders Sandberg. I sent multiple e-mails to various e-mail addresses at the Lifeboat Foundation to try to get permission to use these images, but I never received a response. Perhaps they are too busy with their mission of “encouraging scientific advancements to help humanity survive existential risks” to respond. So I just donated \$200 for their use here :-)

We go forward with complete confidence in the eventual triumph of freedom. Not because history runs on the wheels of inevitability; it is human choices that move events. Not because we consider ourselves a chosen nation; God moves and chooses as He wills. We have confidence because freedom is the permanent hope of mankind.

When our Founders declared a new order of the ages; when soldiers died in wave upon wave for a union based on liberty; when citizens marched in peaceful outrage under the banner "Freedom Now" — they were acting on an ancient hope that is meant to be fulfilled. History has an ebb and flow of justice, but history also has a visible direction, set by liberty and the Author of Liberty.

—US Presidential Inaugural Address, 2005

Genius is eternal patience.

The greater danger for most of us lies not in setting our aim too high and falling short; but in setting our aim too low, and achieving our mark.

The true work of art is but a shadow of the Divine perfection.

—Michelangelo

As simple as possible, but not simpler.

—Albert Einstein

Most of the things worth doing in the world had been declared impossible before they were done.

—Louis Brandeis, US Supreme Court Justice

We have only one alternative: either to build a functioning industrial society or see freedom itself disappear in anarchy and tyranny.

—Peter Drucker

# AFTERWORD

## US v. Microsoft

The political principle that underlies the market mechanism is unanimity. In an ideal free market resting on private property, no individual can coerce any other; all cooperation is voluntary, all parties to such cooperation benefit or they need not participate. There are no values, no “social” responsibilities in any sense other than the values and responsibilities of individuals. Society is a collection of individuals and of the various groups they voluntarily form.

If an exchange between two parties is voluntary, it will not take place unless both believe they will benefit from it. Most economic fallacies derive from the neglect of this simple insight, from the tendency to assume that there is a fixed pie, that one party can only gain at the expense of another.

—Milton Friedman

Microsoft got its tush handed to it in the DOJ trial, but that was because it lost credibility. For example, Bill Gates argued that he wasn't worried about Netscape. If so, why did Microsoft VIPs say they wanted to “smother” Netscape and “cut off their air supply”? Judge Jackson wrote that Microsoft's witnesses in the trial: “proved, time and time again, to be inaccurate, misleading, evasive, and transparently false.” However, what they say at a trial has nothing to do with how they behaved in the marketplace.

Fisher, the government's economist argued that it was “a joke” that Windows was challenged by the Macintosh or Linux. With just a few years hindsight, and the tremendous potential of free software, it becomes very clear that their economist was wrong. Furthermore, in the trial he admitted that he could find no specific harm that Microsoft had done. Microsoft might be a hard-charging competitor, but the software they wrote has been invaluable to the world, and they provided it at a cost much lower than many of their competitors like Sun, IBM and Oracle.

In addition to the Mac and Linux, the web was and is a huge threat to Microsoft — the PC development community used to revolve around Windows apps, but now it is all about web apps.

The government's major argument was that bundling a web browser into the operating system was illegal. However, including a browser with an operating system is a good idea for consumers. Without a web browser, you wouldn't be able to surf the web or send



e-mail. Piecing together an operating system as a jumble of parts is a lot of work and should not be done by the end user. Every operating system today includes a web browser. If the government had its way, progress in Microsoft's software would be frozen with early 1990's technology.

Furthermore, the government never had a clear proposed remedy. The trial court's plan was to split Microsoft up into Windows, Office, and other parts, which would have been an unprecedented intrusion into a private company, and yet wouldn't have decreased the popularity of Windows or Office.

The fawning and shallow press focused more on David Boies' brilliance and ability to quote obscure legislation from memory than the important facts and issues of the case he was making. Almost everything Microsoft did was done by other people in the industry. It was only because Microsoft became the biggest company on the block that it became illegal. Microsoft was a monopoly, and therefore was behaving badly, even when it was acting the same as everyone else. And, no one could describe when Microsoft had crossed that line.

Microsoft succeeded because it built the best products. That Sun, Netscape, Apple and other companies just as proprietary as Microsoft would complain is disingenuous. They were jealous and afraid, and used the government to help them. As Richard Stallman points out, all proprietary technologies are monopolies. If you want to use an Oracle database, good luck learning about it without reading any of Oracle's documentation. If your database crashes, who are you going to call? Oracle is just as much a monopoly as Microsoft, just a less popular one.

Free software, better programming languages, standard formats, and the Internet have long been a threat to Microsoft, it just has not caused Microsoft's revenue to decrease, yet. Milton Friedman wrote that the only kind of monopoly that is a problem is a "coercive monopoly" which are created by government. In this case, there is no need to take an offending company to trial, but to remove the government regulation which created the monopoly in the first place. It would have been interesting if Microsoft would have made *only* this argument, and then rested.

# Microsoft as a GPL Software Company

Prediction is very difficult, especially about the future.

—Niels Bohr

I wrote this book under the assumption that if Microsoft continues on its current path and never adopts copyleft licenses for its code, what Wikipedia did to Encarta will likewise happen to Windows, Office, Internet Explorer, SQL Server, Visual Studio, Exchange, MSN, etc.

A tiny fraction of the software Microsoft produces runs on Linux, so Linux world domination means the end of Microsoft as we know it. Free software will force Microsoft to choose between licensing revenue and relevance.

A lot of people that I meet in the software industry hate Microsoft, but have valued its leadership. Many people recognize that Office is more polished than OpenOffice and pioneered many innovations. Internet Explorer was universally recognized for a significant period of time as a better browser than Netscape's. Microsoft's SQL Server is considered easier to use than Oracle's. Word beat Ami Pro and WordPerfect in the reviews.

Let us presume that Microsoft were to embrace the proposition that free software is a good idea. There would be no reason to call Microsoft "evil" anymore, because copyleft would prevent it. When something is free, no one can control or monopolize it anymore. In adopting free software, they would have two possibilities available to it, however, both involve a dramatic change in what their employees do, and a significant drop in revenue.

Microsoft could release all of its code as GPL, learn to build this software as a globally distributed effort, and compete and better interoperate with other existing free software. I think this would be a mistake because Microsoft's codebases are too old, much bigger and more complicated than existing free software codebases, and the details today are not understood by the existing globally distributed software community.

If Microsoft were to release its code as GPL it isn't clear if the existing free software community would switch, and would create confusion and further fragmentation, an inefficiency which is already a huge problem. Therefore I will say little more about this possibility.

Even more dramatic is for Microsoft to adopt Linux and other free software. Microsoft could easily become the dominant Linux player. Mark Shuttleworth invested only a few million dollars to create Ubuntu and become the most popular Linux distribution, so imagine what Microsoft's billions and army could do? Microsoft could even develop mechanisms to make it easy for people and enterprises to move from Windows to Linux, a challenge they are uniquely able to tackle.

This transition would involve a complete upheaval of the code-base almost every Microsoft employee works on, and would require they find an alternate source of revenue for their most profitable products. But they would likely find a way to employ all their existing engineers and using just their cash on hand, they could hire every one of their engineers for 5 years; in other words, they would have many years to sort out the revenue situation during the transition.

As there is no free vision recognition, speech, or search engine with any critical mass, Microsoft could lead those efforts and lead the future even more dominantly than it owns the present! With the rest of the world helping out, the value of these offerings would be much greater than the proprietary set that Microsoft is currently offering today.

One way Microsoft could jumpstart this transformation would be to purchase Novell or Ubuntu, or just by hiring Debian developers to help seed their existing teams and to cross-pollinate the culture. In early-2008, Novell's market capitalization was \$2.2 billion, a mere 5% of the value of their recent acquisition target Yahoo!, and this is the most expensive way to jumpstart work on Linux.<sup>1</sup>

My one concern about this idea is that after criticizing Microsoft for: building a kernel inferior to Linux, for shipping too infrequently, for not building software with the same reliability that Boeing builds into their airplanes, for not sufficiently embracing standards, for being crippled by backward compatibility that maybe it wouldn't work out that well. Furthermore, I made a switch to zero Microsoft code smoothly and I believe the rest of the world can as well. Therefore if Microsoft made many mistakes, and we don't need them, should we entrust the future to them?

However, if Microsoft were to embrace the GPL, better tools, ship-

---

1 One advantage of buying Novell is that Microsoft would get Mono, the free .Net runtime, and it could use it as a base to build the next-generation programming language, supplanting Sun. However, this would eventually involve merging it with their own .Net codebase.

ping more frequently, having one tree the world can collaborate in to evolve more smoothly, and with a greater emphasis on open standards created in parallel with code, their next stack would be a lot better than their current one.

These changes would allow Microsoft to not necessarily accumulate the baggage they've acquired in their current software, but I would just like to repeat that the most common reason why code is not reliable is that it is too complicated.<sup>2</sup> Whether that complexity is caused by backward compatibility, tools, Intelligent Design, shipping infrequently, too much of a focus on performance rather than code cleanliness, age, or the number of customers, I cannot say, but it is a problem that the rest of the free software community should learn from. Complexity is not only the enemy of reliability, it is the enemy of progress.

This is idle speculation if Microsoft does not adopt free software.

---

2 For example, SQL Server has almost an entire OS inside it with its own memory manager, cache manager, file system, synchronization mechanisms, threads, inter-process communication, code loading, security subsystem, etc. and it still has multithreading bugs. Huge portions of .Net are written in C++ because they thought it would be a few percent faster, but this also has added complexity and bugs and slows progress.

# The Outside World

In 1929, cars were unreliable and belched all sorts of noxious fumes into the atmosphere. Computers did not exist. Airplanes were slow and unsafe. Movies had just recently gotten sound. TV and FM radio did not exist. The iPod and cell phone were misspellings. DNA was just three letters and genetics non-existent. The stock market crashed and we were on our way to a depression.

In 2009, cars are reliable and produce little noxious gases. There are billions of computers in the world, especially when you count the ones in cars, thermostats, cell phones, etc. Airplanes are fast, safe, and affordable. Home entertainment centers have surround sound and quality so good you think you are in the movie theater. TV is digital and FM is everywhere. Close to 200 million iPods and billions of cell phones have been sold. The human genome has been sequenced multiple times...and the stock market crashed. Many argue that we are headed for a deep recession, if not depression. It seems more than obvious to me that we need to import some of the methods of knowledge transfer from the technologists into the rest of our life structures.

—Seymour Friedel

Congress, the press, and the bureaucracy too often focus on how much money or effort is spent, rather than whether the money or effort actually achieves the *announced* goal.

—Donald Rumsfeld, 1974

Just as unskilled manual workers in manufacturing were the dominant social and political force in the 20<sup>th</sup> century, knowledge technologists are likely to become the dominant social — and perhaps also political — force over the next decades.

—Peter Drucker

The challenge for my generation was to provide an intellectual defense of economic liberty. The challenge for your generation is to keep it.

—Milton Friedman

Eben Moglen says that one of the goals of the free software movement is to expand opportunities for billions more people out there — to quit throwing away most of the brains on earth. This book is about free software, but I'd like to end this Afterword with a few ideas on a free press, free markets and several other issues. One cannot write a book about the future of technology without consideration of the outside world. Our scientists can tell us that one pound of Uranium generates the same amount of power as three million pounds of Coal, but whether we choose to use that fact is a decision

of the Congress. Letting the Lawyers decide whether nuclear power is a good idea was a *bad* idea. “Unleashing the power of the atom” was an idea popularized in the 1950s, but it stalled in the 1970s and it still hasn't arrived yet!

If it weren't for our scientists and engineers, we'd still be picking our noses in caves, and the lawyers in divorce proceedings would be fervently arguing about how to divide up the rocks. It's the outside world which determines whether we will put our scientists to work, or sue them. High taxes might have the benefit of taking money away from the “evil corporations”, but they also leave less money to spend on R&D.

Reviewers told me that this material belongs in another book, but the problem is that everything here has already been written — it just hasn't been read and accepted yet! If we knowledge technologists are to become the dominant political force, then there are a few things I think we should understand.

## Powering a free society

After a quarter-century of gas tax hikes, a ban on drilling for oil and a complete destruction of the nuclear power industry in America, I guess liberals can declare: Mission accomplished!

In response to skyrocketing gas prices, they say, practically in unison, “We can't drill our way out of this crisis.”

What does that mean? This is like telling a starving man, “You can't eat your way out of being hungry!” Finding more oil isn't going to increase the supply of oil?

It is the typical Democratic strategy to babble meaningless slogans, as if they have a plan. Their plan is: the permanent twilight of the human race. It's the only solution they can think of to deal with the beastly traffic on the LIE (Long Island Expressway).

Liberals complain that — as Barack Obama put it — there's “no way that allowing offshore drilling would lower gas prices right now. At best you are looking at five years or more down the road.”

This is as opposed to airplanes that run on woodchips, which should be up and running any moment now. Moreover, what was going on five years ago? Why didn't anyone propose drilling back then?

Say, you know what we need? We need a class of people paid to anticipate national crises and plan solutions in advance. It would be such an important job, the taxpayers would pay them salaries so they wouldn't have to worry about making a living and could just sit around anticipating crises.

If only we had had such a group — let's call them "elected representatives" — they could have proposed drilling five years ago!

But of course we do pay people to anticipate national problems and propose solutions. Some of them — we'll call them Republicans — did anticipate high gas prices and propose solutions.

—Jim Bangs

I went through thick and thin in the nuclear plant licensing process. The Shoreham, New York, plant was in operation. It got up to well over 1% power, and then they shut it down. Democrat Governor Mario Cuomo made 'em drill holes in the reactor vessel so it could never be used again. And it was the first plant that had been built under the newest and safest Environmental Protection Agency safety regulations, and it took them years. It cost us well over a billion dollars, and they just wrote it off.

—Jim, Shoreham Nuclear Regulatory Commission Project Manager

The Alvin W. Vogtle Nuclear Generating Station in Burke County, Georgia, has two 1,200-MW reactors sitting on the Savannah River, directly across from the federal nuclear processing facilities in South Carolina. Now Southern Nuclear, which owns Vogtle, wants to build two new 1,000-MW reactors as part of the nuclear renaissance.

Environmental groups have immediately taken up the challenge, arguing that dredging the Savannah River to allow barge delivery of reactor parts will damage the river. The Savannah was dredged regularly for more than a century until the Army Corps of Engineers gave up in 1980 because nothing much was happening on the river. Now environmental groups say a renewal will ruin the environment. The Nuclear Regulatory Commission has nodded agreement and will require an environmental impact statement before early site clearance can begin. That will probably add three years to the project.

—William Tucker

Leonardo Cavallaro, a man who designed nuclear power plants until the mid-70s when the industry collapsed, told me that "Energy changes everything." (An interesting article explaining how nuclear power got killed in the United States is located at <http://tinyurl.com/WhoKilledNuclearPower>.) Each invention which decreased the amount of energy man himself had to expend to accomplish a task increased his quality of life. The use of the animals, water wheels to mill grain, steam, electricity and batteries have each made our lives easier. Few things would further increase our quality of life more than making energy ten times cheaper than it is today.

One of the biggest reasons for offshoring of manufacturing jobs outside of the US is the cost of energy. It takes a lot of energy to bend steel! The idea that it was okay to offshore manufacturing because America is moving to a service economy is one of the dumbest ideas ever uttered because it doesn't recognize that it isn't until you do something that you figure out *how* to do it.

Every step that you take both solves today's problems, and sets you up to take on new problems. At Microsoft, we had a phrase "Crawl, Walk, Run" and it applies to other sectors as well as software:

Make TVs -> Make robots which make TVs -> Make robots which make your bed

99 cent plastic toy -> materials sciences for toys -> materials sciences changing all building materials

Food production -> genetically modified food -> genetically modified everything

10 years ago, President Clinton vetoed legislation to increase drilling for oil inside the US because he said it wouldn't do anything for 10 years. Today, even with our record gas prices, many in Congress say they are against it because it won't do anything for 10 years. Either people in our government are incapable of thinking long-term, or they have been bribed by special interests and treat us like fools.



## Free Markets



*Congress creates the problem, blames the free market, and uses the crisis as an excuse to create more government.*

A man's admiration for absolute government is proportionate to the contempt he feels for those around him.

—Alexis de Tocqueville

So you think that money is the root of all evil? Have you ever asked what is the root of money? Money is a tool of exchange, which can't exist unless there are goods produced and men able to produce them. Money is the material shape of the principle that men who wish to deal with one another must deal by trade and give value for value.

—Ayn Rand, *Atlas Shrugged*

A nation trying to tax itself into prosperity is like a man standing in a bucket and trying to lift himself up by the handle.

—Winston Churchill

Nobody spends somebody else's money as wisely as he spends his own.

A major source of objection to a free economy is precisely that it gives people what they want instead of what a particular group thinks they ought to want. Underlying most arguments against the free market is a lack of belief in freedom itself.

Everybody agrees that socialism has been a failure. Everybody agrees that capitalism has been a success...yet everybody is extending socialism!

Spending by government currently amounts to about 45 percent of national income. By that test, government owns 45 percent of the means of production that produce the national income. The U.S. is now 45 percent socialist.

—Milton Friedman

While recovering from an emergency ruptured appendix operation in a Libyan state-run, universal health-care clinic in Tripoli, an exasperated doctor lamented to me that the fellow who was mopping the floor beside the bed by fiat made exactly what he did.

—Victor Davis Hanson

If I were designing a health care system from scratch, I would probably go ahead with a single-payer system.

—Barack Obama

The sad part about today's intelligentsia is that they do not even accept decades-old economic theories like how the free market generates the most free prosperous society. There is a correlation between free software and the free market, as both are powered by innovation created by millions of disconnected individuals. There is nothing wrong with profits, as it generates more competition and innovation. Greed is not bad — it is what gets people out of bed in the morning!

Nothing would improve our world more than having more people familiar with Milton Friedman's work, such as *Free to Choose*. His writings have profound implications for the cradle to grave welfare state we've been steadily enslaving ourselves into. We citizens should read his work, and demand our elected representatives implement the ideas contained therein. If a medical doctor didn't follow scientific advancements of the last 40 years, he would be sued for malpractice, but somehow in government, economic malpractice is allowed! Paul Krugman, who won a Nobel Prize in economics, wrote: "If the rich get richer, the poor and middle-class must get poorer as a matter of arithmetic." It is stunning that he doesn't understand that innovation brings new wealth to a society.

## The Legislature

Knowledge is one of the scarcest of all resources in any economy. Even when leaders have much more knowledge and insight than the average member of the society, they are unlikely to have nearly as much knowledge and insight as exists scattered among millions of people subject to their governance.

—Thomas Sowell, *Basic Economics*

To provide for us in our necessities is not in the power of Government. It would be a vain presumption in statesmen to think they can do it.

—Edmund Burke, 1795

A government big enough to give you everything you want, is strong enough to take everything you have.

—Thomas Jefferson

Of all tyrannies, a tyranny sincerely exercised for the good of its victims may be the most oppressive. It would be better to live under robber barons than under omnipotent moral busy-bodies. The robber baron's cruelty may sometimes sleep, his cupidity may at some point be satiated; but those who torment us for our own good will torment us without end for they do so with the approval of their own conscience.

—C.S. Lewis

Suppose you were an idiot. And suppose you were a member of Congress. But I repeat myself.

—Mark Twain

The government solution to a problem is usually as bad as the problem.

—Milton Friedman

We don't have a trillion-dollar debt because we haven't taxed enough; we have a trillion-dollar debt because we spend too much.

—Ronald Reagan, 1989

No government ever voluntarily reduces itself in size. Government programs, once launched, never disappear. Actually, a government bureau is the nearest thing to eternal life we'll ever see on this Earth.

A young man, 21 years of age, working at an average salary — his Social Security contribution would, in the open market, buy him an insurance policy that would guarantee \$220 dollars a month at age 65. The government *promises* \$127. Now are we so lacking in business sense that we can't put this program on a sound basis? Barry Goldwater thinks we can.

—Ronald Reagan, 1964

Only a crisis, real or perceived, produces real change. When that crisis occurs, the actions that are taken depend on the ideas that are lying around. That, I believe, is our basic function: to develop alternatives to existing policies, to keep them alive and available until the politically impossible becomes politically inevitable.

—Milton Friedman

The problem with America is the the cost of energy, healthcare, education, regulation, litigation, and taxation. All decrease the rate

of progress and push jobs offshore. All of these problems can be fixed, but it is the US Congress who must fix it. They could pass legislation to fix all these problems in a small number of years if the citizens demanded it. Plans for all manner of reform have been sitting around for decades, waiting for the political will. Ronaldus Magnus talked about the need to fix the looming Social Security fiscal collapse in the 1960s, and today we are still merely talking! America is a few votes away from so much progress.

Before legislation can be voted on in the Senate, it needs a 60 vote requirement to end debate. This is a good check and balance because it can allow a minority to keep out a bad government solution, which can be worse than the problem. However, for the long-standing problems that have existed in the United States, it is entrenched interests in the Senate that have stalled progress. A lot of people blame Bush for the problems in the world because his party controlled both houses of Congress for several years, but this ignores the 60 vote requirement necessary to pass any non-trivial legislation.

Unfortunately, lots of what the Congress does involves so many details that get lost on the public, which is why the media play such an important role in educating or even shaping public opinion.

## Education of a free society

Give me four years to teach the children, and the seed I have sown will never be uprooted.

—Vladimir Lenin

I spent four years in the 1990s working at the centrist Brookings Institution and for the Clinton administration and felt right at home ideologically. Yet during much of my two decades in academia, I've been on the "far right" as one who thinks that welfare reform helped the poor, that the United States was right to fight and win the Cold War, and that environmental regulations should be balanced against property rights.

All these views — commonplace in American society and among the political class — are practically verboten in much of academia. At many of the colleges I've taught at or consulted for, a perusal of the speakers list and the required readings in the campus bookstore convinced me that a student could probably go through four years without ever encountering a right-of-center view portrayed in a positive light.

A sociologist I know recalls that his decision to become a registered Republican caused "a sensation" at his university. "It was as if I had become a child molester," he said. He eventually quit academia to join a think tank because "you don't want to be in a department where everyone hates your guts."

Daniel Klein of George Mason University and Charlotta Stern of Stockholm University looked at all the reliable published studies of professors' political and ideological attachments. They found that conservatives and libertarians are outnumbered by liberals and Marxists by roughly two to one in economics, more than five to one in political science, and by 20 to one or more in anthropology and sociology.

I doubt that legions of leftist professors have set out to purge academia of Republican dissenters. I believe that for the most part the biases conservative academics face are subtle, even unintentional. When making hiring decisions and confronted with several good candidates, we college professors, like anyone else, tend to select people like ourselves.

Unfortunately, subtle biases in how conservative students and professors are treated in the classroom and in the job market have very unsubtle effects on the ideological makeup of the professoriate. The resulting lack of intellectual diversity harms academia by limiting the questions academics ask, the phenomena we study, and ultimately the conclusions we reach.

—[Robert Maranto](#), Associate Professor of Political Science, Villanova University

Sometime in the 1960s, Higher education abandoned their role as advocates of American values — critical advocates who tried to advance freedom and equality further than Americans had yet succeeded in doing — and took on the role of adversaries of society.

English departments have been packed by deconstructionists who insist that Shakespeare is no better than rap music, and history departments with multiculturalists who insist that all societies are morally equal except our own, which is morally inferior.

This regnant campus culture helps to explain why Columbia University, which bars ROTC from campus on the ground that the military bars open homosexuals from service, welcomed Iran's president Mahmoud Ahmadinejad, whose government publicly executes homosexuals.

What it doesn't explain is why the rest of society is willing to support such institutions by paying huge tuitions, providing tax exemptions and making generous gifts.

—[Michael Barone](#), American political analyst

I would rather be governed by the first 2,000 names in the Boston telephone directory than by the 2,000 members of the Harvard faculty.

—William F. Buckley Jr., *Rumbles Left and Right*, 1963

Thirty years from now the big university campuses will be relics. Universities won't survive. It's as large a change as when we first got the printed book. Do you realize that the cost of higher education has risen as fast as the cost of health care?

And for the middle-class family, college education for their children is as much of a necessity as is medical care—without it the kids have no future. Such totally uncontrollable expenditures, without any visible improvement in either the content or the quality of education, means that the system is rapidly becoming untenable. Higher education is in deep crisis.

—Peter Drucker

A study demonstrated that 90% of the money donated by Harvard professors in the 2006 election went to the Democrats. Blogger and alumnus Steven M. Warshawsky tells me that there is one recognized conservative professor at Princeton, Robert P. George. I once asked a political science student at a local Seattle university whether there were any Republican professors, and he merely laughed. Higher-ed today focuses on diversity of skin color, but not on diversity of thought. They are factories of liberalism which have infected journalism and government. For the institutions that are supposed to be preparing the next generation, this is a disaster.

## Free Press

Promote then as an object of primary importance, institutions for the general diffusion of knowledge. In proportion as the structure of a government gives force to public opinion, it is essential that public opinion should be enlightened.

—George Washington Farewell Address, 1796

Freedom of the press is not an end in itself but a means to the end of a free society.

—Felix Frankfurter

Journalism naturally draws liberals; we like to change the world.

—*Washington Post* Ombudswoman

The Republican debate provided red meat for conservatives: anti-gay, pro-Jesus, anti-abortion and no gray matter in between.

—Brian Williams, Anchorman of NBC News, 2000

Tonight we have put the best child care system in the world on the American Agenda. That is to say, the system which is acknowledged to be the best outside the home. It's in Sweden. The Swedish system is run and paid for by the Swedish government, something many Americans [such as me] would like to see the U.S. government do as well.

—Peter Jennings, Anchorman of ABC News, 1989

I thought from the outset that Reagan's supply-side theory was just a disaster. I knew of no one who felt it was going to work.

—Tom Brokaw, Anchorman of NBC News, 1983

This housing crisis was completely preventable. The party that blocked any attempt to prevent it was: the Democrat Party. The party that tried to prevent it was: the Republican Party.

I have no doubt that if these facts had pointed to the Republican Party or to John McCain as the guilty parties, you would be treating it as a vast scandal. "Housing-gate," no doubt. Or "Fannie-gate."

Instead, it was Senator Christopher Dodd and Congressman Barney Frank, both Democrats, who denied that there were any problems, who refused Bush administration requests to set up a regulatory agency to watch over Fannie Mae and Freddie Mac, and who were still pushing for these agencies to go even further in promoting sub-prime mortgage loans almost up to the minute they failed.

Yet when Nancy Pelosi accused the Bush administration and Republican deregulation of causing the crisis, you in the press did not hold her to account for her lie. Instead, you criticized Republicans who took offense at this lie and refused to vote for the bailout!

And after Franklin Raines, the CEO of Fannie Mae who made \$90 million while running it into the ground, was fired for his incompetence, one presidential candidate's campaign actually consulted him for advice on housing.

If that presidential candidate had been John McCain, you would have called it a major scandal and we would be getting stories in your paper every day about how incompetent and corrupt he was.

But instead, that candidate was Barack Obama, and so you have buried this story, and when the McCain campaign dared to call Raines an "adviser" to the Obama campaign — because that campaign had sought his advice — you actually let Obama's people get away with accusing McCain of lying, merely because Raines wasn't listed as an *official* adviser to the Obama campaign. You would never tolerate such weaselly nit-picking from a Republican.

—Orson Scott Card, science fiction writer (Democrat)

In a world without truth, freedom loses its foundation.

—Pope John Paul II

In America the President reigns for four years, and Journalism governs forever and ever.

—Oscar Wilde

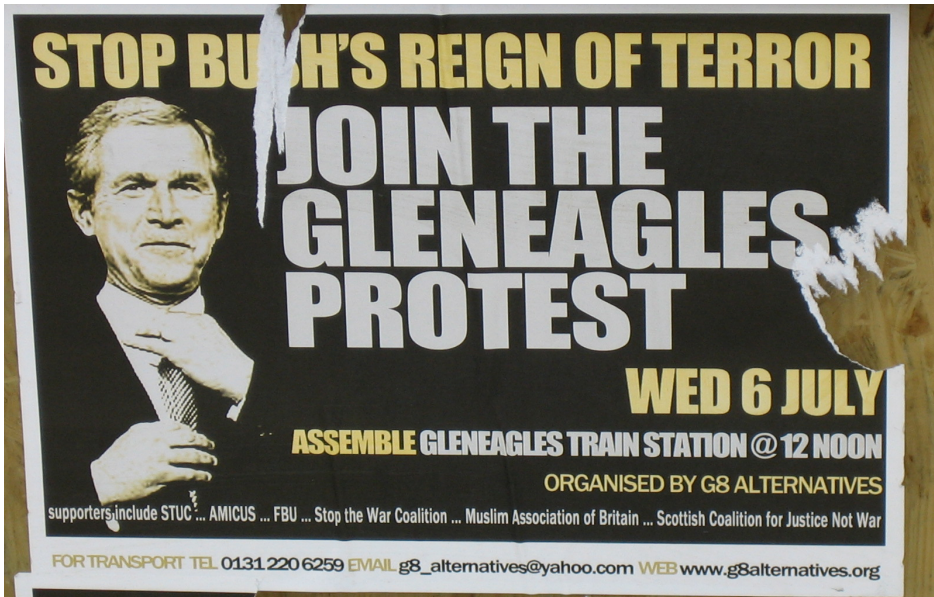


There never was an age of conformity quite like this one, or a camaraderie quite like the Liberals'. Drop a little itching powder in Jimmy Wechsler's bath and before he has scratched himself for the third time, Arthur Schlesinger will have denounced you in a dozen books and speeches, Archibald MacLeish will have written ten heroic cantos about our age of terror, *Harper's* will have published them, and everyone in sight will have been nominated for a Freedom Award.

—William F. Buckley Jr., *National Review*, 1955

The skillful propagandist has the power to mold minds in any direction he chooses, and even the most intelligent and independent people cannot entirely escape their influence if they are long isolated from other sources of information.

—Frederick Hayek, *The Road to Serfdom*



*Poster of a smirking face found on many walls in Edinburgh in 2005.*

A free society requires educated citizens for good governance. The world we live in is very complex, which is why the media have such an important role. Unfortunately, the mainstream media is mostly filled with liberals who push a big-government agenda, and it has been that way for many decades. The press in America are mostly free of government funding and control, but if they advocate for government-provided childcare, higher taxes, and other trappings of socialism, what is the difference? The reason we don't have nuclear power plants in America is not simply because the Congress



has obstructed this progress, but because the media spun Three Mile Island as a catastrophe indelibly etched into the consciousness of the citizens, even though no one even died.

While there are conservatives in the media such as some of the correspondents at Fox News, and *The Wall Street Journal*, and *Forbes* magazine, nearly all the rest, from the big 3 TV networks of ABC, CBS, and NBC, to the cable networks of CNN and MSNBC, to nearly all of the major newspapers from *The Washington Post*, to *The New York Times*, and *The LA Times*, and on and on, lean Democrat.<sup>3</sup> Jack Cashill wrote that *The Kansas City Star*, which serves a center-right community “has created a product, both in its reporting and in its editorial, much better suited to the residents of say, Boston or Seattle, than of Kansas City.”

Not only is the media biased in the US, it appears to be biased around the world as well. There was a recent study commissioned by the BBC which found *itself* to be guilty of a “left-wing bias,” and a “bias of omission.” This study came out 1.5 years ago; a company could reinvent itself in that time, but I wonder if the BBC has made any changes? The bias in the news filters to the outside world: a study found that late-night comics skewer Republicans by a ratio of 7 – 1.

Everyone has personal biases, but if *The New York Times* is an intellectually honest newspaper, why hasn't it endorsed a Republican for President in the last 60 years? If Ronald Reagan was the great president as he is now generally accepted to have been, why did they miss two chances to endorse him? Surely there are countless other things, with hindsight of only 20 years, that they have gotten wrong.

The key to successful propaganda is two things: repetition, and lie by omission. A good example is how the media have reported that President Bush has taken a record number of vacation days:

President Bush recently spent his 879th day at his ranch in Crawford, Texas, breaking former President Reagan's record for taking vacations from the White House.

Note how a vacation day is defined as one not at the White House. However, it leaves out the fact that a President takes the resources of the White House with him wherever he goes, and President Bush has had meetings with foreign leaders, and with his various national security and economic teams at his Texas ranch. The media leave all

---

3 The funniest (if you like sarcasm) and most convincing is Ann Coulter's best-selling *Slander*. She is undeservedly vilified, even by conservatives. She might be “controversial”, and say some unkind things, but doesn't Jon Stewart as well?

these facts out, and just report again and again that they have “the numbers” to prove that Bush is lazy — even though he gets up at 5 am and exercises every day.

It is with repetition and the ability to lie by omission that the media can mold minds in any direction they choose. I have Russian friends from Microsoft who lived in the Soviet Union during the days of *Pravda*, and yet they don't seem to imagine that those same propaganda techniques could and do exist here. I believe the bias of the media is one of the great ongoing scandals of our age.

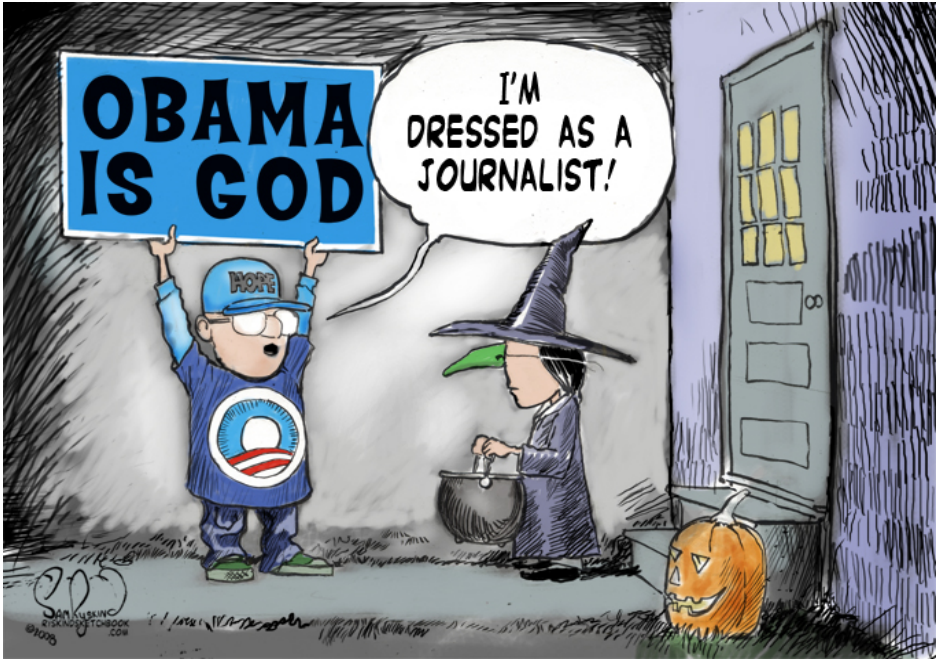
How can the media be leftward in this center-right country? Why doesn't the free market fix this? The answer is that the barriers to entry for a newspaper or TV station are very high. If you live in New York, you read *The New York Times*, and even those who don't like its political bent read it for other reasons such as its Arts section.

Like creating a newspaper, creating a TV network is also very difficult. Even though there is cable news, the big 3 TV networks have 10 times as many viewers as the most popular shows on cable. Furthermore, the major TV networks of CBS, NBC, and ABC have been around for almost 70 years and even if their news business is losing customers, revenue and respect, their many non-news shows can pick up the slack.

And while Fox News has higher ratings than CNN and MSNBC combined, many of the rest of the media have colluded to discredit it as a mere propaganda arm of the Republican party. I've talked to a fair number of fellow software engineers who scoff at the idea of Fox as a legitimate news organization; the simplest way to squelch other viewpoints is to de-legitimize them. Fox News is anchored by Brit Hume, who worked for ABC News for 23 years without getting fired for being a nutcase, but now it is presumed that he is.

The election of Barack Obama is a perfect case study of media bias.

## Barack Obama



*Halloween 2008, by Sam Ryskind*

I start off with the premise that nuclear energy is not optimal. I am not a nuclear energy proponent.

—Barack Obama, 2007

I have not ruled out nuclear.

—Barack Obama, 2007

I am not persuaded that a surge of 20,000 additional troops in Iraq is going to solve the sectarian violence there. In fact I think it will do the reverse. I am going to actively oppose the president's proposal. I think he is wrong, and the American people believe he is wrong.

—Barack Obama, 2007

What I said at the time of the debate of the surge was that when you put 30,000 American troops on the ground of course it's going to have [a positive] impact.

—Barack Obama, 2008

Barack Obama's false and misleading statements on so many matters are so brazen and incessant that it can seem almost futile to try to hold him to account.

—Ed Whelan, *National Review* (conservative)

Media bias in 2008 was the most disgusting failure in our business since the Iraq war. It was extreme bias, extreme pro-Obama coverage. I think it's incumbent upon people in our business to make sure that we're being fair. The daily output was the most disparate of any campaign I've ever covered, by far.

—[Mark Halperin](#), *Time Magazine* (not a right-wing magazine)

The mainstream media were not to blame for John McCain's loss.

—[Washington Post](#) Ombudsman (not a right-wing newspaper)

But Obama deserved tougher scrutiny than he got, especially of his undergraduate years, his start in Chicago and his relationship with Antoin "Tony" Rezko, who was convicted this year of influence-peddling in Chicago.

Another gaping hole in coverage involved Joe Biden, Obama's running mate. When Gov. Sarah Palin was nominated for vice president, reporters were booking the next flight to Alaska. Some readers thought *The Post* went over Palin with a fine-tooth comb and neglected Biden. They are right; it was a serious omission.

—[Washington Post](#) Ombudsman, the previous week

As for the Democrats who sneered and howled that Palin was unprepared to be a vice-presidential nominee — what navel-gazing hypocrisy! What protests were raised in the party or mainstream media when John Edwards, with vastly less political experience than Palin, got John Kerry's nod for veep four years ago?

—[Camille Paglia](#), liberal feminist

During this election, the media in the United States of America was worse than the media in communist Russia. The anchor-men and anchorwomen were reading from the same script. They might have had different haircuts and they might have had different outfits, but they were reading from the same script.

—[Orly Taitz](#), attorney

The swooning frenzy over the choice of Barack Obama as President of the United States must be one of the most absurd waves of self-deception and swirling fantasy ever to sweep through an advanced civilization. At least Mandela-worship — its nearest equivalent — is focused on a man who actually did something. You may buy Obama picture books and Obama calendars and if there isn't yet a children's picture version of his story, there soon will be. Proper books, recording his sordid associates, his cowardly voting record, his astonishingly militant commitment to unrestricted abortion and his blundering trip to Africa, are little-read and hard to find.

—Peter Hitchens, *Daily Mail* (conservative)

Barack Obama is intelligent and has some good ideas, but he won the presidency because journalists fomented anger towards Bush over the last 8 years, and advocated for Obama's victory. The media ignored Obama's education failed efforts at the Annenberg foundation, his liberal voting record, lack of bipartisan accomplishments, his dishonesty, the evidence that Ayers [ghost-wrote](#) Obama's first memoir, and much more. (I am no fan of John McCain!) I'm not arguing that these things are all necessarily true, simply that they weren't discussed.

A study commissioned by the pollster [Zogby](#) of Obama voters found that 86% knew about the scandal that Sarah Palin had spent \$150,000 on clothes, and 94% knew that Palin had a pregnant teenage daughter, but only 40% knew that the Democrats controlled Congress, only 17% knew that Obama won his first election by getting all of his opponents kicked off the ballot, and only 18% knew that Biden quit a previous presidential campaign because of a plagiarism scandal. Millions of people around the world know many "details" about Sarah Palin, but nothing about how Joe Biden was wrong during the Cold War, voted against the first Gulf War, etc. Even though they've only learned half the truth, they don't even notice! All of these examples show how a bias of omission can shape public opinion.

In spite of the fact that I was not an Obama supporter, I believe he could have a great presidency if he tackles some of the long-standing problems in our country. However, as it isn't clear he appreciates the power of the free market, I don't know if it will happen.

## Conclusion

I can easily envision a world where free software has completely taken over, but where *The New York Times*, et al, are still advocating against the policies of a free society. Let's build both!

# HOW TO TRY LINUX

People wanted me to include a Linux CD with the book so that readers could try it out. However, this would add significant costs, makes the book annoying to hold, and free software is evolving so quickly that the CD would become obsolete immediately.

Instead, I suggest you burn a Linux “Live CD” which will run Linux right off the CD-ROM. This gives you a chance to play with Linux and see how it works without touching your existing operating system. I thought it would be hard to make the switch given my many years deep experience with Windows, but within a few months I was using Linux 100% of the time.

Another way to see if you aren't sure if you are ready for Linux, try OpenOffice.org and Firefox, which also run on Windows and the Mac. Once you are using free formats and free applications, you are ready to switch. Demand your hardware and software suppliers support Linux so it will be there when you are ready. Demand free software and it will happen even faster.

If you'd like more detailed information, please visit my website: <http://keithcu.com/SoftwareWars>.

# DEDICATION

Writing books is the closest men ever come to childbearing.

—Norman Mailer, novelist

Art is never finished, only abandoned.

—Leonardo Da Vinci

Edsger Dijkstra, who wrote about the crisis in software in 1972, died in 2002. John McCarthy, who created garbage collection in 1959, is 80. Ronald Reagan, who campaigned on Social Security reform in the 1960s, died in 2004. Milton Friedman, who laid out the principles of a modern free market in the 1960s, died in 2006. Arthur C. Clarke, father of the Space Elevator, died in 2008.

This book is dedicated to the giants who long ago showed us what needed to be done, but who won't be around to see it happen. This book presents an optimistic perspective of man's future if we get off our butts.

## Acknowledgments

I would like to acknowledge my family, friends, teachers, colleagues, reviewers, etc. The total list would be very long and I would likely leave out or misspell names that would mean nothing to my readers, so I won't even try. I am not great about staying in touch, but that doesn't mean I don't cherish our time together!

The cover art was created by Nils Seifert and the cover was designed by Alex Randall.

If you enjoyed a free version of this book, and you want to send me a donation as thanks for my two years of labor, that would be appreciated! You could purchase a paper copy, or go to [keithcu.com](http://keithcu.com) and click on the PayPal button. Any amount donated over \$5 will be given to worthy efforts in free software. If you'd like to contribute money towards a particular area of free software, but don't know how, I can help!

Keith Curtis

[keithcu@gmail.com](mailto:keithcu@gmail.com)

twitter: @keithccurtis





