Open Source in Education

By Timothy Hart

This book is open source. Please feel free to share its contents with anyone. If you would like to contribute by fixing mistakes, adding content, or just have suggesstions or comments, please contact Timothy Hart at tmhart@mac.com.

Version History:

December 2003: 0.6

January 2004: 0.7

February 2004: 1.0

Table of Contents

Schools don't have the luxury of experimenting

Software is better when it's transparent

Proprietary companies are already making their software transparent

Open source threatens intellectual property right

Proprietary software threatens civil rights

Open formats are better

Open source is anti-business, anti-democracy, anti-American, etc.

Open source software isn't reliable

Open source only matters to programmers, since most users never look under the hood anyway.

Part II.

4. What Can Open Source do for Schools?

Saving Money on Software

Lowering Your TCO

The Digital Divide

Open source in the Community

Greater Flexibility

Minimizing Upgrade Costs

Provide Greater Security

To Teach Computing Concepts

5. Open Source Software in Education

Linux

K12LTSP

Apache

GIMP

Tux4Kids

OpenOffice.org

Glossary

Part I

What is Open Source?

Chapter 1 – What is Open Source?

Chapter 2 – History of Open Source

Chapter 3 - FUD

Chapter 1

What Is Open Source?

It is a strange world when educators have to be convinced that sharing information is a good thing. Teachers bestow human knowledge. They do not teach secret information that is only available to those who pay. It seems that the computer industry has led us to believe differently. Open source is changing attitudes towards how we as a society, think about technology. In a world where Copyright has such a viral existence and words like “intellectual property” get thrown onto the most obscure ideas, it can be difficult to see a world products are shared and liberty is given to their use. This world does exist, and the open source movement is in the thick of things.

What is Source?

If you have used a computer before then you have used programs. Programs are software that makes your computer hardware do something. Sometimes programs are called Applications or Commands. Either way, they are pretty much the same thing. Microsoft Word, Apple's iMovie, and the Adobe Photoshop are all examples of programs that allow the user to do something.

Any software that runs on a computer has been created using a programming language. Programming languages are created by people in order to make a computer do different tasks. These languages aren’t much different than a foreign language. Where you learn Spanish to speak to a Spanish person, you learn a programming language to speak (interface) with a computer. And just like foreign languages, there are many different programming languages. They each do things differently but all accomplish the same goal allowing communication between people and computers.

A person communicates through a programming language by writing code. Code is text written on a computer in a particular programming language. So this means that programs are made up of code. Easy enough. We also refer to the code that makes up a program as source code. The following is taken from the American Heritage® Dictionary of the English Language, Fourth Edition.

source (sôrs) n. 1. The point at which something springs into being or from which

it derives or is obtained.

The idea behind using the word source stems from the fact that a program comes from code. Code is at the source of programs. Understanding this allows us to look a little further into how code works and what it looks like.

How does code work?

When an individual is finished writing a program, the next step is to compile. Actually, it is more common to compile during the writing process to see how things are working. This depends heavily on how big a program is. Some programs, like the one shown here, are just a few lines. Others are 6 million lines long. We aren't going to get into that complex of programs. Instead, we are going to look at the basics of programming. These concepts are the same, whatever the size of a program.

As already mentioned, programming languages exist to allow humans to communicate with computers. Through them, computer can understand the directions humans give them. Technically, computers don't actually understand the programming language. I know what you are saying. "But you said that is what programming languages did." I know. It was meant as a simplification of the entire process. If you are confused, keep reading. It will make sense soon.

Computers really only understand one thing. Binary. This is where compiling comes in. When you compile a program, you are translating it from the human readable programming language to computer readable binary. Binary is a string of 1s and 0s. Computers can only understand binary because at their fundamental level, they are just electrical switches that are either on (1) or off (0). These switches are part of a computer's processor. You might have heard of companies like Intel, AMD, and IBM. They make processors that, surprise, process information. More importantly, they process the 1s and 0s created by compiling a program written in a human readable form (a programming language).

At first it might seem like a restriction that you can only have two states (1 = on, 0 = off). Keep in mind that modern processors have upwards of 50 millions of these switches (called transistors) and can make billions of calculations in one second. There is much more to the math and science behind processors, but that is not our goal. If you would like to learn more just consult the Internet or your favorite computer geek.

So a programmer writes code in a programming language and uses a compiler (which is just another program) to translate the code to binary. This makes his application run on the computer and do what he wanted. One might ask why he just didn't write his program in binary. Why take the extra step of compiling? The reason becomes clear when you see how unreadable binary is to humans.

It is tradition to make a "Hello World" program when first learning a programming language. This program just writes the phrase "Hello World" to the screen. This very simple program can give a small idea of how a particular programming language works. In figure 1.1 we can see the phrase "Hello World" in binary.

English	Binary
Hello World	01001000011001010110110001101100011011 11001000000101011101101111011001001100 110001100100

Figure 1.1 – Hello World in English and Binary.

This is not something that the human mind can read. Keep in mind that this is just a translation of the words “Hello World.” It does not include all of the other components a computer requires to make a program work. Imagine writing even a simple program in just 1s and 0s. What if you made a mistake? It would be almost impossible to find and fix.

This should make it clear why we have invented programming languages. We let the computer do all of the hard work so we can accomplish things faster and more efficiently. As mentioned earlier, there are many different programming languages. Some are very complex and others are very easy. Some are designed for certain purposes, while others are more all encompassing. Figure 1.2 shows the traditional Hello World program in some popular languages.

Language	Code	Output

C	#include <stdio.h>
	
	main()
	{
	for(;;)
	{
	printf (“Hello World\n”);
	}
	}
		Hello World

C++	#include <iostream>
	
	int main()
	{
	std::cout << “Hello World\n”;
	return 0;
	}
		Hello World

Java	class helloworld
	{
	  public static void main(String args[ ])
	  {
	  System.out.println(“Hello World”);
	  {
	}
		Hello World

Python	print “Hello World”	Hello World
Perl	#!/usr/local/bin/perl print "Hello World\n";	Hello World

Figure 1.2 – The Hello World program in different languages.

As you can see, the simple task of printing the words “Hello World” to the screen can be done many different ways. Each language has its strengths and weaknesses in different areas. Some tout ease of use, while others claim depth of flexibility. Either way, all are designed to do one thing; take a user’s directions and turn them into a form the computer can use. If you would like to learn more about programming there is a multitude of books on the subject. A great place to start is the book How to Think Like a Computer Scientist by Allen Downey, Jeffery Elkner, and Chris Meyers. It is available online for free and available for a couple of different languages.

So What is ‘Open’ Source?

Now that we know what a program’s source code is, we can look at what the open part is about.

In short, open source software is any computer program with its source code freely available for anyone to see. Open source software also allows a user to obtain, modify, and redistribute the software and its source code for free. This is vastly different from the traditional method of creating software.

Companies like Microsoft create proprietary software. In other words, a user cannot view or modify the source code. If a person wants to use this software, they need a license (read, not own) to use it. People are almost always restricted from making copies or redistributing the software. In fact, doing so is punishable by law with stiff fines or even jail time.

Originally, all software was Open source, even though the name didn’t exist. Computer companies made their profits by supporting the product they made and by selling hardware to run it on. It wasn’t until the late 1970’s that software itself became something to be sold as a product. Companies like Microsoft and Sun started basing their profits on its proprietary software licenses. This began the idea of licensing software and keeping the source code within the company.

Hopefully you are asking, “So, why do I need to see the source code?” or “Don’t they

have the right to keep their intellectual property to themselves?" Instead of answering those questions directly, I will let you come to your own conclusions as you discover more about open source.

There is a popular analogy in describing the difference between open source and traditional methods of software development. In this take on the analogy, all you have to ask yourself is, "welded hood or free engine?"

The Car Analogy

Imagine you are in the market for a new car. You will visit the two dealerships close by to decide which to buy from. You walk on to the first lot to see what they have to offer. This company puts a lot of effort into advertising and letting everyone know that their cars are superior over all others, whether it is true or not. Being a smart consumer, you decide to dig a little deeper to see what their cars are all about. You discover that they sell their cars with the hood welded shut. The dealer is the only one who has access to the engine. Upon further investigation, you also find that you wouldn't even be buying the car. Instead, you would be licensing the privilege to use the car. The company has a detailed license process that restricts the use of their cars and if you break the license, they will hit you with a severe lawsuit. You also find that the car will not even start until you activate your license with the dealer. Being a little disappointed with your discoveries, you decide to go to the other dealership in town, hoping to find a better deal.

The first thing you notice at the second lot is the variety of vehicles is enormous, much more so than the first lot. You soon discover that the hoods are not welded shut, giving you or anyone else the freedom to change anything about it. Even better, the engines this dealer uses are free. You can even take one of their engines and put it in your old car for free, as long as you do it yourself (which is very simple). You could even pay the dealer to put it in for you for only the cost of labor. You also have the choice of buying one of their new cars for a fraction of the price of the other dealer.

Which do you choose, the welded hood or free engine?

Most people will choose the free engine for obvious reasons. People will find the restrictions of the first dealer overbearing and bordering on ridiculous. They will also not be able to turn down the offerings of the second dealer. It almost sounds too good to be true. It isn't. Most people don't know much about cars, but this doesn't mean that they would buy one with it's hood welded shut. This example is meant to get people thinking about what they should expect when buying a product. How many rights should an individual have as a consumer? How much should a consumer be controlled by the company they buy something from? If you apply this story to computer software, does your decision change? Most likely it won't.

In open source culture, software is seen as a tool rather than a way to make money. This isn't to say that you *can't* make money on open source software. That couldn't be further from the truth. There are many companies that make money using open source. For example, Red Hat Inc. sells support for its products and makes millions a year. While some open source projects turn a profit, the main driving force behind the development is to create a really good program. Since the source code is freely available, anyone can look at it and make changes. This allows for collaboration on a grand scale and what is called the many eyes theory. In his essay on the open

source movement, “The Cathedral and the Bazaar,” Eric Raymond wrote, “Given enough eyeballs, all bugs are shallow.” Meaning that when people are allowed to look at the source code of a program, bugs will be found and fixed very fast.

Probably the most popular reasons to use open source software is the ability to see the source code of the program you are using. You may freely change the source code to your needs (some restrictions may apply depending on what license it is under). Assuming you have the skill to do so, you can customize the source code to do anything you want. At first glance, this means very little to the non-programming populous. You might ask, “I have no idea how to program, so what is the point?” Going back to the car analogy, “Would you buy a car with the hood welded shut just because you don’t know how to change the oil?” Just because one does not have the skills to program, does not mean that they should settle for a closed-source product.

Chapter 2

History of Open Source

Where did open source come from? This is the next important question on our road to understanding open source. We will be looking back at the dawn of computing. At a time where it was unheard of for people to own computers. When computers were big-clunky-noisy things sitting in the basements of Universities and government buildings.

When we understand where open source came from, we have a better understanding of why it exists and more importantly, what it can do for us. Also, understanding the past allows us to see where the movement is headed. This will help us make better decisions about the future and how we use technology in general.

Like any account of the past, there are holes in the descriptions which follow. There are many stories that have not been included that had a part in making open source what it is today. The goal here though is to give an overview of the biggies. The people and events that had the most impact.

The Beginning

During the 1940s, computers began helping man do all sorts of things. Back then, the computer was used for scientific problem solving. It was especially good at making mathematical calculations very fast. Government uses of computers ranged from figuring out the trajectories for missiles to making the census faster. Scientists were better able to collect and study data, which took science in new directions. The layman did not use them because they had no reason to. The Internet, email, and mp3s didn't exist. Computers had no practical everyday use for the non-scientist/engineer.

These scientists and engineers had the skill to create their own programs for the task or research they were working on. There weren't any software companies to get programs from. Software was seen in an academic light rather than as a commercial product. In academia, sharing of ideas and work was and is encouraged. It is generally understood that this is how innovation occurs. The people using computers at this time had no reason to horde their software. Sharing was just how things were done.

When the 1950s rolled around, computers were starting to be used for more than just scientific problem solving. Businesses saw potential in the power of computers to do math. They

discovered that computers could make their companies operate hundreds of times more efficiently. Transactions were faster and more accurate, low-level tasks were automatized, and a view of a company's immediate status was possible.

One would think that businesses would be less likely to share their code than academics. Why would a company that develops some code, share it and potentially help the competition? The answer lies with the limitation of computers of the day. They were hard to program, had poor input/output (think keyboards and monitors) peripherals, and had a very small amount of memory.

These problems made a program that worked well (which was an accomplishment), very useful. Business A would share it's program with business B because it could receive the same in return. This improved efficiency for everyone and was generally seen as the way of doing business. The benefits gained from sharing with competitors greatly outweighed the desire to horde code.

The use of computers in business surpassed use in science in the 1960s. This started an explosion in development of new technologies that made computers faster, more powerful, and more useful.

Though rise in use had its growing pains. Computers at this time used many different designs. One may have used processor A, while another used processor B. Each had its own programs and operating systems. When a new computer was bought, all the software that was used on the old machine had to be rewritten. This was a major headache to all who wanted to upgrade. This process wasted time and money on coding and training on the new machine. These inefficiencies hindered technology from moving forward because of the costs associated.

UNIX

The answer to this problem came from New Jersey. AT&T's Bell Labs had a group of engineers that was going to solve this interoperability problem. They wanted to create an operating system that would be portable between multiple hardware architectures. Basically this means the operating system could be easily reworked to run on different types of computers. If such a product were possible, it would require no reengineering of software and no substantial retraining of workers when a new computer was purchased.

Most of this work at AT&T Bell Labs was done by Dennis Ritchie and Ken Thompson. One story of Thompson's interest in the project involves his hobby of video games. He wanted to play his favorite computer game, Space Travel, on a PDP-7 computer. This was not possible at the time so he is said to have started work on an entire operating system to allow him to do so. Sort of the straw that broke the camel's back.

In 1969, the Unix operating system was born. It fixed the interoperability issue that plagued computer users of the day. It allowed basically the same operating system to be run on different types of computers. This meant that the software applications were easily moved to another hardware platform.

Unix was never meant to be a commercial operating system. At first it was only used internally within AT&T. This had some notable effects on how Unix worked. First, it was never meant to be pretty. It was designed to get the job done. Second, it was designed to be portable, practical, and powerful. It achieved these things with flying colors. A fact that can be proved by the

fact that it is still around, 30 years later.

Soon, others wanted to run this operating system and have all of the benefits. Traditional economic thought tells us that when a company develops a product, the product is then sold. AT&T did not do this. There are two factors contributing this decision.

The first was a US government consent decree. Pre-1956, AT&T functioned as a legally sanctioned, regulated monopoly. It was the only company controlling the national telephone system in the United States. In the early 1900s popular sentiment held that a monopoly was needed in order for a national telephone system to work. By middle of the 1900s political philosophy had changed and there were many complaints against AT&T for abusing its grasp on the market. This led to an antitrust suit filed against the company in 1949, leading to the 1956 consent decree. This decree restricted AT&T to doing business with the national telephone system and government work. More importantly for our purposes here, the decree forbade AT&T from entering the market of computing. Selling Unix would definitely be considered entering the computer market. AT&T decided that Unix could be put to good use by many people, so it gave it away. With the payment of a small licensing fee (not like those common today), one could receive the Unix operating system and, more importantly, the Unix source code.

The second reason for giving away Unix was the ideology of the day. Companies like Microsoft or Adobe did not exist. Selling software was not seen as being something that was done, let alone something on which to make a profit. The money to be made in the computing industry was through selling hardware. In turn, this meant that software was traded from user to user for free or for a nominal fee. This allowed users to fix problems with the software and add new features. It also made the software more reliable and more powerful. It is also the roots of the modern free software and open source movements.

Since 1969, Unix has gone through some changes but remains one of the most powerful and flexible operating systems around. With the availability of the source code, many universities and corporations made their own customized versions of Unix. These different versions, called distributions, included different tools and programs that fit the users needs. This remolding gained in popularity and led to the hundreds of different distributions of Unix we have today. Though not all are as free as the original.

As the number of Unix distributions grew, the availability of the source code waned. The computer market grew throughout the 1970s, and the use of commercial licenses grew with it. Regular people started using computers and the personal computer was born. Thus began the snowball rolling into the avalanche of technology we have today. It also began the practice of charging for software. Why sell only hardware when you can sell software as well?

As computers became less expensive and more accessible, non-technical office workers and home users wanted them. In targeting these growing markets, software companies built their business models on control of their source code. Most users don't want or need access to source code, and executives and investors wanted their companies to control key assets (e.g. intellectual property). Companies sold the compiled binary as proprietary software, and kept the source code secret. In other words, they sold the product and kept the process.

To some this was an outrage. The most notable person in this camp would be Richard Stallman.

RMS, GNU, and the Free Software Foundation

Arguably one of the most (if not the most) influential people in open source is Richard Stallman. He has made uncountable contributions to computing including many free programs and the legal backbone of free software.

RMS, as he calls himself, was a hacker for the Artificial Intelligence Laboratory at MIT in the 1970s. Keep in mind he worked as a *hacke*r, not a *cracker*. A cracker is one who tries to violate the security of a system, whereas a hacker is one who enjoys doing clever things with computers. His time at the AI lab provided the roots of his beliefs on computing. This time was also when the computing industry was starting to move to closed source proprietary programs. RMS had encountered this trend and was quite unhappy about it.

One such encounter occured with the lab's move to a new computer system. In the seventies, the hackers at MIT had created an operating system called ITS: Incompatible Timesharing System. Its source code was available to anyone who asked for it. This allowed for new features to be added and bugs to be fixed easily. One of the only drawback of this system was that it was built specifically for the PDP-10 architecture. This meant that the ITS operating system would have to be rewritten to work on another computer. Eventually, the time to upgrade came. With the purchase of the new computer, MIT decided not to move ITS to it. Instead, they bought the commercial operating system which came with the machine.

This new operating system was closed-source. Stallman was unhappy with it but was forced to use it. He could not view the source code. He could not see what was going on in the guts of the system or make any changes or repairs to the system. Though he did have the chance to pay for an expensive license to view the source code, he did not do this because he would not be allowed to share his changes with others.

Another event that contributed to Stallman's strong idealism was a dealing with Xerox. In 1979, the AI lab received the first laser printer from Xerox. On occasion, the printer would jam and require human intervention to fix. This wasn't hard to do but the user sitting at their computer had no way of knowing the printer was jammed unless they walked to the location of the printer. This began to annoy users, as they would have to make many trips back and forth to get things printed. Stallman, being the hacker he was, thought he would fix the problem. He wanted to add code that would send a message to a user when the printer jammed to the printer driver software.

When Stallman asked for the source code from Xerox, he was denied. The story goes that Stallman went to Xerox headquarters personally to confront a real person and was rudely sent away. Xerox was a big company and had made the move to keeping their products secret to make bigger profits.

These and other events laid the foundation for Stallman's belief that software should be free. "Free as in freedom, not as in free beer," Stallman once said in regarding which definition of the word free should be used. Free software has to do with the liberty of having the right to what you want with it. The word free was not meant to denote lack of cost, but instead the lack of restriction. In other words, software that could be shared without restriction. Software that could be modified by anyone and those modifications would be available to all.

Eventually, RMS' principles grew stronger and he resigned from MIT to use his time to create free software. He eventually founded the Free Software Foundation (FSF) in 1985, which

would become the main avenue to creating this unrestricted software. The organization would produce free software and promote its use.

With the creation of the FSF, Stallman also wrote the GNU (pronounced “guh-noo”) manifesto. In it, he laid out the beliefs of the FSF and coined the term “Free Software.” This manifesto formalized a process that had been, up to this point, been going on in an ad hoc fashion.

The thought of creating free software was definitely a good idea. The only problem was that it wasn’t enforceable. Basically this software would be released in the public domain, but there was nothing stopping people from taking it and hording the enhancements they made. Thus changing this free software into proprietary software, defeating the whole purpose behind it. Out of this problem grew a project that would create software that was truly free. Software that would be perpetually free. This project would be called GNU.

GNU is a recursive acronym standing for “GNU’s Not Unix.” It is common to find many of these play on words in the Unix and open source community. The purpose of the GNU project was to create a Unix-like operating system that embodied Stallman’s ideals. While the GNU operating system was delayed for quite a while, the FSF came out with many products that were very important in the development of open source software. The GNU C compiler was used by many people to, as the name implies, compile code written in the C programming language. Many programs were written in C at this time and this tool was invaluable to getting the free software movement started. Another notable piece of software is the Swiss Army Knife of text editors called Emacs. These products, which are still widely used today, are overshadowed by what is arguably Stallman’s greatest contribution to the world, the GPL.

Once Stallman released his software, he was faced with another problem. As mentioned above, by creating free software there was nothing preventing someone from incorporating it into a proprietary project. At first, Stallman thought that doing nothing was the best way to go. It was only after free projects, like the X Window system, was incorporated into closed-source projects, that Stallman saw the need for a system to protect free software.

The answer to this was copyleft. This term was coined by Stallman’s friend, Don Hopkins. The nature of copyleft can be clearly typified by the recasting of the common copyright phrase, *copyright – all rights reserved*, to *copyleft – all rights reversed*.

> The central idea of copyleft is that we give everyone permission to run the program, copy the program, modify the program, and distribute modified versions—but not permission to add restrictions of their own. Thus, the crucial freedoms that define “free software” are guaranteed to everyone who has a copy; they become inalienable rights.

The specific implementation of the idea of copyleft by the Free Software Foundation, was the General Public License (or GPL). The GPL is a license that can be used for any software that one creates to guarantee that it remains free. It has become the legal backbone for free software and open source software.

Stallman’s own commentary on the license is contained in his Free Software Definition, a

document that describes the qualities of free software. He summarizes what the GPL is meant to do as follows:

> Free software is a matter of the users' freedom to run, copy, distribute, study, change, and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software:
>
> - The freedom to run the program, for any purpose (freedom 0).
> - The freedom to study how the program works and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
> - The freedom to redistribute copies so you can help your neighbor (freedom 2).
> - The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. (freedom 3). Access to the source code is a precondition for this.
>
> A program is free software if users have all of these freedoms.

With the freedoms the GPL provides, one can make sure to get the credit they deserve for creating a piece of software and letting others use it for free without the threat of it becoming closed-source.

Another notable point is that the emphasis was never on price. Selling copies of a computer program was and is both allowed and encouraged by the GPL. When buying something licensed under the GPL, you are buying the convenience of providing easily available code in an easily accessible form, say on CD. This differs from buying a closed-source program by the terms in which they are bought. When you buy a Microsoft product for example, you are buying some secret "intellectual property" found in the bits on a CD. The value in a closed-source product lies in how much the company believes people will pay for their intellectual property. Even though one can charge for open source software or free software, it is also almost always available for download from the Internet.

The GPL remains one of the most widely used licenses for free and open source software. It is part copyright contract and part philosophical statement. As of this writing, the GPL has not been challenged in court. When someone has been found in transgress of the GPL, the FSF retains legal counsel and have often had part in negotiations to correct the situation. The document has gained a moral force due to the respect with which is has been treated since its invention.

It is clear that Richard Stallman has made innumerable contributions to the computing world. His work and almost evangelistic efforts to promote free software have help bring open source to the main stream. He is often called the chief apostle of the free-software movement and one of the fore fathers of the open source movement.

By the late 1980s, the Free Software Foundation had created many free software projects. However, something was missing - a free operating system. The GNU project was making progress towards achieving this goal but many things were lacking. One of the biggest missing pieces was a kernel; the core of an operating system that makes all other components to work together.

One of the reasons for this trouble was the design of the kernel. It was designed in many small pieces that all specialized in a specific task. This was to create pieces of the kernel which did its job very well. The problem, and cause of severe delay, was that making these separate parts work together proved to be very difficult.

Another reason for the slowing of the project was the logistics used at the Free Software Foundation for the development of software. Even though people could obtain the source code for a GNU program and even change it, they had no say in the development of the programs themselves. The main coding on the projects was done by the small team at the FSF.

There was also a nontechnical hurdle for the GNU operating system that would have most likely limited its use. GNU software was mainly used in academic circles. There were far more computer users in the business sector and their general view of the people at the FSF was not positive. Most corporations saw the GNU project as one created by a bunch of weird idealists. The GNU project and the FSF's philosophy was mainly ignored by businesses. Due to Stallman's harsh words for businesses using closed source software and his refusal to compromise.

When 1990 came around, a GNU operating system seemed like it was never going to be released. Two things were needed: a working kernel and corporate legitimacy. The answer to both of these problems came from an unlikely place; a young college student from Finland.

Linux

Twenty-one year old Linus Torvalds, a student at the University of Helsinki, began a project that would turn out to become the poster child of the open source movement. In 1991, Linus bought his first 386 computer (what would later become what we call a PC today) and was disappointed with the operating systems available to him. He didn't like MS-DOS and the available commercial Unix systems were beyond his financial means as a student. He eventually settled on a Unix-like OS called Minix.

Andrew Tanenbaum, a professor in the Netherlands, designed Minix primarily as a tool to teach his students about operating systems. The source code was freely available but changes required Tanenbaum's permission. Torvalds had been working on adapting Minix to his own personal needs. There were also many features that Torvalds wanted that Minix did not provide. For example, he wanted to improve its terminal emulation program, by which he connected to his university's server from home. Since Torvalds forte was programming, he found himself creating an entirely new operating system to meet his needs.

In his autobiography, *Just for Fun*, Torvalds says he used Minix as inspiration. As his operating system eventually grew in complexity and functionality. One day, Torvalds accidentally deleted Minix from his computer and triggered the realization that, instead of reinstalling Minix, his creation was ready for the chance to stand on its own.

It is also important to point out that Linus started his project for fun. He had no plans of

making money off of it. He just wanted a Unix-like operating system that was both powerful and fun to work on. This emphasis on fun is key to understanding why people use and develop open source software.

The name Linus originally chose for his project was Freax (pronounced “freaks”). This was a mix of the terms “free,” “freak,” and an “x” at the end. The idea behind the name was Linus’ project was a *free* operating system for computer *freaks* with the usual *x* ending used on Unix variants.

After a while, Linus announced his project on a Usenet newsgroup. Ari Lemke, at the Helsinki University of Technology, was one of the first to read his message. He offered Linus a sub-directory on the university’s ftp site for the system when it became available. It was also around this time that Lemke convinced Torvalds the name Freax was lacking. Instead, Lemke though Torvlads’ private name for the OS was better. Linux. Being a mix of “Linus” and “Unix.” Linus thought this to be too egotistical, but soon agreed.

It wasn’t long after Linus posted his code online that others began to become interested in the project. People with similar desires began looking at the code and started submitting patches and suggesting changes. Linus took these ideas and put them into Linux and gave credit where it was due. This was the start of one of the biggest collaborative projects in history.

The decision by Torvalds to release his code under the GPL, helped his project gain momentum. More than a hundred people joined the effort. Then thousands. Then hundreds of thousands. Soon Linux became a force to be reckoned with and it was free.

In 1994, Linux reached version 1.0 and the number of users world wide was estimated to be around one million. Most of these users were programmers and students but soon commercial vendors became interested. Since Linux was free, these vendors would compile the Linux kernel with other various software and distribute their version. So born was the Linux distribution (or distro or flavor). Companies like Red Hat, Debian, and Caldera gained users worldwide and brought Linux into the mainstream.

Today Linux has gained a cult following and Linus has become a celebrity. Linux has been shown to be a fantastic program that is faster, more powerful, and scales larger than almost any other operating system. For these reasons, businesses have embraced Linux and used it in many different ways. Linux has proven to be a strong contender for the backend (servers) and is currently making the push to the desktop.

The Internet

The scale and reliability of the Internet has largely been possible through the use of open source software. No single company or organization could manage such a large network, so everyone agreed on certain rules (i.e. open formats and protocols). These rules dictate how desktop computers talk to servers and how servers talk to each other.

One such rule is the Internet Protocol (IP) address. Every public server has an IP address. People prefer to enter text addresses (e.g. www.yahoo.com, oss.survey@nwrel.org) but computers need IP addresses (e.g. 216.32.74.51) to find each other. The translation software is open source and everyone uses the same address list. This way anyone on the Internet can find anyone else.

Various software companies offer proprietary Internet solutions. These solutions are built on or around open source and are compatible with similar proprietary and open source software. This prevents a single company from controlling the Internet and keeps the commons open.

The rules change as new features or services are delivered through the Internet. The open source model has excelled at adapting old solutions and creating new ones. As companies try new strategies like online retailing, they depend on Internet software. They often choose open source software like Apache.

The Internet is not completely dependent on open source. Proprietary programs like Adobe Acrobat have become reliable solutions. However, without open source it is doubtful that the Internet would have become as large, diverse, and reliable as quickly.

The open source movement is an Internet phenomenon. While plenty of programmers and proponents work and socialize offline, the driving force behind open source is computer networking. Foremost, open source software is usually downloaded rather than purchased. Patches and other improvements are just as accessible. When users need solutions, they can often get the software immediately.

In addition to distributing and supporting their software, most open source projects depend on the Internet to coordinate and collaborate. Programmers exchange source code almost instantaneously. They use email lists and Web sites for planning, debate, feedback, and publicity. For example, thousands of projects use the SourceForge Web site to coordinate work. Many educators use the K12 Open Source Now mailing list to support one another and discuss issues specific to schools. Publicity is especially important because open source projects depend on volunteers. Projects use community sites like Slashdot.org to invite participation and celebrate project milestones.

The Internet serves the entire open source community, not just the programmers. Email lists and Web sites allow users to promote solutions, give feedback, and help each other learn and troubleshoot. Many educators enjoy and value this responsive, helpful community.

The Cathedral and the Bazaar

As the Internet exploded into the mainstream, open source projects like sendmail, bind, and Apache grew in popularity. Today, Apache is the dominant Web server solution, and it runs under Linux (as well as Microsoft Windows and Apple Mac OS). For this and other reasons, the rapid growth of the Internet and the rising popularity of open source software are interdependent.

The Internet and "e-business" demanded new, cutting-edge solutions. Linux, Apache, and other open source software could outperform costly proprietary UNIX or Microsoft solutions. This success in the "backend" puzzled people. How could a community-created product be more powerful and reliable than proprietary solutions?

To help explain this phenomenon, programmer Eric Raymond wrote an essay called “The Cathedral & the Bazaar.” In Raymond's so-called cathedral, a leader sets the goals, coerces programmers to participate, uses extrinsic rewards (e.g. money, promotions), and controls the product and its secrets. In the bazaar, a leader shares a vision, invites programmers, rewards their contributions with fame and gratitude, and shares the product as open source. Raymond emphasizes the fame reward, using the theory of gift/reputation culture to explain programmers'

eagerness to work hard on code and then give it away.

Raymond's bazaar was clearly inspired by Stallman's copyleft: elevating the principles of community over selfishness. But where copyleft prevents a company from using community-developed resources in a proprietary product, Raymond didn't foreclose on that possibility.

Raymond's bazaar helps explain why a programmer would work hard improving software, then give it away for free. Of course, most educators are generous with their work and ideas, so the bazaar may be less puzzling to them than to companies.

Netscape

In the mid 1990s, a war was brewing. The Internet and the World Wide Web were taking the world by storm and two companies were fighting to come out on top. Netscape and Microsoft were battling for market share. The companies sold competing, proprietary Web browsers: Navigator and Explorer. Both also offered Web editing and hosting solutions, and tried to control the evolution of HTML with secret formats and protocols. (Microsoft calls these "extensions.") For a time, there were Explorer and Netscape versions of the same Web pages. This can still be found today on many sites. It is not uncommon to see a website stating “Page best viewed in (insert browser here).”

Explorer held a significant but minority market share. So Microsoft changed their business model to focus on Windows, Frontpage, Web server software, and Web hosting. Microsoft bundled (and later integrated) Explorer with its Windows operating system, and offered it as a free download. Netscape did not have an operating system or office suite products to fall back on, and it would be years before the Justice Department finished investigating Microsoft's alleged anti-trust behavior. To prevent Microsoft from controlling the browser market (and HTML), Netscape chose a surprise strategy: they "open sourced" Navigator. They mentioned Raymond's bazaar as an influence.

Netscape planned to create an open source browser called Mozilla. Soon after Netscape announced the Mozilla project, America Online (AOL) bought Netscape. As an Internet service provider, AOL competed with Microsoft's MSN Internet service. In light of that competition, Raymond wrote an open letter to AOL:

> The Mozilla project proceeded from Netscape's understanding that Internet Explorer threatened their server business. If Microsoft were able to establish a monopoly lock on the browser market, they could leverage that into effective control of the HTTP/HTML protocol; from there it would be a short step to effectively crippling any competitor's web servers. (Raymond, 1998)

AOL continued to support the Mozilla project. Today, Mozilla is still going strong and has contributed much to the open source community. Even though the majority of users still use Explorer, the open sourcing of Navigator played a big part in bringing the open source model to the attention of the world at large. This legitimized the open source movement for many businesses and people started to take it seriously.

Microsoft

You will not usually find a section on Microsoft in histories of open source. The reason being that Microsoft is vehemently opposed to open source software. They are sometimes the most vocal in the debate between the open source and proprietary models. Microsoft is widely perceived as a driving force behind organizations like the Business Software Alliance (BSA) and the Initiative for Software Choice (see http://www.bsa.org/usa/ and http://softwarechoice.org/). The BSA is investigates license noncompliance and prosecutes software piracy, and thus represents the "strong arm tactics" the open source movement reviles. The initiative lobbies in the USA and internationally to discourage governments from legislating in favor of open source software.

That being said, you might have heard that Microsoft is disliked by some. You may have heard of antitrust lawsuits. Maybe of shoddy software. Maybe even talk of Microsoft being greedy, dominating, controlling or a number of other things that won't be mentioned here. These are points that many bring up when discussing Microsoft, but there is also the other point of view.

Many believe that Microsoft is a picture of the American dream. A guy came up with a product and brought computing to the masses. These people believe that Microsoft is so successful because it makes the best product.

People get in arguments about this everyday. If you want to see a fight, just ask your local geek what his opinion on Microsoft is and disagree with them when they tell you.

Proponents of open source usually take the first ideology. One reason being that Microsoft is not, putting it lightly, popular with the open source movement. The ideas behind open source and Microsoft are usually polar opposites of each other. The information presented below shows a company that many do not know about. This brief history of Microsoft is from the point of view of proponents of open source. It shows how the company has affected computing, which leads to open source.

.....

Founded by Bill Gates and Paul Allen in 1975, Microsoft has gone on to be the largest private sector computer software producer in the world. Its founders are among the richest people in the world. In a capitalist society this is seen as a good thing. But when we look a little closer and see how Microsoft got to (and remains at) the top, the picture isn't pretty.

Gates and Allen were high school buddies and had both been interested in computers. They had talked about starting a company around software, Gates was not convinced and left for college at Harvard. During this year (1974), the Altair came out. The Altair is generally seen to be the first personal computer. Allen was so excited when he saw this computer on the cover of Popular Electronics that he immediately contacted Gates about the opportunity this created for them. The next year, Microsoft was founded. The rest, we say, is history.

From the start, the purpose for Microsoft was to make money. This differs from preceding software that was freely shared for the betterment of computing. It did not take long after the beginning of Microsoft for the company to get on the bad side of supporters of freely available software.

Gates attended a Homebrew Computer Club meeting in late 1975 and discovered that people were sharing his software. Microsoft's software. Instead of seeing this a natural occurrence,

he thought it was stealing. So in January of 1976, he wrote an “Open Letter to Hobbyists,” which was published, among other places, in the Homebrew Computer Club Newsletter. In the letter, Gates called all who shared software, thieves. He also tried to explain the costs involved in writing a major software work.

Gates rant letter turned out to be very unpopular. It set off a huge controversy among computer users. Most retorts were against it, although a few people who were paid for writing software backed Gates. This was the first, in a long line of events that led to some users distaste for Microsoft.

One would think that since Microsoft is a market leader (at least monetarily), they would have contributed a lot to computing. A company that has been around for twenty-eight years must have done a lot of innovation, right? Microsoft has said, especially in its defense in the antitrust lawsuits, that it has innovated many technologies and contributed much to the computing world. Let’s look at some of the “innovations” that Microsoft has given us and see how much this huge company innovates.

Innovation 1, MS-DOS. When IBM made a deal with Microsoft to buy an operating system from them, Gates had no operating system to sell. So they bought one called QDOS from Tim Patterson that he wrote 6 weeks before. Microsoft renamed it to MS-DOS and sold it as its own. They would then go on to add features that had been part of the Unix OS for some time.

Innovation 2, Graphical User Interface (GUI). The idea of using windows, icons, and a mouse to operate a computer was nothing new. Apple had brought this to the masses in the early 80’s with the Macintosh. Microsoft’s only innovation here was changing the Macintosh trash can to a recycle bin. How environmentally responsible. The Graphical User Interface was really developed at Xerox Parc in the 1970s. Also until the release of XP/2000, Windows was built on MS-DOS. Windows was just a program that ran on top on MS-DOS that provided a graphical user interface. This is generally what led to Windows poor stability, which led to the popular use of control-alt-delete.

Innovation 3, Office. Microsoft Office definitely has wide spread use. It is no innovator though. Word was written in 1983. Lexitron and Linolex developed the first screen-oriented word processor in 1972 and WordStar improved the word processor in 1979. MS took these ideas and put them into Word. Excel is another member of the Office product. VisiCalc and Lotus 1-2-3 existed long before Microsoft made an attempt at this type of program.

Innovation 4, Internet Explorer. Microsoft was not an innovator when the Internet was first getting to the masses. They were well behind in 1995 when they actually saw the World Wide Web as a thing that was going to be around for good. Netscape Navigator had been around for a while by the time IE came out. Internet Explorer was not originally developed at Microsoft either. They licensed code from Spyglass, Inc and called it Internet Explorer. Microsoft proceeded to distribute IE for free and denied Spyglass substantial royalties for their contributions to the product.

Microsoft has also been notorious for including web features that only work in Internet Explorer. You have most likely seen “Best viewed in IE,” on a web page before. These extensions are only to force people to use IE and therefore force providers to use MS products as well.

Innovation 5, Multitasking. Microsoft has said many times that its Windows operating system has brought the ability to run more than one program at a time (called multitasking) to computing. In actuality, Unix did this in 1969 and was the first to achieve this. Many Unix people will tell you

that the current implementation in Windows isn't even as advanced as Unix was back in 1969.

Innovation 6, PowerPoint. Now part of the MS Office suite, PowerPoint is a presentation software package that was renamed and rebranded after Microsoft acquired Forthought, Inc., in 1987.

Innovation 7, FrontPage. The popular WYSIWYG (What You See Is What You Get) HTML editor was purchased from Vermeer Technologies in 1996.

These are just a few of the so-called "innovations" from Microsoft. This is one of many reasons that the open source community generally views Microsoft with distaste. They have made no notable innovation other than to buy out companies and sell its products. This is only part of the reason for peoples distaste for anything Microsoft though. Open source proponents see the actions of Microsoft as anticompetitive and generally stifling others ability to innovate.

The Open Source Initiative

In 1998, Eric Raymond, Bruce Perens, and other proponents of open source launched a broad strategy to legitimize open source. Where Stallman and the Free Software Foundation had argued from principles like user rights, the new Open Source Initiative (OSI) would tout better code and untapped profits. Raymond later wrote:

> It seemed clear... that the term "free software" had done our movement tremendous damage... Most of it came from... the strong association of the term "free software" with hostility to intellectual property rights, communism, and other ideas hardly likely to endear themselves to an MIS manager. ... Our success... would depend on replacing the negative FSF stereotypes with positive stereotypes of our own -- pragmatic tales, sweet to managers' and investors' ears, of higher reliability and lower costs and better features. (Raymond, in Dibona p. 212)

The Open Source Initiative is a non-profit corporation that manages and promotes the Open Source Definition (OSD). The OSD defines what constitutes as open source software. The OSI also does a lot of advocacy for open source and through its certification mark and program, provides users with confidence about what is open source software.

Free Software versus Open Source

Even though it sounds trivial, free software and open source software are not the same thing. The FSF used (and still does) the word free, meaning freedom. Their main focus is on the moral and ethical reasons behind creating "free" software. On the other side, Linux developers were only interested in the benefits of creating a free operating system. Even though the goals were alike, the philosophies were different and a new term was needed. The term open source was coined in 1998.

This helped the open source community continue to grow. The "free" in free software had become a problem. The media and others were always confusing the word free with the monetary

definition. While lower cost is a byproduct of this type of development, it is not the main reason behind open source. The word “free” scared executives who thought these people were crazy for giving their programs away. When “open source” was used, it became easier to teach people why it exists and how it was better than developing a closed-source product. According to Russell Pavlicek in his book *Embracing Insanity*, there are four key benefits from the term open source.

1. Open source emphasizes quality.
2. Open source stresses flexibility.
3. Open source decreases development time.
4. Closed source is characterized as non-competitive, rather than immoral.

Today

Clearly, open source software is reaching the mainstream. The debate is stronger than ever and the ubiquity of the Internet and publicity of key open source projects has changed the scope. Microsoft and its business allies continue to disparage open source, creating a vivid conflict in marketing, the media, and mindshare. Apple builds their OS X on Darwin, a derivation of open source BSD. IBM, Sun, Cisco, Hewlett-Packard, Dell, Oracle, and many more are investing in open source. National governments are considering policies and legislation promoting open source.

Like a silent storm, open source runs much of the Internet and the Web (especially Web servers and gateways). It's making inroads on servers, desktops, mobile devices, everywhere. While Stallman, Torvalds, Raymond, and others are strongly influential in the future of open source, it has become a global phenomenon with an definite place in the future of computing.

Chapter 3
FUD

FUD. A term that people within the open source community are all too familiar with. Pronounced fuhd, it stands for Fear, Uncertainty, and Doubt. In one of its definitions, FUD is basically a marketing technique used by a company to scare users away from the competition. The company will rely on scare-mongering and try to release rumors that cast a shadow of doubt over the competitors offerings and make people think twice before they would use it.

Many people point to Gene Amdahl as the originator of the term and concept. In the 1970s, he left computer giant IBM to start his own company. He then became a target for what he called FUD.

> "FUD is the fear, uncertainty, and doubt that IBM sales people instill in the minds of potential customers who might be considering [Amdahl] products."

The idea becomes to confuse would be customers about the competitors product to make them feel better about buying the FUD pushing companies product. This tactic has been used heavily by companies against open source software. While much FUD has been debunked by the hard evidence of the success of the movement, these falsehoods are still prevalent. Most of the time these misguided comments are spoken by people who do not know what they are talking about and just pass on what they have heard before. Even though the information this person may be putting forth may be false, it never the less creates a situation where people may believe it.

Time has proven a good thing for open source FUD. Many early misunderstandings about open source have been so thoroughly refuted that they are now seen as ridiculous by the community. This doesn't mean that the FUD doesn't have an ill effect. People not in the know may still believe such statements. This is common in education because there is often not a person who fully understands the technology that their schools use.

FUD also goes both ways. There are many misconceptions surrounding the open source movement that may have a certain amount in falsity. This may lead decision makers to use open source when they haven't looked at the real implications, they just rely on the rumors.

This chapter makes an attempt to provide information about the most popular FUD surrounding open source. Each topic starts with a FUD statement followed by discussion about that topic. A true, false, or maybe will also accompany the statement giving the general idea on the

truth of the statement. Though one should take these with a grain of salt. They are not meant to provide a set in stone decision on the FUD. One is encouraged to do their own research and find out the information for themselves, which may provide a more customized answer.

Open source has a much lower price.	**True**

The price of an open source program is usually far less than a comparable proprietary program. Open source software doesn't have to be "no fee," but most programs are. Users can either download the software directly or pay a negligible fee to have a CD-ROM burned and shipped. Current users set up distribution networks using community Web sites and CD burners. Their motto is "share and share alike." Open source means anyone can try any program first for free. A user may eventually buy a formal copy (perhaps to get better service), but doesn't *have* to do so. The software will never expire or demand payment.

This may seem crazy. But open source proponents reiterate that the software industry started this way. Activists like Richard Stallman see proprietary software as an abomination. As the car analogy suggests, the open source community could eliminate the retail sale of software. Yet many people prefer to pay for open source software, for the service, quality assurance, and other value-added elements. This is especially evident with Linux distributions, which strive to offer turnkey solutions "out of the box."

By combining inexpensive or donated hardware with an open source operating system and open source programs, schools may be able to deploy computers for less money. This may allow for technology where none could exist (and may address some equity issues). A current user explains: "This was not really a choice between Linux and Windows, but a choice between Linux and nothing...." Many current users are attracted by the price. They continue using open source because they believe the other costs are also less than proprietary solutions.

Some software isn’t compatible with open source.	**True**

Choosing any solution may foreclose on other software. This so-called opportunity cost may not be apparent for years, when the need for the other software emerges. In many cases, open source is still the minority solution. For example, the number of Linux desktops is meager compared to Microsoft Windows. By choosing a Linux desktop, a user forecloses on some software because it may never be created for or ported to Linux.

Opportunity cost is unique for each user or school, based on their needs and goals. A school doesn't have to migrate exclusively to open source and may find incremental advantages by phasing out proprietary software. Emulation may also help offset opportunity cost. Opportunity cost is highest when a specific, critical program is unavailable or incompatible. Some open source proponents downplay this cost, arguing that such specific needs are rare.

Schools may want specific, mission-critical software that's unavailable or incompatible with open source solutions (e.g. specific software for curricula or administration). Clearly, such high opportunity costs should be balanced against any advantages of open source. If more schools adopt open source solutions, the marketplace and community will shift, and new or existing solutions will become available. Finally, as more applications become Web services, the need for specific operating systems or applications will be less important, since access will be browser-based.

> I question the need for MS software and for traditional PC's. Seniors, kids, most of the people who just sit down and want to use a computer have very basic needs: 1. Click on an icon to write or surf. 2. Save it. 3. Print it. Using proprietary software to do this is unnecessary. ... Most people really don't care about the OS. They just want to click and go. (Paul Nelson, Personal communication, December 18, 2001)

People in the real world use X (insert any program here) so we should teach it.	**False**

One argument often spoke for proprietary programs in education is that a certain program is widely used so schools should teach that. At first glace this seems like a smart move. The idea appears to provide students with the tools they will need when they get into the “so called” real world. When given some thought, the idea doesn’t seem to be able to tread water.

In the constructivist view of education, teachers provide students with a base of information to build upon. Instead of viewing students as blank slates to be filled, they are active in a process to construct new ideas or concepts based on their current/past knowledge. Teaching specific programs goes against this very idea and robs students of the ability to learn concepts.

> Concepts are more important than specific applications.

Learning concepts trumps learning a particular way to do something. When we teach a person to drive a car, we don’t teach them to drive a Ford instead of a Dodge. This would be ridiculous, but it is exactly what we do when teaching certain program instead of the underlying concepts.

Microsoft Word is a program that is often said to be important in the real world. Many businesses use it and schools feel they should provide students with skills in this particular program. Even a quick inspection of this reasoning shows that it has no merit.

Ask yourself what is more important, word-processing skills or MS Word skills. They are not one in the same. Providing students with general or specific knowledge of word processing gives them the skills to work with any word processing program, not just Word. There is nothing stopping one from using Word to do so, but things like cost and licensing start to get in the way. The most salient point here is concept over specificity. One should choose a tool that will allow them to do so.

Another point against the argument of using only Word is change. Some educators feel that they are teaching skills in Word because they will use them in the work force. This is not a valid argument. Word was written in 1983 and since then it has gone through many changes. Changes in features, user interface, file format, and many others. By the time a student gets into

the work force the version of Word will look nothing like the version they were using in grades K-12. If all a student learned were specific skills and how-to's, they would be lost when a new version comes out. If they were provided with sound concepts in word processing then they would be able to accommodate version change or even another word processor.

This goes with any program. The highest importance should go to learning concepts. When teaching web authoring, educators should not just use the features in Adobe GoLive. Instead, they should teach the concepts of web design (e.g., HTML, basic WYSIWYG, etc.). By learning the concepts, students will be able to understand the topic and process even if they forget the exact logistics of the process.

Ask yourself what good learning is. Does it encompass a lifelong learning experience or reliance on a specific product?

Proprietary software is more user friendly	**Maybe**

On the frontend, proprietary software has generally had a better track record of providing a working environment with more desirable features. It may be easier to use peripherals like digital cameras with proprietary software. This is because companies like Epson and Canon, who make the cameras, generally write drivers (the software that makes the camera work) for the platforms that give them the highest profits. Today, as Linux becomes more popular on the desktop, more and more companies are jumping on board and supporting their products for the platform. For example, Centrinity, who is a popular vendor of the Firstclass email software, is actively developing their software to run on Linux.

Linux also has a bad reputation of being hard to use. While in the past this may have been true, today Linux can be just as, if not more, user friendly that Microsoft Windows. Red Hat is a good example of a company that is striving for ease of use while still providing the stability and power that comes from running a Linux based system. Installing Red Hat is as simple as putting the disc in and following the directions on the screen. Many have said that it is easier that installing Windows.

Working in Red Hat Linux is also very similar to any other GUI based computer system that you may have used in the past. It uses icons, text, buttons, and menus to allow the user to do certain tasks. If a user has experience with Mac OS or Microsoft Windows then they will feel right at home.

Installing software is another area that proprietary solutions have generally made easier. Microsoft Windows and Mac OS have made installing software as easy as a couple of clicks. Open source software has usually required the user to do certain tasks on the command line. This has been a problem for the regular computer user who has never had experience working with a computer in this way. This of course is also a problem that open source proponents realize and are working to fix. Software like the Red Hat package management system and apt-get provide relatively easy software installing. They are still a ways away from the simplicity of some proprietary solutions.

Schools also may not find mature open source solutions for certain needs. The open

source community is targeting these needs and with a little effort, most proprietary solutions can be replaced by an open source one. On the backend, both open source and proprietary solution often have rich features but include a significant learning curve. Fortunately, the people who are most likely to be dealing with web and file servers are more comfortable with a command line interface, which may make the open source solution more attractive because of its other positives.

Open source is not mature enough for schools.	**FALSE**

The open source development method is not new and much of the software has been used and improved upon for years. Linux is a UNIX-like operating system and UNIX is mature and reliable. It has been around for more that 30 years. The Internet depends and thrives on open source software. In fact, the Internet as we know it today would not exist if open source software did not exist. Business and government are deploying open source solutions at an amazing rate. Schools are also starting to jump on the bandwagon and finding many solutions that are both mature enough for their needs and more cost effective. As more educators join the open source community and give back to it, they will find more education specific programs.

With open source you only pay for what you need.	**TRUE**

Open source software is usually more modular than proprietary alternatives. Modular software mean each program is a discrete piece of a solution. If the user finds a better program, it can be swapped in without replacing the whole solution.

Proprietary solutions often come as packages, sometimes with features most users don't need. It can be difficult or impossible to replace a specific program for something smaller, more reliable, or otherwise more desirable. In contrast, open source is famous for offering several interchangeable, modular programs for various needs. This approach is summarized as “There's More Than One Way To Do It” (TMTOWTDI). Users can find the best components and only pay for what they need (when they have to pay). Open source solutions are usually scalable at little or no cost, while some proprietary solutions are incrementally more expensive with each new user or machine.

As a frontend operating system, Linux remains a niche platform. Low threshold open source programs like OpenOffice.org are constrained by the prior design and limitations of Microsoft Windows. To take full advantage of modular design, a solution has to begin with an open source operating system like Linux.

Open source makes license management easier.	**TRUE**

License management is much easier with open source. Users can install any number of copies, so open source companies don't bother with complicated licenses (and most of the possible licenses favor nonprofits like schools). There is no risk of illegal copies or license audits, and there are no anti-piracy measures (e.g. CD keys, product activation). Current users praise this freedom for licensing liability.

Schools that are using proprietary software can be (and are) asked to conduct an audit. This means that they have to match all proprietary software to a license that they own. This becomes a huge task for schools that are already strapped for time, money, and people. School also receive a lot of donated computers with out licenses on the software that is running on them. Also schools may not have the knowledge of proper use of software. Some teachers may buy a piece of software and put it on all their computers when they are only allowed to put it on one at a time.

Open source doesn't eliminate the need for software management though. Schools should still track which version of what program is on which machine. The quality of any program, especially for security, depends on patches and other updates. Notable, with a thin client model like K12LTSP, software management is minimal. This is due to the school only needing to track and update the servers.

Schools don't have the luxury of experimenting.	**Maybe**

Migration from a proprietary solution to an open source one may be challenging and disruptive. Schools are currently facing a variety of problems, including uncertain finances. Educators may not have the time or energy to revisit existing technology. However, open source software may solve more problems than it creates, after the initial investment and migration. Stakeholders may be more receptive to change when the potential TCO advantages are explained. Current users feel they would be irresponsible to exclude any option that could maintain or improve existing technology at a lower cost. And as schools upgrade or replace technology, they may have the opportunity to explore open source with reasonable risk. For example, a school could install OpenOffice.org on all new computers, and only purchase Microsoft Office if OpenOffice.org was found inadequate.

Software is better when it's transparent.	**TRUE**

Proprietary companies are already making their software transparent.	**FALSE**

If software is transparent, any programmer can see what happens and why. Proprietary software is not transparent. Most computer users are familiar with the strange error messages that appear when a program fails. Often, these messages use special codes to express the problem. These

codes point to secrets without actually revealing what went wrong. Only someone who knows the secrets can understand the codes. For example, Microsoft Certified Engineers are taught to decipher these codes for Microsoft products. This agitates open source proponents because they want more independence and competitive service in technology.

Consider the analogy of an automobile. If a car was proprietary, there would be no oil gauge, tachometer, batter meter, etc. When the engine failed, the car would just offer a special error message. Only a certified mechanic could make sense of the message.

Because open source programs aren't trying to protect secrets, they can offer more exact information about an error. This allows any programmer to identify and fix problems. This is the same principle that motivates teachers to use open gradebooks: students deserve a chance to see how their grade is calculated, and to challenge possible errors. It's the suspicion of such error that causes people to distrust proprietary programs or closed gradebooks. Some countries are suspicious of proprietary software because it could contain spyware or other security threats.

Microsoft, sometimes open some of their code to some colleges and universities for educational purposes. (Microsoft calls this “shared source.”) Microsoft is also starting to open their code to countries like China. This open code is not open source because of the restrictive licensing agreements involved. Only the proprietary company can make changes or distribute copies. Also, it's unlikely that K-12 schools will be given access to code or significantly benefit from others' limited access.

Open source threatens intellectual property rights.	**FALSE**

Opponents argue that open source threatens intellectual property rights. The CEO of Microsoft has said, "Linux is a cancer that attaches itself in an intellectual property sense to everything it touches." (Steve Ballmer, in Newbart, 2001) This is a specious argument designed to scare people away from learning more about open source. It's inspired by the most radical open source license: the GPL. Source code released under the GPL can't be included in closed, proprietary software. But any content created using open source software (including GPL software) still belongs to the author. For example, authors who create documents in OpenOffice.org still have copyrights on their work. With open source, only programmers can lose their intellectual property rights. They are willing to give up exclusive ownership of their code to contribute to the community and benefit from community-created programs. Authorship is still respected and rewarded. Open source does not mean public domain, although some licenses explicitly make a program public domain.

Almost all software is created from existing software, using existing designs ("art") and common libraries of code. Hypothetically, if all software was licensed under the GPL then it would be almost impossible to create proprietary software without starting from scratch. But few people want or expect all software to become proprietary. The open source model allows companies to create proprietary software while preserving a commons of open source code (e.g. Apple OS X and Darwin).

The technology offers many potential advantages and challenges, but it's alarmist to equate open source with the theft of intellectual property. New business models and social norms

are developing, just as they have in the past.

> Music publishers tried to sue player-piano makers out of existence, fearing that no one would ever buy sheet music again. Fifty years later, in 1984, Motion Picture Association of America President Jack Valenti uttered what's undoubtedly the most infamous comment in the history of technophobia: "The VCR is to the American film industry what the Boston strangler is to a woman alone." Today, video rentals account for more than 40% of studio revenues. (Black, 2002)

The distinction between code and content may be blurring, especially on the Internet. Some code may be meaningless without content, and vice versa. If schools join the open courseware movement, they can be strategic in choosing and creating software with desirable licenses to preserve the amount of intellectual property rights they want.

Proprietary software threatens civil rights.	**Maybe**

As society and the economy become more dependent on technology, the implications for civil rights become more vivid. Some proponents warn that proprietary software threatens civil rights in the digital age by giving too much power to software companies and governments. Even so-called "offline" rights are now affected by software (e.g. medical records, credit ratings). Proponents see an assault on privacy, free speech, freedom of assembly, and similar rights, and threats to fair business and transparent government.

Schools are a traditional crucible for debate about some of these issues. Educators should be aware of the debate since the software they choose will affect their students' rights as well as their own. Any proprietary or open source system must be secure, to protect data like financial records, medical records, and job performance. Such security should be in the hands of those most liable: the system administrators.

The early Internet had few rules. The original network was designed for expression and collaboration. Over time, growing concerns about security and crime as well as minors accessing inappropriate content have lead to legislation and technology for monitoring users and censoring content. While the general intentions may be good, the results can be abhorrent. Proponents don't trust companies to create software in the best interests of users' rights, and argue that software itself creates an architecture that fosters totalitarian power. Lawerence Lessig explains:

> This code, like architecture in real space, sets the terms upon which I enter, or exist in cyberspace. It, like architecture, is not optional. I don't choose whether to obey the structures that it establishes -- hackers might choose, but hackers are special. For the rest of us, life in cyberspace is subject to the code, just as life in real space is subject to the architectures of real space. (Lessig, 1998)

Lessig emphasizes the control code creates. This includes control of content: "how it is played, where, on what machines, by whom, how often, with what advertising, etc."

Those who know the secrets have power over those who don't.

(Lessig, 2001) This idea of control-through-architecture is especially apparent in formats. For example, HTML is an open format so anyone can create and distribute a Web browser. But if HTML became a proprietary format, then only certain Web browsers would work. A browser monopoly would exist, and the monopolists could influence the Internet in profound ways. They could include spyware to monitor users and collect data (e.g. for immoral marketing or unjust law enforcement). They could prevent certain pages from displaying properly if they disliked the pages' owners (e.g. other companies, consumer activists). With a proprietary browser it would be difficult to detect or prove these violations. With a monopoly these violations wouldn't even need to be secret. Users would have to consent to the monopolists' terms in order to use the Internet. This is a critical idea in open source: with secret code and formats, those who know the secrets have power over those who don't.

With open source it's nearly impossible to hide spyware or similar threats to privacy and free speech. No single company can control open source so a monopoly can't form. This makes it difficult for unscrupulous corporate employees or over-zealous law enforcement personnel to leverage control of code to violate rights. This is very similar to the principles of transparent government and freedom of information, which so-called "sunshine laws" try to protect. Citizens deserve to know what their elected representatives and government officials do with their taxes and authority. Just as "democracy dies behind closed doors," proponents fear that proprietary software can protect corrupt business or unconstitutional government.

Open formats are better.	**TRUE**

Open source usually uses open formats. As a result, open source programs are generally better at working together and sharing files. For example, the Free Standards Group oversees the Linux Standard Base and the Open I18N standard, which are designed “to make it easier for software to run on different companies' versions of [Linux].” (Shankland, 2002, Verizon)

Data management offers another example. Many states are trying to integrate their data management systems throughout all schools and districts. Open formats make integration much easier. For example, suppose different districts in the same state have different database programs, and each program has a proprietary format. To integrate the data, the state would have to select a program that licenses all those formats, or convert the data to the new format, or abandon some data. Using open formats forecloses on these problems and may curb opportunity costs.

The greatest potential advantage of proprietary formats is quality control. By controlling and licensing a format, a company should be able to promote smooth file exchange between programs (e.g. The different programs in Microsoft Office). Open formats may mean a lower TCO, however, especially since the opportunity cost is high if a school locks data into a proprietary format.

The open source movement is partly a response to incompatibility in proprietary software. One of the driving reasons for Richard Stallman's idealism comes from having to deal with software that wouldn't work well together and the fact he could do nothing about it.

Open source is anti-business, anti-democracy, anti-American, etc.	**FALSE**

Proponents of open source are often told that their model is similar to communism. Nothing could be further from the truth. If anything, open source holds closer to the ideals we hold dear. Sharing, helping, competition are all center pieces of open source. Open source is not communism or anarchy. Old and new American capitalists are calmly thriving by using and selling open source software. Open source may represent a paradigm shift for the technology industry, but it's not anti-business. As the car analogy suggests, open source may undermine some existing businesses. There's little reason to buy a proprietary program when a comparable open source alternative exists. However, the software industry was already transforming from a product model to a service model. Companies like IBM are thriving with open source because they offer quality service. Red Hat sells open source software with value added, including service. The community helps supply the quality code.

Open source is a model of individuals with better ideas, organizing projects of volunteers, and perhaps becoming entrepreneurs in the process. That's a very democratic, American model. Schools should dismiss arguments about "evil software" without hesitation.

Open source software isn't reliable.	**FALSE**

Anyone who tells you that open source software is not reliable is just pushing FUD and have not looked at the facts. Let's take a look at a couple pieces of software that are open and have also made a major impact on computing.

The Internet would not exist as is does today with out open source software. The infrastructure that is the foundation of the Internet relies heavily on open source software. When a person types a web address into their web browser or sends and email they are relying on the Domain Name System or DNS. At the center of the DNS is a program call BIND. Basically it allows users type www.google.com instead of 216.239.37.99. Bind was developed at UC Berkeley and has been maintained as free software under the Internet Software Consortium.

Sendmail is another piece of software that has allowed the Internet to flourish. Sendmail is a mail transport server that allows email communication for about 75% of all Internet sites, even ones who don't know they are using it. Almost every email message sent over the net relies on sendmail for either incoming or outgoing mail. It was created by Eric Allman, again at UC Berkeley and has been maintained as free software by its creator.

The Apache Web server is another open source project that has proven to be reliable. It is responsible for hosting more than 60% of the Internet's web pages. It has proven so successful that companies like Apple, who in the past made their own Web servers, have made Apache a core part of its operating system.

These products have proven themselves on a grand scale to be super reliable and they have helped create the greatest communication system man has ever created.

Open source only matters to programmers, since most users never look under the hood anyway.	**FALSE**

When talking to educators about open source software is it common to hear the question, “Why should I, as a teacher, care if I can see the source code?” The answers to this question are further discussed in the next chapter but deserve attention here since it is a common piece of FUD that gets thrown around.

World economics show us that free markets work. When there is competition between multiple suppliers, the consumer wins. Out of this type of market comes lower prices, more innovation, and more specialization to meet new market niches. A huge benefit of using open source software is that you are no longer locked into one supplier. You are no longer at the mercy of a huge corporation who's only aim is to make money. Open source software comes from a need to accomplish a task. This leads the purpose of development to making the best possible piece of software, instead of a monetary goal.

Since everyone can see the source code for Linux, everyone is on an even playing field. The company that does the best job at creating software gets the top prize. If they fail at providing good software then nothing is keeping someone else from doing it better and taking it's customers away.

One of the most often cited reasons for educators to care about open source is monetary. Schools can save money by using open source solutions. By using Linux a school can download one copy of the software, put it on all computers, and keep them up to date for no cost. This is just not possible for schools to do with a solution using Microsoft Windows or Mac OS X.

It is popular to counter this piece of FUD with car analogy.

Part II

What Open Source Means for Education

Chapter 4 – What Can Open Source do for Schools?

Chapter 5 – Open Source Software in Education

Chapter 6 – Other Ways to use Open Source in Schools

Chapter 4

What Open Source can do for Education

We have now looked at what open source is and where it came from. Now that we have this foundation, it is time to look at some of the ways open source can make a positive effect on education. While the list of specific ways this can happen is immeasurable, the following chapter will show some of the most obvious.

First things first though. It must be understood that the use of open source software in schools is not a silver bullet. It will not solve all problems in schools. Decision makers have to look at how open source can fit into their specific situation and go from there. Chapter 7 goes further into the actual process of making the transition to using open source and makes suggestions about how to go about making wise decisions. This being said, open source does have an abundance of potential to being a major player in how we educate our youth. It can provide us with ways of solving some of the most pressing issues in using technology in schools today.

Saving Money on Software

One of the most obvious ways open source can help education is the money that can be saved by using it. There are many other reasons, and many would say more important reasons, but no one can dispute the positive effects open source can have on a schools budget.

It comes as no surprise that schools are strapped for cash. The reliance on local taxes has proved not to be totally sufficient to cover the ever increasing cost of public education. The use and integration of technology into the curriculum becomes a hard sell in this environment.

There are generally two categories on which money is spent in regards to technology. The first is hardware. This includes physical components like hard drives, keyboards, and monitors. The second cost is software. This includes the programs that make the hardware work and also the category we are concerned with here.

Licensing. This one word has brought many headaches to technology coordinators around the world. The idea is simple; you pay the software company for the right to use their software. Though the actual application of that idea gets complex very quickly. There are different licenses for different uses and the restrictions put on the use of the software is enough to confuse any intelligent person. Read the Microsoft End User License Agreement (EULA) sometime and see how complex it is.

The concept of licensing software is also up for investigation. When an individual pays for Microsoft Office thy are not buying what they might think. This person actually does not own anything other than the packaging. They do not own that copy of MS Office. Rather he or she have paid for the rights to use the software. What's the difference? The difference between owning something and being given the right to use it are very different. Instead of the purchaser owning the software (meaning they own it and can do what they want with it), the software company owns it and grants people permission to use it (with restrictions) once they pay.

Is this necessarily a bad thing? This way provides an avenue for companies to get paid for their work, right? This is American after all. We live in a capitalist society where people can sell their products. There is nothing wrong with getting money for ones work. Open source is and has never been against this idea. Richard Stallman wrote the GPL, which is a license that allows people to get paid for their work. Open source in not only about getting things for free, it is about the definition of the product a person uses.

With Open source software, one is usually not buying the actual code. Instead, many companies that are in the open source business sell support for the product. Red Hat, for example, sells support for its Linux distribution while still allowing users to see the code and basically download it for free.

The main interest this holds for educators, is the availability of software at either no or low cost. A school's monetary resources can be put to better use and go further when using open source. It can reduce the total cost of ownership (TCO) by incredible amounts that are impossible using traditional software products.

This becomes clear when looking at an example. One of the most widely used products on computers is the office suite. An office suite is usually comprised of a word processor and a spreadsheet program. It also commonly includes presentation programs and database support. The most popular is Microsoft Office and many schools have licensed this program from Microsoft at a considerable cost.

As of December 2003, Microsoft offers a license for students and teachers for $150. This number grows fast when buying licenses for more than one computer. 10 computers? $1500. 20 computers? $3000. This can be a heavy blow to any school budget, let alone any technology budget. One also has to take into account the price of upgrading to new versions when they come out. Choose not to upgrade? Microsoft has a history of using closed file formats so newer versions of Office do not work with older ones, forcing you to upgrade..

Many times, because of the cost, this software is not even licensed. It is easy to install a copy of MS Office onto many computers even though it is licensed only to one. This is illegal and can cause serious problems for schools. In 2001, Microsoft audited schools in 35 states to crack down on pirated (unlicensed) software. Schools could either suffer the audit or pay $40 for every computer in the school, PC or Mac, that could run a Windows application, whether it did or not.

If only there was a product that has similar features to MS Office and was available free of licensing hassles. To good to be true? Not with open source. There are many open source office suites that provide users with similar products to Microsoft's. One of the most popular is OpenOffice.org. It is a full-featured office suite including software to word process, make spreadsheets and presentations, create drawings, and even provides support for databases.

OpenOffice.org also has support for other file types, which means it can read and write to

Microsoft Word or Excel files. It uses an open file format based on XML, which is an industry standard that allows for interoperable structured data. In laymens term, this means that files today will work with files tomorrow.

OpenOffice.org is certified open source and licensed under the GNU Lesser General Public License (LGPL) and the Sun Industry Standards Source License (SISSL). This allows anyone to view the source code and make modifications if they feel the need. The program is also available to download at no cost. If downloading 80 megabytes presents a problem, they offer the option of spending 5 dollars to have a CD-ROM shipped to you. Versions exists for all major computing platforms including Windows, Mac OS X, and Linux.

The open source world is full of these types of alternatives. One thing should become apparent: with open source, schools can wisely spend their money while remaining current with technology. Whether rich or poor, every school has the responsibility to the student and tax-payer to at least look into using open source software.

Lowering Your TCO

Total cost of ownership, or TCO, refers to the complete cost of any solution. In regards to software this would include sale price, any hardware and software upgrades, maintenance and technical support, and training (or re-training). Time and frustration and can also be thrown into the mix, although they can be hard to measure. All of these components need to be considered when making decisions on what technology to use.

Using open source software does not automatically lower your TCO. It would be foolish to think so. However, many people around the world have successfully implemented open source software and saved money through out their budgets. Let's take a look at some of the different ways Open Source software can lower TCO.

The most obvious effect open source software has on TCO is the upfront cost. As mentioned in the previous section, the price of an open source program is usually far less than a comparable proprietary program. The price to upgrade an open source program is as easy as downloading the new version. No worrying about licensing or getting money to get the latest and greatest.

A second way open source lowers TCO is by maximizing existing resources. Open source software typically (but not always) has lower hardware requirements than proprietary alternatives. New versions of Microsoft Windows require faster and more powerful computers – referred to by open source proponents as "bloatware." On the other hand, Linux can run well on older hardware. This extends the life and usefulness of computers already in schools. The old donated computers may actually have a use and provide students with a tool that will help them learn. The lower hardware requirements also strengthen investments in new hardware. You get more for your money if a computer you buy today can stay up-to-date and operational for longer.

Open source software also increases the ability for scaling. Scaling is simply taking an existing solution and making it bigger. For example, expanding a computer lab with 10 computers to one with 20 computers. As the situation gets bigger, if you are using proprietary software, your cost goes along the same curve. This is not the case with open source software. In the case of Linux, you can download it, burn a copy, and install it on as many computers as you want. There

are no fees per computer or site licenses to worry about.

Another positive effect open source can have on TCO is the stability and reliability that usually accompanies open source software. It may not make as many errors or crash as often. Since anyone can see the source code of open source software, bugs can be repaired quickly. Author and hacker Eric Raymond call this Linus' Law: "Given enough eyeballs, all bugs are shallow." The software is in constant review and can be quickly patched. Linux has a track record of being one of the most stable systems available. Linux administrators constantly talk about rebooting in terms of weeks and months, not days like Windows users have become accustomed to.

Greater security is also often touted as a way to lower TCO. With the source code being available for all to see, holes that would allow worms or viruses to do damage are found and fixed at an amazing rate. This transparency is sometimes misrepresented as a critical security flaw. It has been said that seeing the source code makes it easier for a criminal to gain access to a program. However, the open source model leverages a community of programmers to maintain and improve security. The collective benefit has shown to outweigh the danger of transparency. Linux leads the industry in defensive design and businesses and governments around the world are taking notice. Schools should be as well. Linux isolates most users from the code viruses and worms need. In a properly configured Linux environment, an email attachment in a user's inbox can't spread destruction to the whole system. This design will lower TCO by lowering maintenance costs and keeping computers up and running. The Blaster worm and SoBig.f virus that plagued windows users in late 2003, had no direct effect on users running Linux or other non-windows based computers.

Getting support for traditional proprietary programs is another area of considerable cost to education. Often support is paid for separate from the product itself. With open source, support is mostly provided by the community. While to an outsider this may seem unreliable, in actuality it is a more efficient and powerful way of finding solutions to your problems. It takes the responsibility out of the proprietary companies hands and puts it into yours. This can be scary at first, but soon everyone sees the usefulness of this method. The open source community is full of listserves, mailing lists, and web sites devoted to supporting users. Linux Torvalds, when he was developing Linux, was often known to release a fix within hours when made aware of a problem. This is a common occurrence. Instead of being put on hold by a bottom level help desk person, you can get help from people that know what they are doing and are eager to help you and contribute to the community. It is also common to be able to talk to the actual developers of the open source software. Something that is almost unheard of with proprietary software.

Even with these strong reasons, open source can not automatically lower total cost of ownership. Proprietary software companies claim their TCO is lower while open source software companies argue the opposite. One long-term study of Web server deployments found lower TCO for Linux over Microsoft Windows and Sun Solaris. (Orzech, 2002) But Microsoft alleges lower TCO with its "comprehensive, integrated, easy-to-use stack of technologies" and has its own favorable studies. (Cooper, 2003) Decision makers need to look at their unique situation and see how open source can lower their TCO. They need to be aware of the issues at hand and see if they have the resources needed to do it correctly. Only with a well thought out plan can TCO be lowered with open source. Figure 3-1 shows a concept map of TCO and some of the aspects

involved.

Figure 3-1

The Digital Divide

The digital divide is a problem facing schools and society. For those without knowledge of this phenomenon, here is a great definition from whatis.com.

> The term 'digital divide' describes the fact that the world can be divided into people who do and people who don't have access to - and the capability to use - modern information technology, such as the telephone, television, or the Internet. The digital divide exists between those in cities and those in rural areas. For example, a 1999 study showed that 86% of Internet delivery was to the 20 largest cities. The digital divide also exists between the educated and the uneducated, between economic classes, and, globally, between the more and less industrially developed nations.

Educators need to be aware of this when they use technology. One of the major reasons people do not have equal access to technology is lack of funds. The lower cost of open source does a lot to bridge this gap.

Schools that do not have a lot of funds can use open source to get access to technologies that would be impossible using the usual routes. Buying computer parts, putting them together, then installing Linux can cost as low as $200. A comparable computer through a company such as Dell, would cost around $900. Also a school that might think it can't afford anything, when taught about open source, might reassess their budget. Looking at the possibilities, open source is obviously an important tool to provide computing to all socio-economic levels.

Open Source in the Community

The use of open source software doesn't have to stop in schools. It can also help parents lower the cost of bringing technology to their children. Parents often make decisions about what computer to buy from the type their child uses in school. If the school is using open source software, the family may see the benefits of doing the same.

The issue of software availability is also made easier and possible by open source. It happens all to often that students use a particular piece of software at school and can only work on it at school. Schools often get discounts on software but these saving are lost to home users. It is also illegal for the students to bring home the software and put it on their computers. By using open source software, the tools their school is using become available and at low to no cost. For example, instead of using Microsoft Office, OpenOffice.org is available to anyone for free and provides the same features needed for most purposes. A school can download it and simply burn copies to provide students with the software for home use. Something that is not possible with proprietary software.

Schools that takes this approach can and should also be a center for getting this word out to the community. Low-income families might realize that they can afford a computer when going this route. Many vendors sell computers running Linux for as little as $200 (without a monitor). This is more apt to be in a families budget than a low end Dell running Windows for $700. The school can also act as support and help these families get technology. Not as a place to take advantage of the technology coordinator, but rather to use the school as a hub for help using the technology. The more communication between home and school, the better prepared students and their parents will be to integrate technology into their learning.

Greater Flexibility

Open source software is very flexible. Since anyone can see and modify the source code, it makes possible the ability for a programmer to port a program from one platform to another. Programs like OpenOffice.org and The Gimp run on Linux, Windows, and Mac OS X. This allows users to have the same software on all types of computers that they may use. Often, schools do have a mixed variety of hardware and software making this type of versatility very important. Flexibility is lessened when it comes to proprietary software. The power to bring a piece of software to a new platform relies solely on the company providing the software. Users have no say in the matter.

Open source software is also flexible in the sense that it works well with other software and future versions. OpenOffice.org relies on an open file format based on XML. This allows anyone to see the specifications on how OpenOffice.org stores date like formatting, font, and content. If someone decides to write a program and wants to be able to a document written in OpenOffice.org, because of the open file format, they will have a relativley easy time. This open format also leaves future versions of OpenOffice.org compatible with older ones.

This is the polar opposite of what proprietary software file formats provide. Microsoft has been famous for changing the file formats of its Office software every couple of years to force people into buying the new version of its software. With open formats, even if OpenOffice.org becomes obsolete (not likely to happen), people will still be able to use these files and not worry about them becoming lost to changing file formats.

Minimizing Upgrade Costs

When considering total cost of ownership, the cost of upgrading software and hardware should come to mind. It has been stated that the upfront costs of open source software are almost always free, and it goes without saying that software upgrades are free as well. When it comes times to do a major upgrade to a computer running Linux, all the user needs to do is download the new version and install it. Upgrading to a new version of Windows or Mac OS X can cost the user $200 and $130 respectively.

Not only can open source save money on software upgrades, it can also save on hardware upgrades. Open source software has a very good track record of being less demanding on hardware than its proprietary counterparts. This means that the hardware lasts longer and will require less upgrading to keep up with the demands of new software.

The easiest way to minimize upgrade costs (monetary and time) is through the use of thin clients. When all the software that computers run are on a server, the only upgrading that needs to be done is to the server. Client computers do not need upgrading for a very long time because of the extremely low minimum requirements for them. An example of thin client software is the K-12 Linux Terminal Server Project. It uses the Fedora (formerly Red Hat) Linux distribution and the Linux Terminal Server Project to provide an easy way for educators to provide powerful and easily manageable thin client computing to students.

Provide Greater Security

Security threats like viruses and worms exploit bugs in software to damage computers. The transparency and reliability of open source may mean better security. Linus' Law describes parallel debugging: multiple programmers independently finding and fixing the same program to discover the best solution. Eric Raymond explains:

> One advantage of parallel debugging is that bugs and their fixes are found/propagated much faster than in traditional processes. For example, when the TearDrop IP attack was first posted to the web, less than 24 hours passed before the Linux community had a working fix available for download. (Raymond, Halloween)

With both open source and proprietary software, effective security depends on thoughtful deployment, regular monitoring, and timely upgrades or other modifications. Some proprietary solutions offer potentially robust security, but Linux leads the industry in defensive design. (This is a critical reason why business and government are interested in open source.) For example, Linux isolates most users from the code viruses and worms need. These threats need unchecked access to execute, replicate, and deliver a payload (e.g. Delete all files). In a properly configured Linux environment, an email attachment in a user's inbox can't spread destruction to the whole system. For schools, security also includes Web filtering. Open source offers popular and effective solutions (e.g. SquidGuard) to restrict Web content to align with school policies.

The transparency of open source software is sometimes misrepresented as a critical security threat. The argument is that since a malicious cracker (not hacker) can see the code of a program, they can take advantage of it. However, this openess is exactly what makes it more secure. The open source model leverages a community of programmers to maintain and improve security. This collective benefit seems to outweigh and perceived danger of transparency. This is an easy point to make when you look at Microsoft, a company who is very secretive with their code, and the trouble they have had with viruses and worms.

Also, the security of a system depends much more on careful deployment and maintenance. Just because the source code for Linux is publically available, does not mean the security of Linux is automatically fool-proof. A system administrator will still have to deploy passwordws, firewalls, additional software, and other defenses. These other methods add to the already secure system. Without careful deployment they security of a system may be compromised, no matter how well the system was designed. For example, if a user gives away their administrative passwords, then anyone may gave unlimited access to a system.

The open source movement is partly a response to safety and security concerns with proprietary software. Open source offers powerful potential security advantages by preventing spyware and promoting encryption. However, some of these issues are fairly remote from schools. One should remember to take security in mind when sensitive data is put onto computers. Remember, there is not security through obscurity.

To Teach Computing Concepts

One important idea in education is learning concepts. In the area of technology this is no less important, but too often it is forgotten. Many times too much effort goes into teaching procedural knowledge of specific applications. For example, Adobe GoLive is a program for creating Web pages that is aimed at the advanced user. It uses certain abstractions in an attempt to make creating Web pages more efficient. When a person who has no experience in Web page creation first learns on GoLive, they miss learning the underlying concepts of HTML. This the teaching to go into the “click here, then here, then here” type of learning. The problem with this type of learning is that these processes are easily forgotten and, more importantly, hinder the content learned to that specific application. When these GoLive beginners try to use Dreamweaver or Frontpage or even try to learn HTML, they will have no advantage because they failed to learn the fundamental concepts of Web page creation. Technology literacy should mean more than skill in specific applications. It is of far more worth for future employees and consumers to be taught

general skills to avoid unnecessary dependence on any specific program. Students will see software change many times before they graduate, they need to be lifelong learners of technology.

Since open source software doesn't have the tendency to change file formats or become incompatible with itself, teaching concepts is easy. One popular argument against using open source is that the students will be entering a Microsoft dominated work force and they need to learn these programs. In reality, Microsoft Word (the word processor in the Office suite) has changed so many time since its inception that teachers constantly need to change the way they teach it. OpenOffice.org offer word processing very similar to Microsoft Word. In fact, they are so similar that learning most features in one will transfer to the other. Word processing as a concept is a valuable tool for students, teaching specific programs are not.

Since obtaining open source software is inexpensive, educators do not have to worry about only teaching one way to do something. They can use multiple programs to teach word processing or Web page development. There by teaching from multiple angles and providing a greater breadth of information.

This is not saying that schools may not have a situation that may require a specific program to be taught. This is more likely to happen in high school because they are closer to the work force or a school that might be using a particular program. A school has to look at their own situation and decide, for themselves, which direction to go.

Chapter 5

Open Source Software in Education

It is clear that the open source community has a lot to offers educators. There are thousands of projects that are currently being used in schools and many more that could be. While presenting every program would be nice, it is far beyond the scope of any book. This chapter looks at a few open source projects that schools are using.

The best way to learn about open source software is to join the community. There are many places on the Internet where you can find software and communicate with others who might be able to point you in the right direction. Sourceforge (sourceforge.net) and Freshmeat (www.freshmeat.net) are two such popular repositories for open source projects. There are also many mailing lists available for educators who want to use open source. Red Hat hosts the K12OSN mailing list, which users from around the world ask and answer questions about open source. Schools interested in open source should look to the open source community for advice, opinions, and guidance.

This chapter presents many of the popular open source project available that are applicable to a school setting. No attempt is made to explain everyone in depth (history, installing, etc.). Instead, projects are touched upon to show that the open source community has many projects that can be used in education. You should do your own reasearch to find programs that fit your schools needs.

Linux

Developed by Linus Torvalds in the early 1990s in his dorm room, Linux has exploded into many markets. Businesses are using it to lower their spending costs. Governments are using it for the stability it provides. Schools can reap the same benefits by using Linux.

Linux is technically a kernel, not an operating system. A kernel is the core of an operating system that handles low level tasks. Companies like Red Hat and Mandrake use the Linux kernel and bundle other softwhools can also use Linux to make use of technology in education more effective.

Linux is technically a kernel, not an operating system. A kernel is the core of an operating system that handles low level tasks. Companies like Red Hat and Mandrake use the Linux kernel and bundle other software to make their own distributions. Many people just use the term “Linux” because of the ease.

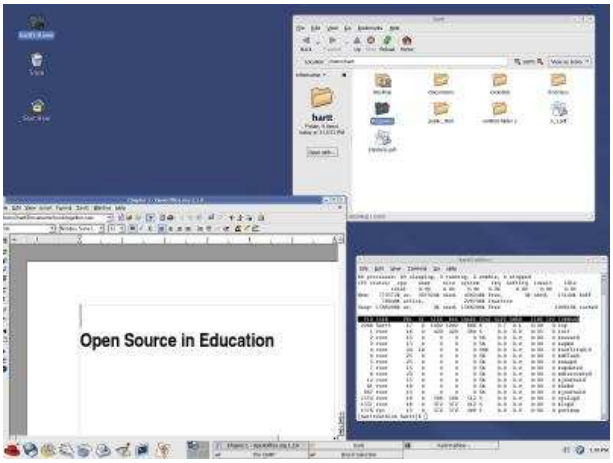


Figure 5.1 – Typical Linux desktop.

Linux provides computer users with an operating system that embodies the open sourrce movement. Most Linux distributions are free of cost and can be downloaded from the Internet or bought in a store. Figure 5.1 shows a screen shot of a typical Linux desktop. As you can see, everything one would expect from an operating system can be found in Linux. Menus, windows, icons, text, and other user interface components are used similarly to the ways they are used in Microsfot Windows and Mac OS. Most users find using Red Hat or Mandrake as easy as using these other proprietary operating systems.

Schools may find Linux more apropriate to use due to its cost benefits and the ability to provide students with a multitude of software. Linux can be used on all major hardware platforms and may be able to provide a uniformity to working environments on the diverse computers schools sometimes have.

A great place to start learning about Linux is the Internet. There are many sites out there devoted to educating the masses about Linux and what it can do. There are also many books on the subject that present material in a readable manner rather than “geek-talk.” Appendix B presents many of these great resources.

K12LTSP

K12LTSP is based on Red Hat Linux and the Linux Terminal Server Project. It is a project developed for education that is easy to install and configure, and it is distributed under the GPL. Once installed, K12LTSP lets you boot diskless workstations from an applications server. You can use old PC's as diskless clients or buy new ones for under $200.

This is an example of the thin client model. All applications run on the terminal server. The workstations are “thin.” They have no need for software or hard drives. The clients run software off of the server. Thin-clients are perfect for schools becase they are easy to install and require little maintenance. There is no need to install a piece of software on all lab machines. Installing a program on the server makes it available to all client side machines in an instant.

K12LTSP is available to download for no cost. It includes many software packages that schools may find useful (OpenOffice.org, Gimp). To learn more, visit the project's home page at

http://k12ltsp.org/contents.html.

Apache

The Apache HTTP server is one of the most widely used open source projects in the world. A recent Netcraft Web Server Survey found that more than 64% of the web sites on the Internet are using Apache. Educators should be aware of this important piece of software as more and more schools choose to have an online presense. s to put it on multiple computers. They can have only one or two machines with Elements on it. Or a school could buy one copy of it and install it on every machine they want. The problem with this last option is that it is illegal. Though it happens all to often. GIMP, of course, presents another option. Educators can obtain G

There are many other web servers out there, but none have reached the popularity of Apache. Many schools may be using it without knowing it. Apple is using Apache in their Unix-based OS X and OS X Server. The most popular Linux distributions almost always come with Apache as the main web server.

Schools may be interested in Apache because of the cost savings and the security and stability the system provides. Scince it is open source, it is offered for no cost. It is also being constantly updated to keep up with the advances in the HTTP protocol and web developments in general. Interested users should start at the Apache web server homepage at http://httpd.apache.org/.

GIMP

GIMP stands for GNU Image Manipulation Program. It is a piece of software for photo retouching and image composition. It is a program that is very similar to Adobe Photoshop in ability and use. Gimp can be a simple paint program or advanced users can use it to do expert photo retouching. It runs on multiple platforms including Linux, MacOS X and Windows.

Schools often have the need for the ability to process images and retouch them in some way. Maybe just rotating them or adjusting the size is all that needs to be done. Some schools may even have the need to work between file formats such as jpg, gif, png, or tiff. Whatever the need, GIMP should be able to provide photo manipulation tools to schools for no cost. Gimp is also very expandable. Through the use of plugins and extentions, a user can augment the program to meet the specific needs of a school.

Many schools are meeting their photo manipulation needs with an Adobe product. Photshop is the professional standard and has wide spread use and support. The problem with Photoshop is that it is expensive. It is a professional application after all. Adobe has realized this and has released Photoshop Elements which is aimed more at the every day user. Many schools have gone this second route. Photoshop elements is cheaper than the full-featured version and provides most of the feautres that schools need.

Figure 5.2 – GIMP in action.

The problem with schools using Photoshop Elements is the same with other proprietary software. It costs too much. Even with the savings Elements provides over Photoshop, licensing becomes an issue very quickly when trying to provide multiple copies. Schools have multiple options in this case. They can pay the licensing fees to put it on multiple computers. They can have only one or two machines with Elements on it. Or a school could buy one copy of it and install it on every machine they want. The problem with this last option is that it is illegal. Though it happens all to often. GIMP, of course, presents another option. Educators can obtain GIMP for free and install it on as many computers as needed. Thus, providing educators with a powerful tool for image manipulation without the licensing hassles. See http://www.gimp.org/ for more information.

Tux4Kids

Tux4Kids is an organization that produces educational software that is open source. Their aim is provide educational software for any child that wants it. They maintain and fund many educational projects.

Figure 5.3 – Tux4Kids Logo

Tux4Kids also believe that one of the biggest problems facing schools today is proprietary software.

> Schools need technology in their classrooms. Parents want their children to be exposed to technology. Unfortunately, technology is costly and difficult to properly setup and utilize. Because of the expense, schools districts typically cannot purchase sufficient proprietary software licenses to completely cover their needs. Combine this with the fact that such software licenses are complicated and difficult to understand, and you get a situation where the dangers of copyrights violation and legal troubles rise...We would like to ultimately have a family of complete Free and Open-Source software solutions to fill the needs of schools, and allow them to remove the dangers proprietary software can cause them. (Tux4Kids, 2004)

There are many projects that educators might find useful. Tux Typing is an educational typing tutor for children. Tux of Math Command is a math progam that requires students to solve math problems while defending different cities. TuxPaint is a drawing program for young children. It is very similar to the popular KidPix art program that many school use. These programs are in active development and improving all the time. Visit www.geekcomix.com/tux4kids/ for downloads

and more information.

Figure 5.4 – Tux Typing

Figure 5.5 - TuxPaint

OpenOffice.org

OpenOffice.org is an open source multi-platform office productivity suite. It provides users with key desktop applications such as a word processor, spreadsheet, presentation manager, and a drawing program. It looks much like other office suites in terms of user interface as well as features.

Schools may be surprised to find how powerful OpenOffice.org is. This allows schools to accomplish the same tasks they do with a proprietary office suite, without the high licensing costs. It is available in 25 languages and runs on all major operating systems including Linux, Windows, and Mac OS X. OpenOffice.org uses an open file format and also supports a number of other formats, including the ability to read and write Microsoft Office formats. This allows for easy integration on OpenOffice.org into an existing proprietary solution.

Figure 5.6 Writer, Calc, and Impress

OpenOffice.org is easy to download and install. Visit www.openoffice.org for more information.

Chapter 6

Other Ways to use Open Source in Education

Open source is not only restricted to software. While the ideals of open source remain the most popular with software, the practical methods and beliefs of the movement are spilling over into other areas as well. It is extending into freely available information and the right to use it. Everyone from musicians to writers are embracing the ideas behind open source.

For educators, the idea of freely available information is an important one. This is the basis of public education, public libraries, and even a parents wisdom. In a digital era, knowledge can be shared without expense. Content can be delivered through computers, giving educators another option in teaching.

Many educators around the world have embraced the ideals of the open source movement. They have made specific aims to create learning materials that adhere to the same guidelines as open source software. 1) Freely available information. 2) Created from a need to fill a void. 3)The information may be used in any manner the user may seem fit. 4) This information will emphasize quality. These are the reasons why open source is not just for software. Educators may find that their beliefs about how information is provided may be along these same lines.

Textbook Problem

Textbooks have been a staple of education for many years. The have had an undeniable effect on the curriculum and lesson plans of school. The have become popular in education by having like information presented in an easily available format. They have become so influential in education that they often are the decision maker when it comes to how things are taught, learned, and what information is presented. No one can deny that they can be used in a positive manner by teachers and students using them as a resource when traversing lesson plans based on their schools curriculum. This being said, they also have serious drawbacks.

To often, textbooks are the be all and end all of education. This amounts in a loss of independence and objectivity when it comes to information. No matter how skillfully written a textbook is, the information is still coming from one place and more or less from one point of view. This point becomes clearer when we consider the ongoing controversy of teaching evolution versus creation. Textbooks almost always offer one view of history. It becomes apparent that teachers students should treat the text as a single source, rather than a definitive one. This should

seem obvious to educators, yet when textbooks are used as the sole provider of information, this is exactly what happens.

An answer to this would promlem would be to have multiple textbooks that each presented differing views. This isn't practical, textbooks are just plain expensive. Textbooks are linked to big business. Publishers would like nothing better than to lock schools into buying their textbooks for every subject. With the push towards mass purchases these companies work deals with states and districts to make choosing books and material easy. Ease is not everything. Responsible educators have existing curricula that they should be looking at to see if a textbooks fits into. The mass purchasing of textbooks undermines this and can make curriculums and standards pointless.

Another problem with textbooks is the speed in which they become obsolete. Since the cost of textbooks usually makes purchases possible every one in a blue moon, it is common to have old and outdated material in use. While certain information always stays the same, we are constantly in flux with how we understand the world and with the approach we feel best to present this information to students. The traditional textbook model does not fit this nature of teaching.

It seems that most schools are already understanding these problems. Although it does not make it any easier to choose textbooks that “kind of” fit into their needs. There are also other things to worry about other than price and obsolescence. Copyright also gets in the way. Schools are not allowed to do as they wish with the information. They can not legally make copies or change it in a way that would make the information better fit their needs. These are the issues that the open source movement aims to solve.

Textbook Fix

When educators apply open source methodology to textbooks, many problems dissolve. Generally proponents of the open source movement believe in free information. Again, not as in money but as in liberty. They believe you can not own an idea. Thus, the decision of how to use a particular piece of information falls on the user of that information.

This concept alone fixes a large issue with the textbook problem. When a text becomes open source, they can modify the contents to exactly the specification their curriculum and standards set forth. While at first site this may seem like a daunting task, attention should be paid to how open source works. Individuals are never alone in the open source community. It is filled with people who have a propensity to help others. Meaning, schools are not going to be alone when they might need help adjusting a project. Even though it may seem too large task to undertake, in the end, it chalks up to a much greater flexibility. This flexibility will greater serve the students and the school as a whole with learning material specifically tailored to their needs.

The most obvious benefit of open source, low or no cost, also helps in the textbook problem. When a school does not have to spend money on buying textbooks, they can better utilize the funds. Maybe to a more deserving place such as classroom budgets or teacher salaries. Also, since very little is being spent on textbooks, the ability to have multiple viewpoints by having different texts, becomes possible. The only limit may be the amount of texts available. This issue is only getting better with time and as more educators become aware of this method of providing information to their students.

Open source textbooks are able to provide the latest information. Part of the open source

methodology is having the content is constant revision. With the traditional textbook, a benchmark is reached and it is marked the first edition. The content stays that way until forces (usually monetary) behoove the company to revisit the material and issue the second edition. Just like software, open source textbooks allows mistakes to be fixed and new features to be added at an astonishing rate.

It should be noted that the idea of open source textbooks are not mandating that printed textbooks be replaced. Since most open source textbooks are available in digital form, people jump to the conclusion that the information will be presented digitally as well (on screen). While this is certainly a possibility that has its merits, it is not the only choice. Because the open textbooks give the freedom to reproduce, it is possible to print a copy and then reproduce that. While this may take a small amount of time and paper, the cost is heavily offset by the money saved on the textbooks. The bottom line is that the open source methodology makes producing and using textbooks less expensive.

Open Source Information Projects

Since the idea of open source started with software, it isn't surprising that the first open source texts were geared to computers. Today, the idea has caught on and many other professions are taking part. This is in contrast to the harsh realism of intellectual property, where ideas are monopolized and the rights to use them are given only by the owner.

As we have seen, this open publishing is similar to open source, except the issue is the written word instead of code. If content is published under an open license, anyone can use, change, or redistribute it as they see fit. This book is under such a license and sharing and modification is encouraged. Some educators and institutions are opening their curriculum (primarily through the Web). This subset of open publishing is called “open courseware.” A Massachusetts Institute of Technology spokesperson explains: “We are fighting the commercialization of knowledge, much in the same way that open source people are fighting the commercialization of software.” (J.P. Potts in Festa, 2002).

While open source and open courseware are decidedly separate issues, their common principles of community are very similar. Open courseware projects try to use and support open source software to serve their goal of universal access. To ensure that access, open courseware is still accessible through proprietary program (e.g. Microsoft Explorer, Adobe Acrobat). To the extent the principles overlap, open courseware may be a compelling reason to use open source software. A school may want to adhere to the singular idea of open access, to align policies on software and courseware.

There are many such projects out there and the Internet is, for obvious reasons, the best way to deliver them. A simple search on Google will find results very quickly. The rest of this chapter presents a few of these projects and should give you a general idea of what is out there. The descriptions used, are from their websites, as they know their projects best.

MIT's OpenCourseWare (MIT, 2003)
http://ocw.mit.edu/OcwWeb/index.htm

With the publication of 500 courses, MIT OpenCourseWare (OCW) offers educational materials from 33 academic disciplines and all five of MIT's schools.

The idea behind MIT OpenCourseWare is to make MIT course materials that are used in the teaching of almost all undergraduate and graduate subjects available on the Web, free of charge, to any user anywhere in the world. MIT OCW will advance technology-enhanced education at MIT, and will serve as a model for university dissemination of knowledge in the Internet age. This venture continues the tradition at MIT, and in American higher education, of open dissemination of educational material, philosophy, and modes of thought, and will help lead to fundamental changes in the way colleges and universities utilize the Web as a vehicle for education.

In 1999, MIT Provost Robert A. Brown asked the MIT Council on Education Technology to provide strategic guidance on how MIT should position itself in the distance/e-learning environment. The resulting recommendation – the idea of MIT OCW – is in line with MIT's mission (*to advance knowledge and educate students in science, technology, and other areas of scholarship that will best serve the nation and the world in the 21st century*) and is true to MIT's values of excellence, innovation, and leadership. MIT OCW provides a new model for the dissemination of knowledge and collaboration among scholars around the world, and contributes to the "shared intellectual commons" in academia, which fosters collaboration across MIT and among other scholars.

MIT's OCW is a publication of MIT course material. It doesn't require any registration and is perfect for the self motivated learner. To learn more please visit ocw.mit.edu/index.html.

California Open Source Textbook Project (COSTP, 2002)
http://www.opensourcetext.org/

The California Open Source Textbook Project (COSTP) is a collaborative, public/private undertaking. It has been created to address the high cost, content range, and consistent shortages of K-12 textbooks in California.

California currently spends more than $400M annually – and rising – for K-12 textbooks. With K-12 enrollments projected to rise in the coming years, revenue demands for textbooks and other curriculum materials in California will increase proportionately.

COSTP will employ the advantages of open sourced content and innovative licensing tools to significantly reduce California's K-12 textbook costs – eventually turning K-12 curriculum and textbook construction from a cost into a revenue generator for the State of California.

Three – amongst many – COSTP benefits will be 1) the complete elimination of the current $400M+ line item for California's K-12 textbooks; 2) a significant increase in the range of content afforded to California's K-12 textbooks: 3) a permanent end to California's textbook shortages: and 4) creation of fully portable content holdings database that scales with classroom technologies as they are introduced.

It is important to note that COSTP's *mandate does not replace printed textbooks;* it simply makes them less expensive to produce; and, in doing so creates many additional benefits, economies, and efficiencies that will fully leverage California's activities in the K-12 textbook

publishing domain.

The cost of K-12 textbooks has risen steadily over the years. Whatever the reasons for increasing costs, it seems likely that today's high K-12 textbook prices are not inevitable. The past history of textbook prices, and the existence, even today, of textbooks that occasionally cost significantly less than average, indicates that textbooks *could* be produced and sold for 33-50+% less than currently charged by textbook publishing companies. COSTP's goal is to make the latter pricing scenario a reality, while providing multiple additional advantages for California's K-12 schools, teachers, students, and taxpayers.

Open sourced, distributed content can be made available gratis, and/or through the use of innovative copyright tools provided by organizations like Creative Commons, its partners, and other organizations.

Thus, COSTP will provide a new model for textbook creation in the State of California by 1) leveraging free, already-existing, and widely available K-12 educational content in the public domain; 2) better leveraging the substantial curriculum-based intellectual capital of California's best K-12 teachers; and 3) using innovation copyright tools to secure new and dormant K-12 textbook content that would not otherwise be made available.

COSTP is projected to augment current K-12 textbook supply chain, be self-supporting with 18 months of starting up, and save the State of California upwards of $200M+ per years for K-12 textbook allocation within five years.

In phase two of the CASTP plan (following year five), California will be able to offer (by license, at nominal cost) K-12 textbook and curriculum materials to other educational organizations and international agencies. This will create substantial cost savings for those entities, and will result in the complete elimination of all funds heretofore budgeted for California's K-12 textbook purchases – currently $400M+ per year. Additionally, California will realize a surplus (profit) from it's K-12 content licensing activities as this second stage of COSTP is deployed.

O'Reilly Open Books Project (O'Reilly, 2004)
http://www.oreilly.com/openbook/

Over the years, O'Reilly and Associates has published a number of “Open Books” -- books with various forms of “open” copyright. The reasons for “opening” copyright, as well as the specific license agreements under which they are opened, are as varied as their authors.

Perhaps a book was outdated enough to be put out of print, yet some people still had an urgent need for the information it covered. Perhaps the author or subject of the book felt strongly that books should be published under a particular open copyright. Or maybe the book was written collectively by a particular community.

O'Reilly holds the copyright to some of its books; individual authors hold the copyright to others. Regardless of who holds the copyright, the decision to apply an open copyright, and the choice of license, is left to the author or authors of the work. Some of the licenses include the GNU Free Documentation License, the Open Publication License, and the GHU General Public License.

Open Book Project (Elkner, 2003)
http://www.ibiblio.org/obp/

The Open Book Project is aimed at the educational community and seeks to encourage and coordinate collaboration among students and teachers for the development of high quality, freely distributable textbooks and educational material on a wide range of topics. The advent of the Internet and the World Wide Web are making collaboration among educators on a global scale possible for the first time. We want to harness this exciting technology to promote learning and sharing.

The Open Book Project publishes books licensed under that Open Publication License. One of the titles released so far is *How to Think Like a Computer Scientist*. This book is a perfect example of open source publishing in action. Originally by Allen B. Downey, it has been hacked on by many people and now has four versions each covering a different programming language (Java, Python, Logo, and C++).

Wikipedia (Wikipedia, 2004)
http://en.wikipedia.org/wiki/Main_Page

Wikipedia is a wiki-based free content encyclopedia with supporting almanac-like and gazetteer-like information. Free means both free to use and free to edit. Wikipedia is a multilingual and an open content collaboratively developed creation managed and operated by the non-profit Wikimedia Foundation. As of February 2004, it contains over 200,000 articles in English, and over 250,000 articles in other languages.

Wikibooks (Wikibooks, 2004)

Wikibooks is a project for collaboratively writing textbooks and related non-fiction books about different subjects. It is a sister project to Wikipedia. The goal is to create a free instructional resource – indeed, the largest instructional resource in history, both in terms of breadth and depth and also to become a reliable resource.

Part III

Using Open Source in Education

Chapter 7 – Implementing an Open Source Solution

Chapter 8 – Open Source in the Real World

The success of any technology program lies in its planning. When enough quality time is spent observing your situation, the actual implementation is going to go a lot smoother than otherwise. Good planning occurs when principles of good technology planning are balanced with the practical realities that face education. Table 6.1 shows some of these principles and realities. They are neither complete nor comprehensive.

Some Principles of Good Technology Planning	Some Realities of Technology in K-12 Education
Goals should be based around student achievement and effective learning environments. Stakeholders should have ownership. Deployment is only the beginning. Be flexible. Effectiveness should be evaluated. Update and revise regularly.	Technology can only be part of the solution. Change is good and bad. The sale price is not the total cost. Schools can leverage the experience and models of others. Technology is a social, legal, and ethical issue.

Table 6.1

Principles of Good Technology Explained

Goals should be based around student achievement and effective learning environments.

With good planning and implementation, technology can support learning, assessment, administration, and/or communication. Technology should not be deployed for its own sake. A good tech plan is formal, clear, and broad. The plan should fit with the needs, goals, and capabilities of the school (e.g. improve reading skills, use the Internet to enrich curricula).

Using existing resources and expertise are important for a good technology plan. Schools need to reuse and expand existing solutions to build capacity over time, rather than discard

everything and start from scratch. Such capacity includes hardware, software, integrated systems, and the skills and enthusiasm of staff. Open source will require new expertise, and some solutions include significantly different architecture. (e.g. A school may want to replace a lab of stand-alone desktop computers running Microsoft Windows with networked Linux thin clients.) The attractive advantages of open source must be tempered with wise spending.

Some schools already pay for more technology than they need. Many users probably don't use the numerous features of programs like Microsoft Word. The open source community offers a variety of free solutions for Web browsing, email, and basic productivity. Many open source solutions run on Microsoft Windows or Apple Macintosh. With either proprietary or open source software schools should teach thinking and technology literacy, not dependence on specific programs or brands.

Points to Ponder	Goals, not brands or boxes, should be paramount.

Questions	What technology do you actually need to support learning goals? Are you trying to support advanced curriculum in technology?

Stakeholders should have ownership.

A good tech plan can only be created and implemented with stakeholder involvement. Stakeholders may include administrators, instructors, other staff, students, parents/guardians, community members, and technology experts. Only a group can accurately answer questions like, "Where are we now?" and "Where do we want to go?" Decisions should be made as a group to foster buy-in and to erode resistance to change. Stakeholders who feel included and empowered, who feel their needs and goals are being addressed, are essential for effective implementation.

Most stakeholders will accept short-term hardship for long-term gain, but only if they believe it's only short-term and the goals are worthwhile. Some stakeholders will always be "ahead" of the plan by integrating technology on their own. Praise their initiative and invite their expertise.

Most stakeholders probably aren't familiar with all the aspects of their school's technology, much less possible open source solutions. Stakeholders need to feel some mastery of the options without being overwhelmed by technical details or complex controversy. Current users recommend downplaying the "revolutionary" mood of the open source movement, and instead focus on the cost savings of useful solutions.

Many schools are already using open source on the backend. As stakeholders examine their current technology and expertise they may find unexpected existing capacity. For example, a district network technician who maintains open source email and Web servers may be an asset in

migrating to Linux thin clients in schools.

Points to Ponder	Stakeholders are rightfully resistant to dictatorial mandates. Communicate with stakeholders early and often. Planning should be an open process.

Questions	Are some educators already using open source, at work or at home? Can you convene a tech planning committee with diverse, open-minded stakeholder?

Deployment is only the beginning.

Integration requires more than deploying hardware and software. Leadership and communication are essential throughout the process. Users and other stakeholders need to understand why change is occurring and how hardships will be mitigated. Set time lines and celebrate milestones.

Training, acclimation, and integration take extra time and other resources. With open source or proprietary solutions, the critical factor is support. Technology without sufficient support may be worse than no technology. Too many schools are filled with unused technology because it's not well supported or users don't have enough training and comfort.

Points to Ponder	Technology without training and user comfort is wasteful and may be worse than no technology at all. Think about replacement because it won't last forever (even without Murphy's Law).

Questions	Can you afford training? Can you maintain or build capacity for support? Will your needs and goals compel stakeholders to endure hardships and acclimate to new solutions? Can you risk depending on the open source community for help or do you need more certainty?

Be flexible.

Flexibility is valuable when managing present and future change. Where possible, schools should choose open formats, modular solutions, and commodity software. If a school becomes dependent on a specific, inflexible proprietary solution, change can be forced on the school by software

companies.

To mitigate the hardships of change, schools should look to low threshold solutions. Changes to servers may be invisible to most users. It may be easier to experiment with one computer lab than with the computers in a dozen classrooms. Current users recommend having at least one test machine for trying any new software, open source or proprietary, before depending on a new solution.

Schools may need to offer some backward support after any migration. Some users or data may be locked into a specific solution. Schools shouldn't abandon good teachers or content.

In some cases open source offers low threshold solutions for easy experimentation or gradual change. For example, OpenOffice.org can run on Microsoft Windows and supports Microsoft Office formats, so users can experiment when possible and still use Microsoft Office when necessary. Educators may discover that open source software like OpenOffice.org is more than adequate for most needs and goals (e.g. students working in a writing lab).

Open source offers some independence which may be attractive for those stakeholders who want more empowerment. Advanced users may enjoy the customization and user community of open source. The philosophical principles may be intriguing and attractive to stakeholders, but these will probably be minor criteria in the planning process. However, while identifying needs and goals stakeholders may express progressive education goals and/or bemoan pressure from proprietary companies (regarding licenses or upgrades). Then the principles are very relevant.

Points to Ponder	Lock in, forced upgrades, and end-of-life are real hazards.

Effectiveness should be evaluated.

Technology planning should include evaluation since stakeholders desire and benefit from knowing the effectiveness of implementation. Effectiveness (or ROI) can be evaluated short term or long term. Long-term effectiveness matters more, especially for building capacity. But if the short term is too challenging or unrewarding the long term may be virtually inaccessible. So low threshold solutions may be essential to sustain long term change.

Evaluation should include broad criteria since schools want to integrate technology across classrooms and teachers. However, at least some criteria should be easily accessible so that stakeholders can enjoy progress and accomplishment in the near future. Just increasing the level of dialog and intention about technology is a worthy activity. The complete realities of total cost of ownership should be evaluated, not just sale prices.

When considering or evaluating open source software the plan should focus on the effectiveness of specific solutions. Some open source software isn't worth the trouble, while other solutions may have a low threshold and/or long-term advantages. Schools should focus on what works and avoid discarding or foreclosing on any effective solution, proprietary or open source.

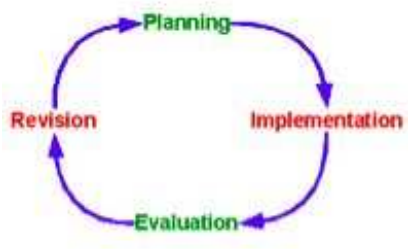
Points to Ponder	The criteria for evaluation should reflect the needs and goals already identified in the plan.

Questions	Do your needs or goals demand a specific, proprietary solution? Do your stakeholders crave some short-term success in technology integration? Can you incubate an open source solution long enough to evaluate its actual effectiveness?

Update and revise regularly.

Schools have limited resources so the plan should include cyclical planning, implementation, evaluation, and revision. It may be unwise to migrate to an open source solution now because of pressing needs or inadequate open source options. But every aspect of every solution should be periodically revisited (even if only to conclude "it works and it's worth the expense").

When reviewing the plan, stakeholders may re-prioritize their needs and goals. Schools may want to expand, remove, or replace existing solutions. Open source may be the best choice during such changes (e.g. replacing a lab). Open source allows schools to experiment at little cost, and specific programs may mature significantly over time. As schools phase in new technology, older hardware may still be useful by deploying open source. For example, older desktop computers may be recycled as thin clients on a K12LTSP network. (Some current users only buy new Microsoft Windows computers with the kind of network cards that enable this future option.)



Questions	Are you willing to re-plan and use data to make changes? Are you willing and able to consider and experiment with several solutions? Can you build or shift capacity over time?

Realities of Technology in K-12 Education Explained

Technology can only be part of the solution.

Technology should solve more problems than it creates. Schools have limited resources so technology may not always be the best expenditure. For example, time and other resources might

be better spent on reducing class size, increasing course offerings, or supporting extra curriculars. The savings from open source may provide more resources for these other worthy goals, but any savings won't be certain until after migration. Resources should be spent pragmatically with a clear sense of what technology can accomplish and how the tech plan fits in the larger vision for the school.

Points to Ponder	The plan should clarify priorities because you can't "do it all."

Change is good and bad.

Good tech plans address change as both desirable and problematic. Change is inevitable because technology has a life cycle from affordable to obsolete. Continuous improvement is desirable and schools should be learning organizations. At the same time schools already have certain technology and staff, and this status quo strongly defines what's possible. Schools face numerous other challenges and concerns, with limited time and other resources. As an organization, a school needs to prioritize change to avoid too much general disruption or impact on morale. The school should balance any major changes to technology with other changes, and only try to make major changes during relatively calm periods in the calendar (e.g. summer).

Opportunity costs are a hazard of change. By choosing a popular solution (e.g. Microsoft Windows), schools minimize the risk of incompatibility with future programs. Schools should look to their needs and goals when considering whether the other advantages of open source are sufficient to warrant the risks.

A good tech plan addresses the need for training and support, especially during and soon after any major changes. Given the hectic daily life of most educators, it's better to err on the side of excess support, if possible. The open source community may be a valuable asset for supporting stakeholders.

Points to Ponder	Churn and burnout are real hazards.

Questions	Have you recently made major changes in technology? Will you be making major changes soon? Are you able and willing to experiment?

The sale price is not the total cost.

There are shrewd ways to spend money and other resources to integrate technology. Some

companies offer comprehensive packages and such "single cost" solutions may be attractive. If proprietary software companies continue to move toward subscription licensing, schools will face a choice between leasing solutions or building their own capacity. Schools should look for ways to minimize purchase costs using existing expertise, infrastructure, and other resources. For example, older computers may not need to be upgraded if they can still meet some needs. The cost of retraining support staff and users may not be worth the advantages of newer solutions.

Open source has immediate, visible cost savings since it's free to install. These savings may initially compel stakeholders to consider open source solutions, but the discussion needs to expand to total cost of ownership (TCO). Even where open source may reduce TCO, it won't eliminate all costs. Open source is not "magic pixie dust." Migration may be very expensive. However, schools may want more independence and flexibility. With open source, schools can invest in capacity rather than software licenses.

Points to Ponder	Try to choose the best solutions because they may not be replaced for years.

Questions	Would you rather buy certainty or invest in capacity? Are you struggling with budget limitations?

Schools can leverage the experience and models of others.

Schools shouldn't start from scratch. Similar problems have similar solutions, so schools can learn from current users and adapt existing models and resources. The best solutions are usually found elsewhere in K-12, since schools have different needs, goals, and resources than business or government. For examples, schools can depend more on volunteers and most licenses favor nonprofits like schools.

One of the most attractive advantages of open source is the user community, especially the thriving subpopulation of educators. The community offers powerful free support through email lists and Web sites. Current users generally depend on this community for learning and troubleshooting, and commend the speed, tone, and effectiveness of the help. The nature of open source means users can offer more help to each other, including customized distributions (e.g. K12LTSP) and sophisticated scripts and tools. If a school does migrate to open source, some stakeholders will take advantage of the community. Most users will depend on staff in the building.

Open source fosters experimentation and customization, so schools can pick and choose. And while robust solutions like Linux, Apache, and OpenOffice.org are created for business and government, schools can enjoy the windfall of such software's free, unlimited distribution and thriving user community. Of course, some staff will be more or less willing or able to participate in the community and ask for help. However, by deploying customized, modular solutions and training (or retraining) staff, schools invest in their own capacity rather than just the success of software companies. (This is a significant reason why some countries are choosing open source.)

Questions	Are you willing and able to search for solutions rather than buy them? Will your technology staff be willing and able to get help from the community? Will your advanced users be willing and able to help other users? If you lose an important staff member, will you have a backup? Or does your technology infrastructure stop functioning?

Technology is a social, legal, and ethical issue.

Technology includes far more than hardware and software. Stakeholders may express some social, legal, or ethical concerns related to technology. Such concerns may include: equity and the "digital divide," intellectual property in the digital age, inappropriate content or behavior, virtual monopolies, and balancing technology with other priorities in schools. Technology planning and technology literacy should address these larger issues for all stakeholders, especially students.

Open source offers unique answers to some of these concerns. It may address equity by making infrastructure more affordable or by allowing schools to send free software home with students. The open source community is thoughtful and outspoken in the debates about intellectual property. Free speech and respect are contiguous to the open source philosophy, while open source solutions challenge proprietary models of content filtering and behavior control. (e.g. Open source users can personally control which sites students can access, rather than depending on a proprietary service.)

The implications of these larger issues may be important criteria in the planning and evaluation process. But schools often make decisions individually and without deliberately confronting these issues. Some current users are the exception, and want to engage these issues whenever it doesn't interfere with other, more immediate goals (e.g. supporting effective learning environments).

Points to Ponder	Schools serve an important role in educating students on these social, ethical, and legal issues.

Questions	Are you trying to foster more awareness of the larger issues in technology?

Other Salient Points

Now that we have looked at some of the issues regarding a good technology plan, there are some other points we should look at. These points should be taken into account when deciding how to implement open source into your individual situation.

Schools are unique

Most of the research and discussion about open source is focused on business, especially large corporations and their needs. Government also receives some attention, although the United States has been relatively slow to consider open source. Schools have unique needs, goals, and resources. It's not always easy or valid to judge the value of open source software using business or government evaluations. States or large districts may have the needs and the technology literacy to study solutions on the enterprise/ERP/CRM scale. But most schools need to evaluate open source on a smaller scale.

Points to Ponder	Own your technology decisions. You already get the consequences.

Fortunately, schools have resources business and government lack. The open source community is eager to help schools, and many LUGs (Linux User Groups) are helping educators experiment and migrate. The community includes a thriving subpopulation of educators. They are eager to welcome other schools and build their collective capacity.

Schools may also want to involve students. Students can build computers with inexpensive or recycled hardware and open source software, learning valuable skills and understanding while creating technology they can use. Students may help deploy and manage solutions, especially in high school. In all such instances student work should be an educational experience, not just "slave labor."

The Open Source Initiative certifies true open source licenses. Such licenses usually favor schools. Schools should be familiar with the OSI certification and look for it. Most license stipulations are unimportant for most schools. Some licenses prevent future commercial use. If a school is creating marketable learning objects, they may need to be wary of such licenses (e.g. GPL). This isn't an issue if the objects will be open courseware, which most products created by schools are.

Talk with stakeholders

Schools need to keep stakeholders informed and involved. Some educators may use open source already. Others may be dismayed by any foreclosure on possible savings or other advantages. Business journals are advising corporate leaders to have open source plans. Even if the plan is watch and wait, every school should be discussing open source with interested or affected stakeholders.

Is Open Source right for you?

The following checklist was created by the Northwest Educational Technology Consortium as a quick and simple way to see if open source might be suitable to your situation. Keep it mind it was designed for initial observations only and isn't suitable for making concrete decisions.

+1	-1	Needs and Goals
		What technology do you actually need to support learning goals? General technology: +1, Specific programs: -1 Many specific programs aren't Open Source and aren't compatible. Schools may have learning goals that require a specific program.
		Do your needs or goals demand a specific, proprietary solution? No: +1, Yes: -1 Open Source can meet many general needs and goals, but it's incompatible with many specific programs for school.
		Are you trying to support advanced curricula in technology? Yes: +1, No: -1 Open Source is excellent for advanced curricula, especially programming and Web development.
		Are you trying to foster awareness of the larger issues in technology? Yes: +1, No; -1 The debate about Open Source includes many of the larger issues, including intellectual property, free markets, and privacy.

+1	-1	Capacity
		Are you willing and able to search for solutions rather buy them? Yes: +1, No: -1 It may not even be possible to buy certain Open Source solutions.
		Would you rather buy certainty or invest in capacity? Capacity: +1, Certainty: -1 Current users report greater empowerment and self-reliance. But certainty may be worth the cost for some, especially for critical solutions (e.g. finance).
		If you lose an important staff member, will you have a backup? Or does your technology infrastructure stop functioning? Backup: +1, Stop Functioning: -1 Avoid too much dependence on any single staff member for any technology. A clever-but-inscrutable solution is risky.
+1	**-1**	**Leadership and Planning**

		Can you convene a tech planning committee with diverse, open-minded stakeholders? Yes: +1, No: -1 Open Source is complex and controversial. People who understand it are needed to succeed.
		Are you willing and able to consider and experiment with several solutions? Yes: +1, No: -1 Current users recommend experimenting and migrating incrementally.
		Can you incubate an Open Source solution long enough to evaluate its actual effectiveness? Yes: +1, No: -1 Long-term results may be worth the cost and stress of migration and adjustment.
		Are you willing to re-plan and use data to make changes? Yes: +1, No: -1 Some solutions may work while others are too problematic.
		Can you build or shift capacity over time? Yes: +1, No: -1 Migration may be expensive, but over time staff and other users may develop comfort and capacity for long-term gains.

+1	-1	Support
		Can you maintain or build capacity for support? Yes: +1, No: -1 With Open Source, users may need a lot of initial support, because they have more mind share and comfort with proprietary, closed source software.
		Will your technology staff be willing and able to get help from community? Yes: +1, No: -1 The community in one of the greatest potential advantages of Open Source.
		Can you risk depending on the Open Source community for help or do you need more certainty? Risk: +1, Certainty: -1 It may not be easy to find answers. The community is not contractually bound to help. Though better support may be provided though the community.
		Can you afford training? Yes: +1, No: -1 Technology staff may need training, especially on the backend. Some users will benefit from training, including "informal" in service training.
		Will your advanced users be willing and able to help other users? Yes: +1, No: -1 Schools can emulate the larger Open Source community of mutual support and enjoy some of the same benefits more locally.

		Are some educators already using Open Source, at work or at home? Yes: +1, No: -1 These current users can be invaluable, as with any technology. Praise their initiative and invite their expertise.

+1	-1	Change and Morale
		Have you recently made major changes in technology? No: +1, Yes: -1 Change is disruptive. Another big change may cause more problems. Consider Open Source in the future.
		Will you be making major changes soon? Yes: +1, No: -1 Any new software requires training and adjustment, so it may be easier to try Open Source during major changes.
		Are you struggling with budget limitations? Yes: +1, No: -1 Open Source may be able to save money.
		Are you able and willing to experiment? Yes: +1, No: -1 Open Source may be risky, but the long-term advantages may be worthwhile.
		Do your stakeholders crave some short-term success in technology integration? Yes: +1, No: -1 Programs like OpenOffice.org and the GIMP may foster improved integration, especially since teachers and students can take the software can take the software home.
		Will your needs and goals compel stakeholders to endure hardships and adjust to new solutions? Yes: +1, No: -1 Current users say that stakeholders are more accepting of Open Source when they understand the potential advantages, especially saving money.
		TOTAL =

Add up the two columns then subtract the left from the right.

Example:

17	**7**	**TOTAL = 10**

A large, positive total means open source may be right for you. These numbers only

illustrate the relative impact of choosing open source. They are not a definitive measure. The actual impact of some questions will vary from school to school. Answers may be different on the backend (e.g. Web Servers) versus the frontend (e.g. desktop computers).

After Deciding Open Source is Right For You

Join the open source community

The open source community is welcoming, lively, and helpful. People skills are important to joining the community: humility, patience, a cool temper, and a sense of humor about technology. Educators need to be able to describe their problems clearly, and sometimes they have to learn enough to make sense of the answers.

Points to Ponder	Asking questions reveals your ignorance. But it's necessary to learn the answers to better support your stakeholders.

A community of mutual aid is very different than a vendor-client relationship. The community thrives on the initiative and creativity of programmers. Members respect hard work and ingenuity, and vocally despise lazy, stupid, or ungrateful people. Current users explain that many open source proponents want to help schools and the subpopulation of educators is growing. So it can be easy and rewarding to find help. The community is more than an ideal: most members strive to "share and share alike" and "pay it forward."

Some other tips:

- Share information. Every person brings a unique perspective, so you may know of a Web site, article, or solution that other people are interested in.
- Let the community know who you are and what you're doing. Any use of open source in schools is exciting to the community.
- Say "thank you." You're getting something for nothing.
- Use meaningful subject lines. Advanced users may delete messages just based on the subject.
- Many people in the community go home for the weekend. Questions on Friday might not be answered until Monday.
- Laud the help you get. The community even has formal systems for gratitude and building prestige. (See: http://www.affero.com/ and http://www.linuxfund.org/)
- Programmers like interesting and challenging problems.
- If your problem isn't interesting or challenging, explain why it's important (e.g. improving the education of children).
- An answer may be too technical. Try to demonstrate what you

do understand to get a clearer explanation.
• You may have to study. The community answers intelligent questions voluntarily but can't always shelter novices.
• An answer may be condescending (e.g. "read the f***ing manual"). Try to find email lists populated by educators and other sensible, helpful people.
• Ignore stupid and/or angry people. Remember your goals and pick your battles.

Choose a distribution carefully

Open source eliminates most license concerns, but software management still matters. Most open source software is available in several progressive versions. Furthermore, each Linux distribution has unique features. The most current version of a distribution (or "distro" or flavor) will be the most feature rich, but a slightly older version will probably be more stable. Schools should try to standardize on a single distro and version, and keep track of what's on each computer. Schools should change or upgrade all copies of the software at the same time, if possible. (Thin client servers make all this very easy.)

Educators should talk to other educators when considering open source software, and especially when considering a Linux distribution. The community favors certain programs and distros. Since community help can be important to a lower TCO, the community's preferences are viable criteria for decision making. If many people use a specific program or distro it will be easier to get help.

No matter what Linux distribution you decide to use, you should understand that the different types are more alike than they are different. Often the differences are of a cosmetic nature and don't have much to do with the way the programs operate. One way to understand these differences is to think of car companies. Ford and Dodge make different cars, but that does not mean that you won't be able to drive one if you can drive the other.

Red Hat is commonly referred to as the most popular. While Mandrake has a reputation of being user friendly. Debian is incredibly stable, though not on the cutting edge. Red Hat is also leading the industry by providing a certification program for technicians, which may make it easier to hire and train staff.

When a deciding which flavor to go with, schools should do as much research as possible. The Internet has a large amount of places to see what may fit your needs. Table 7-1 presents some major distributions and websites where you can find more information about choosing a distribution.

Red Hat www.redhat.com	MandrakeSoft www.linux-mandrake.com
Debian www.debian.org	Slackware www.slackware.com

Red Hat www.redhat.com	MandrakeSoft www.linux-mandrake.com
Lycoris www.lycoris.com	Caldera OpenLinux www.calderasystems.com
SuSE www.suse.com	Gentoo Linux www.gentoo.org
Great ISO Download Site www.linuxiso.org	Just Linux www.justlinux.com

Table 7-1

Jumping In

The decision to use open source does not have to be either or. In fact, most of the time it would be an unwise decision to jump totally into using open source software. This would probably lead to a failing program because schools usually do not have the resources available to provide the support needed for such a big jump.

It is not necessary to switch to Linux to benefit from open source. There are many open source projects that run on proprietary operating systems such as Windows and OS X. There are versions of OpenOffice.org, the Gimp, and Mozilla that run on all major operating systems. There are a thousands of other open source projects available that fit this description and may be able to help schools.

An easy first step to implementing open source is to find a replacement for one program and see how it works. One of the best replacements would be MS Office with OpenOffice.org. This is a small step that brings the benefits of an open source application. The money that can be saved on licensing this one product alone, will most likely be noticeable in the budget. OpenOffice.org opens and writes MS Office formats so working between programs during switch should see little problems. After schools have successfully implemented OpenOffice.org, they can then make baby steps further into using open source software.

Another option to allow schools to step slowly into the open source movement would be using both a proprietary and open source operating system on the same computer. By using multiple hard drive partitions it becomes possible to install as many operating systems on the computer as needed. A school that wants to use Linux, but have certain programs available only to Windows or Mac OS X, can install both operating systems and get the best of both worlds. This process is usually referred to as “dual booting” because most people only need to install two operating systems on a computer. Most popular Linux distributions have tools that make this process relatively simple. The documentation of each distro should also provide step by step directions on this process.

Another option that is similar to dual booting is to install Linux on the computer and use open source projects like Wine and Mac-on-Linux to run the programs you need. Wine is a

program which allows the operation of DOS and MS Windows programs on Unix operating systems such as Linux. This process allows a school to run Windows programs without having to have a license to the Windows operating system. It also saves the user from having to reboot the computer into another operating system, a process that takes time and make work less efficient.

Mac-on-Linux (MOL) is very similar to Wine except for it works for PowerPC based computers (Apple). MOL allows an Apple computer running Linux to boot into Mac OS or Mac OS X at the same time. By having multiple operating systems running at the same time, users have the ability to use any program they might need.

Read Some of the Literature

The best way to understand the open source movement is to read some of the great literature available on the topic. The Internet is the obvious place to get information, but there are many books available one the subject. The Appendix includes a list of books and other sources of information that may be meaningful to educators.

Chapter 8

Open Source in the Real World

So far we have talked about what open source is and what it can do for education. This should give you a foundation for using it in your particular situation. But we have yet to see how open source is being used in the real world. When we see how others have used open source, we may better understand how it can fit into our schools.

Before we look at how open source is being used in education we should see where it has had its greatest impact so far. Open source continues to grow in popularity among businesses and governments. This acceptance and success does not necessarily mean open source is a good choice for schools. However, these results may illustrate the maturity and potential of open source software.

Business

As businesses have grown more dependent on the Internet they've used more open source software. As Linux and other powerful solutions mature, major corporations are eager to explore the potential advantages, especially cost.

Each week brings new examples of deployment and alleged savings. For example, IBM will be supplying thousands of Linux-based cash registers to Regal Entertainment, the American movie theater company. (Economist, 2002) Verizon, the telecom company, says they "saved $6 million in equipment costs by moving its programmers to Linux computers." (Shankland, 2002, Verizon) Dell and HP-Compaq have been offering open source solutions for years to businesses and individuals. Sony's new line of Cocoon Internet/multimedia devices run on Linux.

Some financial companies have been especially vocal about joining the open source movement. For example, Merrill Lynch is doing "a top-down... large-scale Linux deployment in an effort to cut their costs and boost revenue." (DiCarlo, 2002) Morgan Stanley, Credit Suisse First Boston, and the Goldman Sachs Group are deploying open source solutions.

E-Trade and Amazon.com have chosen open source. Amazon.com allegedly saved $17 million in technology and telecommunications costs in one quarter, due in part to their migration to Red Hat Linux. (Adelson, 2002) Hollywood special effects powerhouses have used open source for years. Pixar (*Monsters Inc.*), Weta Digital (*The Lord of the Rings*), and Industrial Light and Magic (*Star Wars: Episode II*) use Linux and other open source solutions. (Bruce Perens, a leader

of the Open Source Initiative, is a former employee of Pixar.)

Government

Open source software has become a visible issue in many governments. For example, Germany is deploying Linux and the open source model at the federal, state and community levels. Secretary Siegmar Mosdorf argues that open source is more secure than proprietary software. (Gillespie, 2000) Minister Otto Schilly explains that Germany is seeking independence from any one vendor, while "creating more diversity in the computer field." (BBC, 2002, IBM)

In Peru, Congressman Edgar Villanueva introduced a bill that would require open source solutions for all government computer systems. While Villanueva argues from principle, he also points to cost. Many schools in Peru are so poor they cannot afford to pay their water and energy bills. Villanueva explains, "They are so poor that they could not afford these services, and the provision was being interrupted.... Does it make sense in this context to use tax-payer's money to pay Microsoft licenses?" (d'Empaire, 2002)

Venezuela has announced a policy that requires open source solutions for all government computer systems. The policy also requires that "all software developed for the government must be licensed under the GPL." (Proffitt, 2002) According to Minister Pérez-Martí, "the government and the people of Venezuela were increasingly concerned that over 75 percent of the funds for software licenses went to foreign nations, 20 percent to foreign support agencies, and only 5 percent to Venezuelan programmers." The new policy will try to keep more money and expertise in the country.

Mexico is also turning to open source to save money and increase national independence. Open source means "they're not just sending their money to the United States all the time," explains Gary Chapman. (Scheeres, 2001) Mexico has been criticized for poorly implementing open source solutions but is continuing to try to save money for social programs.

China is promoting Linux and other open source solutions in its government and to its citizens. The government favors Chinese-developed open source software, like the Chinese Academy of Science's Red Flag Linux. Like other countries, China wants to build its computing infrastructure and economy with domestic spending and expertise. In contrast to typical open source motives, the communist government may also prefer open source because they can better monitor and control their citizens' computing. (BBC, 2002, Linux)

Singapore and Australia are also exploring open source policies and solutions. As in business or education, decision makers in government sometimes discover they're already using open source on the backend.

The United States government is still largely focused on proprietary solutions. At the federal level, the use of open source is limited and rare. As one IT official explained, "Linux is not on our list of approved operating systems... That generally dictates whether it's used or not." (McCullagh & Zarate, 2002) Open source is used at the U.S. Department of Agriculture, the Federal Aviation Administration, the U.S. Department of Energy, the U.S. Air Force, and Pinellas County, Florida. (Galli, 2003)

Education

Seeing how businesses and governments have used open source can shed light on what effects it can have on education. Even though schools are very different, open source can provide the same effects; saving money and decrease dependence on a single company. Let's look at specific examples of how some schools are using open source.

School	Dresden Elementary. Dresden, Maine
Grade Range	K-5
Number of Students	120
Number of Teachers	9
Tech. Coordinator	Classroom teacher stipend position

Dresden Elementary is a small school in Maine. The Technology Coordinator is also the full time first grade teacher. It is a small stipend position and the amount of time she can give to technology related issues is small.

Since it is a small school, the amount of money allotted to spend on software licenses is very small. Recently, the school used grant money made available from the No Child Left Behind Act to purchase a laptop for all teachers. This has allowed teachers to be more productive and has been an overall success with the teachers. However, the teachers have been asking for Microsoft Office. The school lacks the funds to provide this for every teacher. The technology coordinator is currently looking at OpenOffice.org to fill this gap.

School	Vassalboro Community School. Vassalboro, Maine
Grade Range	K-8
Number of Students	550
Number of Teachers	60+
Tech. Coordinator	Full-time.
Related URL	http://www.vcs.u52.k12.me.us/linux

Vassalboro Community School is a front-runner with the use of open source in Maine. Since 2001, the school had been using K12LTSP (K12 Linux Terminal Server Project) to lower cost of licensing and raise ease of use and administration. This has allowed the school to greatly increase access to technology and deploy mini-labs in nearly every classroom.

The deployment of computers running K12LTSP has also lowered the cost on hardware as well as software. K12LTSP runs by using thin-clients (computers with no hard drives) that do not need high specs and runs well on old donated machines. The main cost for the school has been a robust server that is powerful enough to handle to load of the thin-clients.

School	Beechwood Schools Ltd. Streatham, South London (UK)
Grade Range	Primary and Secondary
Number of Students	200
Number of Teachers	20

Tech. Coordinator	NA
Related URL	http://www.beechwoodschool.com

The Beechwood school was in the market for reliable servers. They decided that the Microsoft route was too expensive, was not reliable, had no real support, and was not flexible enough for their needs. The school tried Windows 98 on client machines but had trouble with security. They tried to improve this with Windows NT but still had trouble with students installing programs that messed up machines which then required reinstallation. Viruses were also running rampant. They also moved to ADSL broadband and required a more complex firewall system.

The school is now running Linux based servers for file sharing (NFS and Samba) and a bastion/choke firewall. The total cost was on hardware only and totals to $3000. This cost included switches, cabling, the server itself, and a couple of odds and ends. No money was spend on software licensing. The school is now in the process of moving all of its client computers to Linux. Every new computers that the school buys, have to meet compatibility requirement for Red Hat Linux. They are also using OpenOffice.org as an office suite at all levels of the school (student, teacher, administration).

School	Blue Ridge School. Jackson County, North Carolina
Grade Range	K-12
Number of Students	290
Number of Teachers	30
Tech. Coordinator	Full-time
Related URL	http://www.blueridgeschoolpk12.com

The teacher and administration at Blue Ridge School wanted a total information system whereby teachers could post grades, homework assignments, class information, attendance, and a variety of other forms of information on the Internet. Thereby allowing parents and students to access this information and better communicate with teachers. It was decided that the server would use GNU/Linux due to its low cost, excellent security, and overall proficiency as a web server.

Preation Inc. was selected to partner with Blue Ridge School to create a school information portal. Preation created a custom software package that later became MainBrain School. GNU/Linux was chosen as the platform to host the website on for a variety of reasons. They needed a fast, secure, reliable operating system. They also needed a top-notch database to interact with the perl scripts on which the software is based. After significant research, they decided to use the open source MySQL database.

The school reports the finished product as a complete success. Parents, teachers, and administrators at Blue Ridge School have been thrilled with the project. Linux has been the perfect solution for their needs because of ease of administration, price, and reliability. Further, Linux's ability to interact nicely with other operating systems has made integration into heterogeneous networks simple. Finally, the standards compliance of Linux and the tools (Apache, MySQL, Perl, etc.) has made extending the software easy to accomplish.

Open Options Study

The Northwest Regional Educational Laboratory has done a lot of work in seeing how open source software fits into education. They have done case study research in an aim to provide K-12 educators with information to make decisions about open source software. Their survey targets educators in K-12 (including teachers, specialists, administrators, and students). Below is a summary of their study taken from their website, www.netc.org.

Who are they?

The majority of survey respondents work for school districts (rather than schools) and serve student populations of 2000 or more. Their districts are urban, suburban, or rural. Of those respondents who work for schools, the majority serve student populations of 100-499 students, and most work for high schools. Most respondents don't teach students. Many have administrative responsibilities and most choose software for other people's desktop computers. Most respondents are from the United States, and about half from Oregon and Washington.

What are they using?

Educators are primarily using open source on the backend, for Web hosting, Web filtering, email, and similar solutions. They prefer the Linux operating system, and many use recycled hardware. By deploying Linux, Apache, and other backend solutions, educators are leveraging the most mature, robust solutions the open source movement has created. These open source projects are large and stable, and the community of expertise and support is expansive. The community includes a thriving subpopulation of educators, so current users rely on mailing lists like K12 Open Source Now and community Web sites for advice and support. Many use open source operating systems for their own desktop computers, and many are using thin clients for their stakeholders. They choose open source solutions for the reliability, desirable features, and price. They also like the security. On the frontend they also look for interoperability.

Are they using open source on the desktop?

Most respondents use open source on the backend, but far fewer use it on the desktop. This may reflect the long maturity of open source solutions for servers, compared to the more recent development of user friendly frontend solutions (e.g. Mozilla, OpenOffice.org). This may also reflect the greater ease of migration on the backend.

Is it difficult to use open source in K-12?

Compared to similar solutions, most respondents find implementing most solutions easier or about the same difficulty. In write-in responses and interviews, some educators explain that migration and initial deployment can be challenging, but the solutions are more reliable and otherwise satisfactory in the long term. Resistance by supervisors or other stakeholders is light,

perhaps no more than resistance to any migration in technology. In write-in responses and interviews, some educators describe significant user resistance to changing operating systems, largely based on mindshare. Much of this resistance dissolves when users began to appreciate the new operating system or decide it's a trivial part of their computing.

What solutions do they recommend?

Red Hat Linux is popular among respondents, including K12LTSP (a distro built from Red Hat Linux). The partially reflects how the survey was publicized, as well as the relatively expansive community of Red Hat/K12LTSP educators. The minority responses reflect the variety of opinions and TMTOWTDI in the open source community. On the backend, Apache and SquidGuard are popular. Many respondents recommend OpenOffice.org, the GIMP, and Mozilla as frontend solutions.

What are their general opinions about open source?

Nearly all respondents believe that some open source solutions are mature enough for schools. (This is not surprising since they're deploying solutions in school.) Most want to use more open source solutions where they can. In write-in responses and interviews, some educators explain their plans for incremental migration as new solutions mature and the community grows. Most try to be somewhat pragmatic about choosing solutions based on their needs, while many are influenced by the values and philosophy of the open source movement. In write-in responses and interviews, some educators explain that they want to use more open source software, but that they would never deploy an inferior solution or otherwise neglect their professional responsibilities to their stakeholders.

Part IV

Appendices

Bibliography

Sources of Further Information

Glossary

Appendix A

Bibliography

Adelson, A. (2002). *Amazon.com: Migration from Unix to Red Hat Linux.* Framingham, MA: IDC. Retrieved March 27, 2002, from http://www.redhat.com/whitepapers/services/Amazon_case_study.pdf

BBC. (2002, June 3). *IBM signs Linux deal with Germany.* London, England: Author. Retrieved February 25, 2003, from http://news.bbc.co.uk/2/hi/science/nature/1749441.stm

Black, J. (2002, December 10). *Hollywood's digital love/hate story*. BusinessWeek Online. Atlanta, GA: McGraw-Hill. Retrieved February 25, 2003, from http://www.businessweek.com/technology/content/dec2002/tc20021210_0483.htm

COSTP. (2002). *California Open Source Textbook Project*. Retrieved December 15, 2003 from http://www.opensourcetext.org/

d'Empaire, A. (2002, July 27). *Microsoft's big stick in Peru.* San Francisco, CA: Wired News. Retrieved September 9, 2002, from http://www.wired.com/news/business/0,1367,54141-2,00.html

DiCarlo, L. (2002, March 27). *Wall Street embraces Linux.* New York, NY: Forbes.com. Retrieved October 1, 2002, from http://www.forbes.com/2002/03/27/0327linux.html

Economist Global Agenda. (2002, September 26). *Linux gets a break.* New York, NY: Economist Group. Retrieved October 1, 2003, from http://www.economist.com/agenda/displayStory.cfm?story_id=1338664

Elkner, Jeffrey. (2003). *Open Book Project*. Retrieved February 5, 2004, from http://www.ibiblio.org/obp/

Festa, P. (2002, October 10). *MIT tries free Web education*. San Francisco, CA: CNET Networks. Retrieved October 14, 2003, from http://news.com.com/2100-1023-961563.html?tag=cd_mh

Galli, P. (2003, Feb 3). *Microsoft warns SEC of open-source threat.* Woburn, MA: eWeek.com. Retrieved February 25, 2003, from http://www.eweek.com/print_article/0,3668,a=36573,00.asp

Gillespie, R. (2000, July 5). *German federal government to support open source software.* St. Paul, MN: Internetnews.com. Retrieved October 1, 2002, from http://www.internetnews.com/bus-news/article.php/408271

Lessig, L. (1998, March). *The laws of cyberspace*. Essay presented at the Taiwain Net '98 conference, Taipei, Taiwan. Retrieved September 23, 2002, from

http://cyberlaw.stanford.edu/lessig/content/articles/works/laws_cyberspace.pdf

Lessig, L. (2000, June). *Open code and open societies* [Draft]. Keynote address at Free Software - a Model for Society? conference, Tutzing, Germany. Retrieved September 23, 2002, from http://cyberlaw.stanford.edu/lessig/content/articles/works/opensocd1.pdf

Lessig, L. (2001, December). *May the source be with you*. San Francisco, CA: Wired News. Retrieved September 23, 2002, from http://www.wired.com/wired/archive/9.12/lessig.html

McCullagh, D., & Zarate, R. (2002, June 12). *U.S. gov't still penguin shy.* San Francisco, CA: Wired News. Retrieved October 1, 2002, from http://www.wired.com/news/linux/0,1411,53005,00.html

MIT. (2003). *OCW Help*. Retrieved December 15, 2003, from http://ocw.mit.edu/OcwWeb/index.html

Northwest Regional Education Laboratory. (2004). *Making Decisions About Open Source Software for K-12*. http://www.netc.org/openoptions/ Parts of the NREL's public domain content can be found in chapters 3, 4, 7, and 8 of this document.

Newbart, D. (2001, June 1). *Microsoft CEO takes launch break with the Sun-Times*. Chicago Sun Times, p. F57.

O'Reilly Media, Inc. (2004) *About the Open Books Project*

Orzech, D. (2002, October 8). *CIN: Linux TCO: Less than half the cost of Windows.* St. Paul, MN: Jupitermedia, Linux Today. Retrieved October 8, 2002, from http://linuxtoday.com/it_management/2002100801926NWBZMR

Pavlicek, Russell C. (2000). *Embracing Insanity*. Indianapolis, IN. Sams Publishing.

Proffitt, B. (2002, August 30). *Venezuela's government shifts to open source software.* St. Paul, MN: Jupitermedia, Linux Today. Retrieved October 1, 2002, from http://linuxtoday.com/developer/2002083001126NWLLPB

Raymond, E.S. (2003). Halloween Document I [Author commentary]. Retrieved September 21, 2002, from the Open Source Initiative Web site: http://www.opensource.org/halloween/halloween1.php

Shankland, S. (2002, August 15). *Verizon switches programmers to Linux.* San Francisco, CA: CNET Networks. Retrieved October 1, 2002, from http://news.com.com/2100-1001-949913.html

Scheeres, J. (2001, March 16). *Mexico City says hola to Linux.* San Francisco, CA: Wired News.

Retrieved October 1, 2002, from http://www.wired.com/news/politics/0,1283,42456,00.html

Tux4Kids. (2004). *Goals*. Retreived on February 13, 2004 from http://www.geekcomix.com/tux4kids/goals/

Wikibooks. (2004). *Welcome, newcomers*. Retrieved January 25, 2004 from http://wikibooks.org/wiki/Wikibooks:Welcome%2C_newcomers

Wikipeida. (2004) *Wikipedia*. Retrieved February 7, 2004 from http://en.wikipedia.org/wiki/Wikipedia

Appendix B

Sources of Further Information

DiBona, Chris., Sam Ockman and Mark Stone. Ed. *Open Sources: Voices from the Open Source Revolution*. Sebastopol, CA: O'Reilly and Associates, Inc., 1999

Feller, Joseph., and Brian Fitzgerald. *Understanding Open Source Software Development*. Harlow, England: Pearson Education Limited, 2002.

Free Software Foundation (www.fsf.org)

Freshmeat (www.freshmeat.net)

Just Linux (www.justlinux.com)

K-12 Linux (www.k12linux.org)

Linuxiso.org (www.linuxiso.org)

Negus, Christopher. *Red Hat Linux Bible: Fedora and Enterprise Edition*. Indianapolis, IN: Wiley Publishing, Inc, 2002.

Open Source Initiative (www.opensource.org)

Open Source School (opensourceschools.org)

Pavlicek, Russell C. (2000). *Embracing Insanity*. Indianapolis, IN. Sams Publishing.

Raymond, Eric S. *The Cathedral and the Bazaar*. Sebastopol, CA: O'Reilly and Associates, Inc., 2001.

Raymond, Eric S. *The New Hacker's Dictionary*. Cambridge, MA: The MIT Press, 1998.

Schoolforge (schoolforge.net)

Simple End User Linux/EDU (www.seul.org/edu)

Sourceforge (sourceforge.net)

Slashdot (www.slashdot.org)

Torvalds, Linus., and David Diamond. *Just For Fun: The Story of and Accidental Revolutionary*. New York: HarperCollins, 2001

Williams, Sam. *Free as in Freedom*. Sebastopol, CA: O'Reilly and Associates, Inc., 2002.

Appendix C
Glossary

A

Apache
An open source program for hosting Web pages.

B

Backend
The servers and other hardware that support a network. A typical backend includes an Internet gateway, Web servers, and email servers. Compared to a desktop computer, backend solutions are usually complicated and challenging, whether proprietary or open source.

Berkeley Software Distribution (BSD)
A UNIX-like operating system, comparable to Linux. One of the earliest examples of open source software, and the basis for Darwin. Apple builds its premium OS X on Darwin. See: http://www.bsd.org/

Beta testing
The process of testing and improving unfinished software. Beta testing is most useful when the testers aren't the original programmers and can provide specific, meaningful feedback. The open source model is sometimes considered permanent beta testing, since every program can be perpetually tested and improved.

Binary
The 1's and 0's of a compiled computer program. Computers can only understand binary, but programmers can't understand binary and need source code to fix or change a program.

Bind
An open source program for Domain Name Services. Software like Bind is necessary for computers to translate text Web address (e.g. www.netc.org) into the numeric IP addresses they need to retrieve Web pages.

Bug
Error in a program that cause problems. See also: Debug

Burnout
The low moral or hopelessness stakeholders feel because of chum overwork, unrealistic goals, and/or inadequate support. People stop trying because they feel like they've tried and failed, or their work is meaningless or unappreciated. A good tech plan tries to prevent burnout.

Business model
The plan a company uses to generate revenue. Companies that distribute open source software can't depend on control of the source code for their business model, so they have to thrive on service and other sources of revenue.

C

Capacity
A school's ability to create and support solutions. Capacity depends on good tech planning to retain staff, build expertise, deploy modular solutions, maintain some reliable solutions while experimenting for the future, and otherwise take ownership of the options and impact of technology.

Careware
An alternative software "licensing" program. The distributor asks the user to pay money for the software; instead, the distributor asks for a postcard, a story, or some other form of gratitude. Thus, the software transmits a philosophy as well as functionality. See also: shareware More: http://www.arachnoid.com/careware/index.html

Churn
The unpleasant feeling of unpredictability and instability created by frequent or erratic change. Churn is a real danger for schools, which should strive to maintain some predictability and stability for the good of all stakeholders. A good tech plan tries to prevent chum See also: burnout

Client
A computer accessing a server. If you're reading this page online your computer is accessing one or more servers, so it's currently a client. In computer science lingo, computers are clients while people are users.

Closed source
A program without accessible source code. See also: proprietary software

Command line
Perhaps the simplest user interface, a command line only allows the user to enter text. Perhaps the most widely-known command line is the MS-DOS prompt (C:\>). Command lines are widely considered to **not** be user friendly. Many programs include an accessible command line for debugging or other advanced uses.
Compatibility
How well a program works with a different program, especially sharing files. See also: Interoperability

Cookie crumbs
A short visual map of where you are in a Web site.

Courseware
Curriculum in an electronic form. This includes software specifically designed to support learning.

Copyleft
The opposite of copyright, as explained by Richard Stallman. Where copyright protects a creator's right to control copies and changes to a work, copyleft protects a user's right to copy and change a work. See also: GPL.

Cracking
Illegally accessing source code. Cracking also refers to other illegal activities, like copying video or music. Among computer programmers, hacking is skillful programming, while cracking is malicious. Open source software can't be "cracked" because it's already open. Also called "black hat hacking."

Customer Relations Management (CRM)
The technology solutions for large companies or other organizations to manage their contact with customers, including sales (and Sales Force Automation, SFA), shipping, service, and other "front office" activities. See also: ERP

D

Debug
Troubleshooting software, to find and fix mistakes (“bugs”). Requires access to the source code; otherwise, a programmer can only find mistakes.

Desktop computer
A complete, autonomous computer with its own operating system and hard drive. The greatest strength and weakness of a desktop is isolation; the computer continues to work even if a network fails. But compared to a thin client, a desktop computer is harder to repair or upgrade. Sometimes called a microcomputer.

Distro
A specific distribution of Linux. Each distro (e.g. Red Hat, SuSE) is built from the common Linux operating system, with added programs and functionality. Some people are very loyal to specific distros, creating a friendly rivalry not unlike the rivalry between Apple and Microsoft Windows users.

Driver
Special software for controlling hardware. Without the right driver, hardware (like a CD-ROM drive)

won't work.

E

Embrace and extend ("extensions")
Including new technology in an existing program or family of programs, then adding new formats and protocols. As a result, other or older programs using the technology are incompatible with the new formats and protocols. See also: Lock in. See also: Embrace, extend, and extinguish

Embrace, extend, and extinguish
Using the incompatibility of "extended" software to force the death of competing products. See also: Lock in

Emulation
Using special software to create an artificial operating system on top of a different operating system. For example, Wine creates a virtual Microsoft Windows operating system on top of Linux. This means critical, Windows-only programs can still run under Linux. However, emulation is imperfect and technically challenging.

Encryption
Complicated software or other methods to protect user privacy and prevent strangers from reading personal content (e.g. email). Many open source proponents also advocate for encryption, and open source software is better for encryption because it reduces the possibility of spyware.

End User
The person who uses a program after it's been compiled and distributed.

Enterprise Resource Planning (ERP)
The technology solutions for large companies or other organizations to manage their resources, including customers, supplies, accounting, and other "back office" activities. Similar to Materials Resource Management (MRP) but broader in scope. See also: CRM

F

Formats & protocols
The standards and rules a program uses to store and process information.

Free software
Like open source software any computer program with accessible source code. Anyone is legally and technically able to change and/or redistribute the software. Unlike open source software, the

source code can't be included in future closed source software. See also: FSF, Open source software. See: http://www.fsf.org/philosophy/free-sw.html

Free Software Foundation (FSF), The
Founded by Richard Stallman to promote the idea and practice of free software. See also: Open Source Initiative See: http://www.gnu.org

Freeware
Software distributed at no or negligible cost. May be free software, but not necessarily. Same as: No-fee software. See also: Shareware

Frontend
A desktop computer and the programs that run on it. See also: Backend

G

Gateway
The connection between a computer or network, and the larger Internet. Internet service providers offer gateways, to navigate, translate, and filter the Internet. The majority of gateways use at least some open source software.

GNU Graphical Manipulation Program (GIMP)
An open source image editor. Very similar to Adobe Photoshop. See also: GNU.

GNOME
One of the leading projects to create a GUI for Linux. The eventual success of GNOME and other GUIs is widely considered essential to mainstream success of open source software. See also: KDE

GNU
An open source operating system still in development. Richard Stallman founded the GNU project, as well as the Free Software Foundation. GNU stands for "GNU's Not UNIX" and is designed to be UNIX-compatible. Linus Torvalds used the free tools from the FSF to make his own open source operating system: Linux

GPL
The GNU Public License. The GPL was created by Richard Stallman to enforce the principles of free software. Where most software licenses restrict the rights of the user (e.g. to make copies), the GPL protects the rights of the user. Stallman calls this idea copyleft.

Groupware
Software to communicate and collaborate in a group. Scheduling software is groupware. Same as:

workgroup productivity software

Graphical User Interface (GUI; "gooey")
The visual symbols and choices to control a program. Most GUI's use windows, menus, and toolbars. Most operating systems use GUI's because most users are uncomfortable with a less user friendly a command line.

I

Internet Access Provider (IAP)
Same as: ISP

ICP
(1) Internet Commerce Provider; a company or organization that provides hosting or technical assistance to do business over the Internet
(2) Internet Connection Point; the junction between a local computer or network, and the larger Internet. See also: ISP
(3) Internet Cache Provider; a company or organization that stores frequently-accessed Internet content for faster service to its users; such stored content is called a cache

Initial Public Offering (IPO)
The initial sale of stock in a public corporation. The dot-com boom of the late 1990's coincided with the mainstreaming of open source software, leading to some over-rated IPO's for open source companies.

Internet
The global network of computers, including the World Wide Web. Beside Web pages, information like email and instant messages are transmitted over the Internet. The Internet and open source software are interdependent. The growth and stability of the Internet has depended on open source, and the open source movement thrives in a connected international community of programmers.

Internet Service Provider (ISP)
A company or organization that provides the connection between a local computer or network, and the larger Internet. Same as: IAP

Interoperability
How well a program works with another program, especially different operating systems. For instance, OpenOffice.org has strong interoperability because versions exist for Microsoft Windows, Apple Macintosh, Linux, and Sun Solaris. See also: Compatibility

K

K12LTSP
The K-12 Linux Terminal Server Project is a specialized distribution of Linux. This distro uses a server and thin clients, rather than stand-alone desktop computers. See: http://K12LTSP.org

KDE
One of the leading projects to create a GUI for Linux. The eventual success of KDE and other GUIs is widely considered essential to mainstream success of open source software See also: GNOME

Kernel
The beating heart of an operating system. All major software instructions pass through the kernel. The software equivalent of the central processing unit (CPU) hardware.

L

License
The official terms of use for a specific program. A software license is a legal document since it formally restricts the rights of the user. Some open source licenses (e.g. GPL) are radical reactions to restrictive licenses, under the idea of copyleft.

Linux
The leading project to create an open source operating system. The most successful example of open source software. Linux has two leading GUIs: GNOME and KDE.

Linux Users Group (LUG)
A grassroots organization of mutual support and community, especially for Linux hobbyists. Schools may find significant help from local LUGs.

Lock in
When users can't choose a new company's software because they have invested too much time or have too much data in their current software, and the change is too costly or otherwise impossible.

Loss leader
A company willing to lose money to attract an enduring customer base. The company sells a product or service below cost to attract customers.

Low threshold opportunity
A new solution that's easy to deploy for experimentation with limited risk. Open source offers many low threshold solutions, since the price is usually low or zero. Low threshold opportunities should be considered in a tech plan. (Some people refer to low threshold opportunities as "low hanging fruit.")

M

Microsoft Corporation (MS)
The largest software company in the world. The MS Windows 95/98/ME/2000/XP series is the most-used operating system in the world. MS is also an ISP. MS is widely considered to be aggressively opposed to open source software.

Mindshare
How many people are aware of a product or idea. Mindshare is marketing term; companies want to build mindshare for their products and services. In software, proprietary software (e.g. Microsoft) enjoys far more mindshare than open source (e.g. Linux).

Modular
A solution made of several discrete pieces, so that any piece can be replaced without rebuilding the whole solution.

Monoculture
A society that only offers one choice. For example, Microsoft is accused of seeking a software monoculture for its Windows and Office software. A monoculture is similar to a monopoly but has larger impacts. For example, if users can only choose a single operating system then they are limited by that software's design and features. A more diverse culture allows different groups, companies, and users to create and experiment with different designs and features, promoting innovation and competition.

Mozilla
An open source Web browser.
Mozilla Public License (MPL)
An unusual license that illustrate the difference between free software and open source software. Under this license the software company (i.e. Netscape) can add community modifications to Mozilla to a new proprietary program (e.g. the next Netscape browser).

MRP
See: ERM

N

No-fee software
Same as: Freeware

Non-disclosure agreement
A binding legal agreement that prevents the user from sharing software or knowledge about software. Part of many software licenses. See also: Shared source

O

OEM
This term once meant "original equipment manufacturer" or the company that created and assembled the components of a computer. However, most components are now commodities, so OEMs like Dell don't actually manufacture components. In the real economic model they are middle marketers, assembling components and selling complete computers to customers or vendors. But since few people assemble their own computers, the label is still used.

Open source software (OSS)
Any computer program with accessible source code. Anyone is legally and technically able to change and/or redistribute the software. Unlike free software, the source code can be included in future closed source software. Open source is a controversial term that is still being defined. See also: Free software. See: http://www.opensource.org/docs/definition.php

OpenOffice.org
An open source office productivity suite, including word processing, spreadsheets, and slideshows. Very similar to Microsoft Office.

Operating System (OS)
The essential software to control both the hardware and other software of a computer. An operating system's most obvious features are managing files and applications. An OS also manages a computer's connection to a network, if one exists. Microsoft Windows, Macintosh OS, and Red Hat Linux are operating systems.

Open Source Initiative (OSI)
The nonprofit organization behind the open source movement. Key leaders are Eric Raymond and Bruce Perens. The OSI's Open Source Definition and Open Source Certification Mark were created to distinguish open source software from free software. As a result, open source is considered more business-friendly than free software. See also: Free Software Foundation. See: http://www.opensource.org

Opportunity cost
The cost in dollars or stress when old choices prevent new choices. For example, by migrating to Linux a school might foreclose on using future software developed for Microsoft Windows. See: Total Cost of Ownership

P

Pile of PCs (PoPC or "popsie")
A supercomputer composed of networked microcomputers. Linux is famous for enabling low-

budget, high-performance PCs.

Proprietary software
Software without accessible source code. Most users only buy or receive the binary. Sometimes called "closed source."

R

Return on investment (ROI)
The total value gained after a solution has been deployed. This is usually a figurative term, like total cost of ownership, since the true cost of deployment or migration is hard to quantify. A positive return on investment is desirable, since that means the solution has solved more problems than it creates.

S

Sendmail
An open source program for routing email messages.

Server
Computers are usually networked to share files and communicate. A network of computers is usually controlled by a master computer, or server. There are many kinds of servers. A network server governs access to the computers. A file server offers a common hard drive for many people to use. A Web server stores Web pages and sends them over the Internet as users surf that Web site.

Shared source
A Microsoft plan to make some of its software transparent to some users. This is not open source because only some people have access to the source code and they can't make changes or copies.

Shareware
Software you can use without paying for upfront. If you regularly use the software or want all the features you pay for it. Not necessarily open source. Not the same as freeware. See also: Careware

Shifting Standards
Regularly changing the format or other parts of a program. This accidentally or deliberately interferes with other programs' compatibility. For example, the Microsoft Word format has shifted several times, making it difficult or impossible for people to use older versions or competing word processors.

Source code
The special language a program is written in. A programmer needs to source code to fix or change a program.

Spyware
Secret code hidden in an otherwise harmless program. Spyware permits unauthorized access to a computer, allowing someone else to observe the user, read data, or even control the computer. Open source is transparent, so it's nearly impossible to hide spyware.

Stakeholder
Stakeholders may include administrators, instructors, other staff, students, parents/guardians, community members, and technology experts. A good tech plan can only be created and implemented with stakeholder involvement. Stakeholders should be included in the tech planning process.

T

Thin Client
A minimalist workstation connected to a server.

Third-party program
Typically, a program developed for another company's operating system. (The first party is the customer. The second party is the operating system company.) Third-party programs depend on the success of the operating system for their own sales.

TMTOWTDI, "Tim Toady"
There's More Than One Way To Do It. An alleged principle of good programming and good software. In programming, TMTOWTDI means respecting and exploring multiple solutions to a problem. In software, TMTOWTDI means offering the user multiple ways to use the software. For example, in most word processing programs the user can use a menu or use a key combination (File: Save or CTRL-S).

Total cost of ownership (TCO)
The total price in money, time, and resources of owning and using software. TCO is widely considered to include the purchase price, and the cost of hardware and software upgrades, maintenance and technical support, and training (or re-training). TCO is an essential part of technology decision making since a product may be cheap or free to get, but involve excessive maintenance or training. TCO is not unique to computers. For example, the TCO for commercial jets is measured in cost per mile. This calculation includes the sale price of the plane, the cost of fuel and other expendables, maintenance and repairs, etc. See also: Return on Investment.

Transparent
Anyone can see how a program works. Transparency doesn't necessarily mean open source. For example, some proprietary software companies let special customers see how a program works, but the customers can't make changes. Open source software is transparent and anyone can make changes.

Turnkey
A solution that works automatically with little or no troubleshooting. Much like a new car, a turnkey solution just works without demanding additional expertise or work. This is a critical issue for TCO, and many current users desire/expect more turnkey solutions in open source software.

U

User
A person who uses a computer, including a programmer or end user.

User friendly
A program is widely considered user friendly if the user interface is clear and intuitive, if the program offers useful help and tutorial features, and if the user can start and finish a task without learning or using more features than necessary. Apple Macintosh is widely considered a leading user friendly operating system, so many operating systems have copied it features.

User Interface (UI)
How the user controls a program. Perhaps the simplest UI is a keyboard and command line, to enter text commands. Sometimes called a "console." See: GUI

W

Wine
Emulation software designed to allow programs that usually run on Microsoft Windows to run on Linux instead.