

Software Freedom: How OpenSource Drives the Market

i. Preface

It's typical for an author to place a preface to a work right after the foreword, explaining the purpose of the work and the reasons for writing it. There is no foreword to this paper simply because I couldn't find anyone gullible enough to say great things about me and my research methods. I've found that self-deprecation works wonders for lowering the expectations of an audience, allowing thinly disguised arguments and poor conclusions to sound pretty good. Free beer doesn't hurt, either. If you're reading this paper, please come by my office to get your complementary adult beverage before passing this point; it will make your experience much more enjoyable. I am open to criticism for anything I have written in this paper, and in fact seek information contrary to my own findings.

If you've gotten this far, you want to know what this paper is about: it concerns computer science students at Texas A&M, and how they feel about programming in general. It is based on my own experiences as both an observer and student the past year and a half, as well as some formulated experiments and discussions I had with several students. Perhaps this project would be better suited for a psychologist, anthropologist, computer scientist, etc. However, I feel that the point of philosophy is to not necessarily answer questions, but to create new ones. I have raised more questions in my own mind than I answered, so I figure it's a net gain. The advantage a philosopher has in doing this sort of research is that any answer is a good one. It is apparent after reading the introduction that I had an idea of what sorts of results I'd get, but the fact that the actual results differ from

what I expected is better in the long run. I suppose this goes back to wanting to learn something instead of trying to produce a paper that tries to prove a point. I enjoyed doing this project, mostly because I never knew what to expect.

ii. Introduction

This seemingly strange title for a “philosophical” paper is quite appropriate considering the conclusions one reaches after researching computer science for any length of time. I started my internship with the intention of producing some great work detailing how copyright law was a legal quagmire as concerns the software industry, and that it created a state of non-innovation; what I discovered was far more interesting, at least to the casual observer. Certainly copyrights are important, and I cover them in some detail in this report, but after a length of time it became apparent to me that this was just the tip of the proverbial iceberg. It had been my guess, and prior experience had given me decent justification to think such, that most computer science students were concerned with copyrights and their implications on the development of software¹. This **is** the case, but not for the reasons I had attributed previously. Perhaps as a student of philosophy I was thinking too much about the topic and imposing some of my own preconceived notions about what had bothered me in the past when creating computer programs. I don’t think any author or researcher can totally avoid this, at least initially, but I tried to remain open to anything just in case my opinions were shaping my perceptions. Let’s just say that I didn’t want to be biased just to create something that fit my own thoughts on the matter. Hopefully, my attempt at impartiality succeeded.

¹ I am going to focus almost entirely on software in this paper, as hardware design is more the province of engineers and less apropos to a discussion on the current state of computer science. This has a lot to do with a definitive shift that has taken place in computer development as hardware has become an almost commodity item with various vendors forming standards used by everyone. Software, however, has remained highly competitive, with very little sharing involved. Innovation is rarely, if ever, given away. Opensource strives to shatter this current model.

To get to the “interesting” parts I mentioned, the results are really quite simplistic yet compelling. Most students, especially here at Texas A&M, have an almost love/hate relationship with software copyright and patents, to the point of absurdity. They seem to revel in being digital cowboys, coding by the seat of their pants (Gap khakis replacing Wranglers) and producing quality work that they guard judiciously, yet seek to share with their peers. Originality is rewarded, but theft is not. Implementing an idea in the same method as a superior programmer, which could be seen as a type of benevolent worship of skill, is frowned upon and perhaps chastised. Still, the programmer who succeeds in creating something useful wishes for peer appreciation, to be accepted and acknowledged as someone who can produce quality. This dynamic isn’t entirely student-generated; it doesn’t arise purely from the interactions the students experience with each other. It appears that the manner in which classes are presented, on-campus recruiting, events, organizations and the general feeling on campus greatly influences the average CS student into following this line of thought. As you read this paper, it is my hope that you come to the same conclusions I have, although simply gaining a greater understanding of the thought process of a student doing work in computer science would be better.

My research was done using several methods, nearly all of it interactive. As I mentioned earlier, one of the social sciences might seem more appropriate for such investigation considering the manner I pursued, mostly because engaging with the subjects of research is something that can skew results. I sat in on several classes over the course of the Fall semester of 2003, namely Programming in C and Operating Systems Design, both undergraduate classes. Spring 2003 I was enrolled in a graduate-level programming class,

and Spring 2004 I took Database Design, another graduate-level computer science class. In the classes that were taken for a grade, I used this experience to see precisely what a normal student in the CS department would encounter during their studies. This was especially appropriate considering that the Database class itself was cross-referenced with an undergraduate class, and in fact 80% of the students were undergraduates. The other classes which I simply attended were extremely useful because I was able to focus on less technical aspects and more on the social dynamics of the lectures. Some of the students I met in these classes agreed to discuss with me their thoughts on education and software, which gave me further insight. The completion of several class projects was a rather enlightening way to directly experience the social interactions between students when producing quality (or, in some cases, shoddy) work. Finally, taking a page from Eric S. Raymond, I conducted my own private experiment, much like he describes in his essay *The Cathedral and the Bazaar*. This paper is the culmination of personal experience with a philosophical bent, although I like to think it was rather scientific in method.

The beginning of this paper outlines some of the ideas behind free software, or Opensource, to provide an introductory background for the ideas presented later. I then cover some of the problems that exist at the university level with educating students using closed and open source software, namely how students' views on software are shaped by the method in which classes are taught. Part three details group dynamics, how students interact with each other and what sorts of issues arise from working together. This is followed by an account of how students feel about licensing, the protective aspects contrasted with the restrictive characteristics. I close with a detailed account of the

experiment I conducted, which sheds light on some of the difficulties using open and closed source software (as well as demonstrating some of the good parts).

I. The Goals of Free Software

Before I can give an accurate report of my experiences, it is appropriate to explicate some ideas concerning free software; the reader might require this background to fully understand parts of this paper. Free software, in its current form, is not so much a business model as it is an idea, almost a way of life for some. The free in free software means freedom according to Richard Stallman, the man responsible for starting GNU². GNU's original purpose was to create a completely free operating system, which unfortunately never materialized over the years. However, GNU has produced numerous high-quality tools, a very prominent one being gcc, a compiler used for producing binary code from source files. The most important contribution to the free software movement, however, is not the numerous utilities GNU provides, but instead the GPL³, or GNU Public License. The GPL is a software license that imposes many restrictions on the manner of distribution of computer programs; some would say the GPL is **too** restrictive, hence the proliferation of numerous other licenses, some of which are detailed later. The purpose of the GPL is to require any software that falls under it to be distributed with the source code, files that can be used to produce the binary information computers understand. Microsoft, for instance, does not provide the source code to its customers. If a bug crops up in Windows, the user must wait for Microsoft to issue a patch to correct this error. The current proliferation of viruses, trojans, worms and Denial of Service attacks on the Windows platform is a primary indicator that relying on a corporation to correct bugs has its problems. Programs that use the GPL are user-modifiable; if a bug is discovered, the user himself can correct it. While most users of computers lack the skill

² GNU is pronounced Gah-noo, unlike the large draft beast used as the GNU mascot. See <http://www.gnu.org> for information on all the available free software projects.

³ <http://www.fsf.org/licenses/gpl.html>

or inclination to pursue such fixes, the fact that a choice is available shows the type of freedom Stallman strives for. Free software can have a price associated with it, and very often does. Red Hat Linux, for example, costs in the neighborhood of \$100, even though it is composed of entirely free software. The \$100 pays for the work involved with putting together a reliable installation, one that can be seen as a sort of standard users are familiar with.

Linux is a popular buzzword in the computing industry, but also an extremely popular operating system as well. As mentioned previously, GNU never produced a complete solution; the GNU kernel⁴ remaining vaporware for many years. Linus Torvalds, however, created his own kernel called Linux nearly 14 years ago as an exercise in computer programming. The initial goal of the Linux kernel was to create clone of Minix, an educational Unix-like system. Torvalds became annoyed with the limitations of Minix and began coding his own operating system. He successfully used the GNU tools to put together a complete solution, one that spread through the internet community like wildfire. The free software model allowed Linux development to take place at an almost breakneck speed, and it soon rivaled commercial operating systems in reliability and features. Before Linux, GNU software was used primarily on proprietary Unix systems such as HP's HP/UX, IBM's AIX and Sun's Solaris. These systems still exist, but are rapidly falling behind as Linux and other free operating systems take market share. After Linux, GNU had a foothold in the software industry as a total solution, no restrictive binary-only licenses needed. Linux can be seen as a major catalyst to the explosion of

⁴ A kernel is the lowest level of software abstraction, the code that directly interacts with the hardware. The GNU kernel (HURD) has been released finally, but is still in early alpha testing and quite unusable for most tasks.

free software in business and home use, as even Walmart sells Linux-based computers for the average home user.

Free software is also often called Opensource, a term created by a number of free software advocates to differentiate between free as in price and free as in freedom. A popular tagline is “think speech, not beer”, i.e. the freedom to do something as opposed to getting something for nothing. The Opensource group was founded specifically to counter many of the misconceptions people get when they hear of free software, and perhaps to tone down the always brash and outspoken Stallman. From experience reading Stallman’s writing, it seems he is not only extremely opinionated, but unwavering in his goals. I am both impressed by his tenacity over the years, and appalled at his behavior regarding certain aspects of the software industry⁵. The formation of the Opensource group is easy to understand after dealing with GNU software for any length of time.

I’ll mention here that various operating systems besides the well known Linux are referred to in this paper, the most prominent being FreeBSD. FreeBSD falls under a BSD⁶-style license⁷, or a license patterned after the original license granted to the University of California-Berkeley. The BSD license is very loose and unrestrictive compared to the GPL, which in turn creates a different atmosphere among the users as compared to those in the Linux camp. I attempted to capture some of this atmosphere

⁵ Stallman has been extremely abusive of individuals who question his motives, or the GPL. A quick Google search, or a read of <http://www.stallman.org/>, reveals some of his anti-establishment writing. He has been labeled as a Communist due to his overwhelming hatred of the method in which the software industry is run. I don’t think this lessens his contributions to the Opensource movement, but it does make one wonder about his true intentions.

⁶ Berkeley System Distribution

⁷ <http://www.freebsd.org/copyright/freebsd-license.html>

with my personal experiment, as it is quite fascinating to see the different approaches different groups take towards free software.

It should be obvious that I can't cover the entirety of computing history in a few opening pages of this paper, but the preceding should be enough background to have a decent understanding of what I'll discuss later. The list of references I've provided give excellent background information and are quite entertaining as well.

II. The Commercial/Educational Rift

It seems rather strange that an institution of higher learning would advocate proprietary methods of performing computing tasks, yet this is what occurs in a sense here at A&M. I suppose this has as much to do with the misconceptions of what exactly Computer Science (CS from here on) is as anything else. There is a camp of individuals who believe that the goal of a CS program is to produce individuals who can perform tasks for monetary gain, and another group who seeks people to do research for a sort of “greater good”. This is no different than in any other discipline, but the lines are drawn much more clearly. For example, chemistry majors might one day work for a pharmaceutical company creating new drugs, or they might get a PhD and create the same drugs under the mantle of a university. If the drug is useful, the corporation might release the rights to copy it, relying on patents to collect income. University sponsored research might give the drug away for free, relying on the prestige gained for future endowments. In either case, there is a sense of scientific endeavor; the chemist himself strives to help mankind in at least some small way. Not so with software. CS research in an academic setting is often seen as pure rhetoric with no real application in the marketplace. There have been plenty of instances where academia has produced excellent ideas used throughout the computer market, but the disdain held for most academic researchers is quite astounding. Research in large corporations is often viewed as the only method capable of producing real results. Sun, Microsoft, Novell, Oracle and IBM, to name a few companies, all have large R&D departments that regularly innovate and create. These companies seek the best from CS, attempting to separate them from education. A researcher working for IBM is

never going to be able to release his work in the same manner the chemist will. NDAs and other restrictions imposed to keep the work proprietary will always limit just what the software engineer is able to do. So while the pure sciences almost revel in the sharing of information, commercial software research prohibits it in many ways. The academicians typically stick with standards for essential technologies; the World Wide Web Consortium (W3C) is a good example. The W3C seeks to maintain standards for HTML and various other web tools so any company that wishes to produce a web browser can follow a set of guidelines that ensures web pages will look how they're meant to. Before the W3C existed, the browser wars between Netscape and Microsoft almost splintered HTML into several different versions, which would have segmented the internet; do we really want a Netscape internet and a Microsoft internet? Many programmers on both sides saw the W3C as a bunch of stuffy professors who were trying to dictate the market into some less than ideal mediocre compromise. It's easy to see from this historical example why the university researcher is generally not liked in commercial software. Still, there is a sense of respect given to these researchers, and standards are generally followed once they are laid out. The line between academic pursuits and commercial ones are very clear in software development.

What does this have to do with the typical CS student? As mentioned before, large corporations regularly seek bright students to recruit, much like many companies do at the various career fairs we have on campus from time to time. However, computer companies are much more draconian in their attempt to draw in future employees. Microsoft is the primary bugaboo in the software industry now, their operating systems

and products being nearly ubiquitous in offices and homes around the country. The closed nature of their software means that any employee working for them must remain close-lipped and buy into their model with no reservation. To accomplish this task, Microsoft nearly always seeks new graduates who have no prior industry experience. They regularly fund parties and events on and off campus, provide free books, give students access to much of their source code, and generally attempt to create a feeling of goodwill between the company and the students. Landing a job as a Microsoft programmer or researcher is seen as a good thing by most students, even if it means their work will never be shared with anyone outside the company. Within their own department, however, cooperation is essential to producing products. This is a stark contrast to what goes on in the typical CS class. When working on group projects, for instance, there is a requirement to share information to produce quality code. Most students seem rather reluctant to give away their best ideas, for reasons I have yet to determine. Perhaps “give away” is a bit loaded, but I’ve experienced a lot of opposition from some rather bright students to let everyone in on what they were doing. Within their own group, these students are very secretive, sometimes at the expense of the group finishing with good code. It’s possible these students see the sharing of information diluting their chances of recruitment upon graduation: if everyone knows what they know, there may not be enough to separate them from the crowd. Even though this secretiveness exists, these same students often brag about how they’ve solved the problem in only five lines of code, or something similar. They truly seek group approval but are unwilling to actually give anything that can be approved of. Doing CS work for the sake of learning, which is the goal of academia, is frowned upon on a pragmatic level.

The advocacy that the university places on doing things in a proprietary manner isn't explicit, but it should be apparent above. Texas A&M encourages Microsoft's interaction on campus which in turn makes the students work in a "closed" environment. The guarding of their knowledge creates a proprietary method of code production wherein ideas are not shared. It is almost as if these students are trying to maintain corporate trade secrets that could be used by their competitors to cut into their bottom line, i.e. hiring potential.

Reliance on Microsoft Windows in the classroom has further encouraged this closed-model. Every classroom on campus that has a presentation system (the number is actually quite staggering) uses Windows 2000 or Windows XP. Professors give lectures with PowerPoint slides and send out class information via email; many students read this email using the Open Access Labs which are equipped with nearly all Windows machines. In my Engineering Ethics class, many students would submit drafts of their papers for comments; every single paper I received was written with some version of Word. In fact, I typed this paper up with Word, giving even more anecdotal evidence on the pervasiveness of Microsoft products on campus and in use by students. This almost total domination of Windows on desktop computers might lead some students to believe that Microsoft **is** computing. Several students I spoke with admitted to having no exposure to anything but Windows before coming to A&M, and even then it was exposure in a limited capacity. It is my belief that this domination causes a sort of mob mentality where the average student sees no alternative to using anything other than

Microsoft products, which shapes their ideas on how software should be developed and distributed. Again, the university encourages the proprietary nature software; subtly, although hard to ignore.

In an almost complete contradictory manner, the university actively encourages open computing. I recall one lecture in Operating Systems that focused specifically on file systems. Since the NTFS file system, the one used by Windows NT, 2000 and XP, is closed in nature, it would be an extremely poor example for discussion. File systems and their implementation are incredibly important to how an operating system functions, so it is vital to have an intimate understanding of a file system before good OS design can take place. NTFS cannot be dissected, nor can it be reproduced by a student unless he resorts to illegal reverse engineering, an action that A&M probably wouldn't want to encourage. Instead of focusing on a very pervasive file system, the lecture covered the BSD file system, commonly called UFS⁸. The source code that implements UFS can be found in roughly two seconds using Google, a far cry from the closed nature of NTFS which one might be able to license from Microsoft for a substantial sum of money. While UFS is used by a great number of Unix systems, most students are probably unaware of its existence. Still, it is generally agreed upon in the computing world that UFS is a very robust file system, and possibly one of the best (the fact that many commercial versions of Unix have used UFS throughout the years supports this). As a learning tool, UFS cannot be beat. It is a commercial grade implementation that students can pick apart and learn at whatever level they wish to delve, very unlike NTFS which someone might

⁸ UFS is actually an outgrowth of the original FFS, or Fast File System, employed by the free release of BSD, generally called 4.4 Lite. UFS stands for Universal File System, giving one some insight on the humbleness of programmers.

become acquainted with on a usage but not technical level. So while the university provides Windows machines and almost requires the use of Microsoft products in many ways, it must resort to open computing to actually educate.

Another example of the encouragement of open computing would be the use of database resources. My Database class required a group project, an implementation of various schemas into a working database. At the start of the semester, we were told Oracle, a closed program generally held in high regard, was available for our use, as well as PostgreSQL, a freely available database whose source can be downloaded at will. While I will admit that looking at the source code of a database sounds rather dull, if I wished to understand certain aspects of database programming such an exercise might be very useful. Around a week or two after class began, we were informed that Oracle was no longer working and wouldn't be available for student use. We were told to use PostgreSQL, provided by the university, or MySQL (another free database similar to Postgres) which we could install ourselves at home. Never once was Microsoft SQL mentioned; the feeling I got was that it was assumed no student had the money for a license. Rightfully so, as licenses for MSSQL are generally several hundred to thousands of dollars. We were encouraged to use free solutions and not the closed ones to do our work. This might be seen as purely a cost-based strategy, but I can hardly blame the professor for telling us to use MySQL as it's not only free, but extremely robust and dependable. I was interested in learning about databases, not spending money on corporate-priced software.

The general disdain with which Microsoft products are viewed gives another example of the contradictory nature of the university's views on proprietary software. Going to software.tamu.edu, a student or faculty member can purchase Windows XP, Office and several other Microsoft products for a minimal charge. A quick glance at a fee statement reveals the licensing costs the university charges students to fund this dirt cheap software. Listening to the average student or faculty member, one might think no one purchases or uses this software. Countless times in class, someone, either the lecturer or a student, would tell of a bad experience with a Windows-based system. "This is a brain-dead way of doing it" was overheard in reference to the method used by Windows file-sharing. I don't remember one student I spoke with saying they thought Microsoft was a leader in the development of good software, though many students gave plenty of gripes. It can be argued that professors do not speak for the university, but in a classroom dealing with a particular subject, if a professor states that a certain way of doing something "sucks", this can be viewed by the average student as an admonishment of a product by the university. So while A&M is providing Microsoft products at a reasonable cost, the general consensus of the university is that these products lack quality; students might already believe this prior to time spent in the classroom, but when their professor justifies their belief it will typically be set in stone. I do not blame a faculty member for providing their personal views on software, but I do think it is rather interesting that the reliance on Microsoft products is met with reluctance.

The reliance on Microsoft products is a common theme in corporate computing. Many companies like Opensource (the proliferation of Linux points to this), but seek a

scapegoat of sorts. In the past, IBM was seen as a requirement in business. Either you had IBM products or you were inferior. Now, Microsoft has taken over the mantle from IBM, but in a different manner. Where IBM was reliable, Microsoft is fallible. My prior experience working for computer companies leads me to think that many IT professionals chose Windows 98 not to be compatible, but so they could simply say “Well, it is Windows” whenever a crash occurred. If everyone was using Linux, there would be no central entity to blame. “I hate Bill Gates” is a statement that can contain a real emotional component; “Those thousands of nameless hackers, many of whom have yet to graduate high school, will feel my wrath” doesn’t roll off the tongue so easily, and the anger felt at a software issue is diluted severely. Due to the number of Windows installations in business, the university most likely feels the need to provide students with systems they are familiar with. Why would the students be familiar with Windows just because businesses use it? Since nearly all businesses have Windows systems (I can’t think of any offhand that don’t⁹), a family will probably have a Windows system, as taking work home is very common. Children grow up being accustomed to Windows machines, hence their familiarity. I think, again, that outside “pressure” influences the university environment to mirror what goes on in a corporate environment; the university wants to produce students who can gain employment, too. It doesn’t appear, to me anyway, that this directly affects CS students as much as students in other majors. A business major will probably spend a substantial amount of time creating PowerPoint presentations, an English major writing papers. Since these students rarely have contact with a computer

⁹ It’s interesting to note that I have a few friends who work for Apple Computer in Austin. You might think that Apple would be the best place to start when looking for companies that don’t use Microsoft products; this assumption is incorrect to say the least. Windows NT/2000 servers abound, and every desktop system has Office installed (albeit the OSX version).

outside of some basic tasks (email, reading web pages, etc.), they see computers as mere tools. Their views on what a computer is and what it's supposed to do are reliant on what they use in the lab; many may not even own a home computer. Their future employer probably expects them to be intimately familiar with these programs as well, so knowing how to use PowerPoint might be the only real use for a computer a student has. CS students are expected to know systems in great detail, so they are much more open to alternatives, and probably experiment¹⁰ excessively. A good analogy might be automotive repair. Most people in Texas buy trucks, and a whole lot of them buy Fords. The general consensus might be that Ford trucks are good for the money, so the average individual who has to haul around some drywall or lumber doesn't feel the need to worry about any other models. The truck hauls stuff from point A to B. The mechanic, however, knows more about trucks than most people care to hear about. If an average Texan said "I drive a Ford", the mechanic might list a thousand recalls with the '04 model, as well as reasons why the Toyota is vastly superior. Even though the mechanic might think Fords are junk, he can still repair them. The mechanic might even drive a Ford himself, simply because he can fix it easily due to the abundance of parts. He still reads about the Toyotas and Dodges and repairs them from time to time, but generally sticks with Fords because that's what makes the money. Tex drives a Ford because it's the accepted standard. In the same way, the CS student uses Windows because he knows the OS inside and out, working with it everyday. He might use Linux at home, or subscribe to a Macintosh magazine, but Windows is what provides his paycheck. The businessman just wants to do his PowerPoint slides so he can sell more units of bath soap; the computer holds no appeal other than a tool to do a job.

¹⁰ Most definitely...

The university needs Opensource to educate, but must provide closed software such as Windows to meet the needs of the vast majority of students on campus. For whatever reason, Microsoft is a dominant force in education, even though their model is highly resistant to providing learning tools. I think the university is in a catch-22 situation: if it does not give students Windows and Office to use, they won't learn skills necessary to get a job. Further, these same students have a certain expectation of computing that must be met for a minimum of technical support. By sticking with proprietary systems for the majority of systems available, however, the university leads many people to believe that these systems are the only way to perform certain tasks. The businessman wants PowerPoint, he hires a CS student to maintain PowerPoint for him at some point in the future, which means the CS student must move away from other operating systems if he wants a job. The market drives the university to use Windows, and the university in turn drives the market to do the same. It's a very strange chicken-and-egg problem that I don't think can be fixed anytime soon; whether or not it should be fixed is a matter for some debate. I think that having an almost monopolistic model of computing is bad for the industry as a whole simply because innovation occurs only through competition, but how much innovation do we actually need to play Yahoo! Checkers and type up reports? Computers have certainly gotten exponentially faster the past few years, as predicted by Moore's Law, but how much faster must they be for the average person?

III. Group Dynamics or Lack Thereof

As I mentioned earlier, most students are rather protective of their work, often to the point of detriment in a group setting. Let me clarify: **good** students are protective of their work. The ones who are struggling generally do not care about sharing, or have nothing to share in the first place. Still, they seek approval from their peers, which creates a strange environment in which to accomplish a given task. By approval I mean many students literally want their peers to revere them as “badass hackers”, programmers who can create things of beauty that not only functional but innovative. The problem arises when these students try to show their peers their work: how can they maintain their trade secrets and still show off? Before I delve deeper into this enigma, I’d like to lay out some observations I’ve made about groups in general.

My experience has been that the typical student workgroup is very similar to a corporate setting in many ways, the primary being how the group is constructed by various individuals. What I mean specifically is that there are a few types of personalities represented in any given group which dictate the manner in which development proceeds. To put it another way, there are students who sandbag, just like in Corporate America. The types of “workers” I’ve observed follow.

Natural Born Leaders (NBL) - Those individuals who like being in charge and accept the onus of responsibility in exchange for power. Many of the NBLs are lackluster coders, or have no true talent at programming, but they can be depended on to make sure other

group members do their jobs. Excellent at scheduling meetings and keeping track of who's doing what, etc. It's my guess that they'll one day make great managers.

Middling Ability Joe (MAJ) – An average student who will do whatever task assigned by the NBL, but usually doesn't provide any real ideas to a project. Some might actually fit into another category, but for whatever reason are apathetic with the whole process. Many graduating seniors seem to fit this description.

Know It All (KIA) – Exaggerates about their own abilities, undependable, all talk and little work. Might delay a project due to not doing their part. Sometimes the KIA is simply dissatisfied with the direction the group is taking, so he resorts to poor behavior as a sort of rebellion.

No Show (NS) – The guy who is officially in your group but never attends a meeting or does any work. Is usually smart enough to attach his name to the finished product before it's submitted for a grade.

Real Hackers (RH) – Exceptionally bright programmers who not only do the majority of the work, but seem to enjoy doing it as a test of their skills. Sometimes they propose things far beyond the scope of a given assignment in an attempt to create something truly original. Judiciously guard their work unless it's absolutely necessary to share.

These few categories obviously can't define every student, and some overlap might actually occur, but for quick-and-dirty stereotyping, I think it works rather well.

Given the rough stereotypes above, it isn't hard to put together a hypothetical group and have some idea of how that group operates. In my own personal experience, I was fortunate enough to be in a group of one NBL, a RH and a MAJ (I'll let you figure out which one I was). This group dynamic seems to work the best as the leadership is restricted to one person and there aren't any slackers. Whatever the hacker suggests is usually approved of, and work gets completed on time. To provide a contrasting group, consider one composed of two NBLs, two KIAs and one NS. This group was a nightmare to say the least. The two alpha-types vied for control instead of delegating work; the KIAs constantly bickered amongst themselves about whose idea was the best. The fifth member never even came to class, although I wonder at the true reasons behind this (he might have seen the futility of trying to work with his group members). Having done software development at the corporate level, the similarities between student groups and development teams are striking. I am unsure if this is a function of how people generally act when they are put in group situations, but I'd be inclined to say it is. Philosophically speaking, why some people feel they have the right to be totally without blame in case of failure but wish for admiration for another's hard work is puzzling. There might be a need to change how group projects are graded at the university level in an effort to correct the poor work ethic some students have, and perhaps to improve work conditions in corporations. Then again, many students openly complain about the NSs and KIAs but do nothing to actually curtail their behavior. The general consensus is "let's get it done",

regardless of how much extra effort they must exert. I'd like to think that these students will become successful in the future, while the sandbaggers are still stuck in entry level positions for years to come.

To get back to the enigma I started off with, it's easy to see what students we're talking about. The Real Hackers are the ones doing the majority of the actual coding, and most of the original thought as well. Many times these individuals are not socially advanced and have a hard time participating in a group setting. The average person might call them nerds. With the internet becoming more and more prevalent, face-to-face interaction has become less of a requirement for communicating, and software projects can be (and are) completed without the participants ever meeting each other. The hackers have been allowed to become more and more reclusive due to technological advances. Within their own circles, which exist purely in electronic form, these hackers are seen as elite individuals. They frequently challenge each other to competitions of skill; this might be coding, algorithm determination, or some other task that computers are suited to solve. When the hackers are required to attend a class and do a group project, most of them feel out of place and uncomfortable. The admiration they commonly experience is missing, instead replaced with what many of them consider incompetence. I think this is the primary reason the hackers do not share their code: they feel the people they are working with really aren't their peers. Still, they do recognize the necessity in participating because their grade is directly affected, and usually they can be coerced to share their work. Further, even if the group itself is not ideal or his group members might not be "worthy", the hacker still wants to show off in a manner he's used to. For the hacker,

coding is an art, and thus a pure expression of his personality. He might want everyone to know how advanced his programming skills are, but feel they can't fully understand this expression. Thus, the situation described above occurs. The hacker wants to feel comfortable, and that means he shows off his work. He wants others in his group to recognize his superior skills. However, no one else will really understand this work; hence he keeps it to himself. Copying a hacker's code is roughly the same as plagiarizing a novel or photocopying a painting; it is art to be admired, not stolen.

Of course, not every CS student is a hacker, but many of them still act in a similar manner. One plausible explanation might be that since the hacker is reluctant to show his code, other students act the same way in a sort of childish exchange. In my experience, groups of students working on projects are notorious for adopting a limited willingness to help anyone who won't help them. The hacker doesn't want to share his code for various reasons, but it has nothing to do with an unwillingness to work. This is seen by other group members as the hacker's being difficult, so their response is to do little or no work themselves. In the case I mentioned above, the hacker in our group was given free reign to implement the project how he wished, which seemed to placate his desire to show off. Thusly, he was very open with his code. In other groups, especially those consisting of individuals who have no real technical skill, the hacker will be difficult to get along with. In a corporate setting, most hackers will be surrounded by people they feel are their technological equals, so the sharing of work comes naturally.

In the same manner that the presence of corporations on campus and the lure of employment after graduation shape many students' methods of participating in group projects, personality conflicts and misunderstandings of programming goals further shape these methods. "Trade secrets" are often the result of communication problems, or lack of respect. Whether or not this can be changed is an interesting question. The aesthetic qualities most hackers associate with programming are either non-obvious or ignored by other students. Changing group dynamics might entail encouraging the discussion of programming as an art; it allows other students to see coding as more than the means to an end, the production of a product for monetary gain. Convincing some of the more pragmatic students might take more work than it's worth, although their awareness of the possibility could help group interaction.

IV. Licensing Concerns

I've talked quite a bit about Opensource software so far, but haven't spelled out how students really feel about it. While it should be at least semi-obvious by now that many students are decidedly anti-Microsoft and pro-Linux¹¹, they still use Microsoft products in overwhelming numbers. Microsoft likes to say that since they have a large market (their market is actually astronomical in size), their products are thought to be superior, not to mention liked. My experience has been nearly the reverse. The mediocrity of their products makes them appealing to use, at least in some sort of strange pragmatic sense. As I covered earlier, most businesses go with Windows on their desktops; Microsoft must make a product that can withstand the assault of an amazing number of users who have neither the skill nor inclination to use a technologically superior OS. The businessman, like I said earlier, just wants PowerPoint. He could care less that his operating system has been optimized to squeeze every ounce of power out of an old piece of hardware; he'll just get a new laptop if the current one runs slowly. Windows has to run on a billion different configurations, out-of-the-box. If Joe Average installs a video card he bought at Best Buy into his Dell, Windows must be able to cater to him. Linux requires a vast amount of configuration to get a video card to work; Windows is essentially idiot-proof. The requirement of the average businessman means Windows must be very robust. This robustness in turn lowers the overall quality. To use a previous analogy, Windows is a Ford, Linux is a BMW. Sure, a BMW is much nicer to drive, superior technologically, etc, but it's hard to buy accessories for it at Walmart. If all you need to do is drive a car to work, the Ford will fit your needs. In fact, it can be argued that a BMW really only

¹¹ I'm using Linux as the paradigm example for Opensource since it is so prevalent, and Microsoft as the paradigm closed-source OS for obvious reasons. When you read Linux think Opensource; when you read Microsoft think closed-source.

appeals to those who don't see vehicles as mere tools for transportation, but instead want a "driving experience". Linux offers a better "computing experience", Windows displays your PowerPoint slides.

So we have lots of students using Windows because it works; does that mean they like the licensing? Do they even like Windows? The answer to both of these questions is a definitive no. I mentioned that students can purchase Microsoft products very cheaply; Windows XP, for example, costs \$5. Even a poor graduate student can afford a product in that price range. The very low price of Windows negates any advantage Linux would have here (usually free). Add the fact that nearly any computer purchased from a major manufacturer includes Windows, and it's easy to see why there are so many installations on campus. But again, most CS students hate the licensing restrictions imposed by Microsoft. The learning aspect, the ability to read the source code and see how something was implemented, severely limits students. Linux, on the other hand, can be altered at will. If a student doesn't like how a particular function is performed, they can simply change the source code and recompile. Before the reality of such an action is questioned, I'll point out that many major advances in free software have been at the hand of university students working on pet projects. This customizability is essentially "hotrodding" for nerds. CS students don't work on cars, they program their computers. Windows doesn't allow students to do any sort of custom work beyond a certain point; the limitations of the licensing won't allow it.

One of the major aspects of doing custom car work is showing it off. Hackers¹² show off by distributing their code, as I stated before. If other hackers can see their code, they will be respected and praised. Closed licenses don't allow for the sharing of code, hence most hackers will use the GPL or BSD license to release programs they have written. While it might be wondered why a hacker would even care about licensing, as was explicated earlier, they guard their work judiciously. Not giving credit is the hacker equivalent to murder, the highest crime that can be committed. In the same way an author copyrights his work in order to protect the intellectual property involved, so too do hackers use licenses. Opensource is the only real way hackers can show off to friends and still keep the code theirs. Linux is essentially a collaborative effort of many hackers, all showing off their code and then accepting the input of others to make the code better. Car club enthusiasts frequently help each other to enhance their vehicles. Perhaps a custom exhaust installation is a bit over the head of an otherwise excellent mechanic, so he might call on a buddy who knows his way around an arc welder. The hacker might design user-interfaces better than anyone, but lack in areas like network coding. He might need a fellow hacker to help him complete some networking part of an otherwise amazing program. In a case like this, the sharing of information or code isn't seen as a threat, as the original hacker is still viewed as the "owner" of the program, although the help he receives is acknowledged very openly. Closed-source coding makes such an exchange impossible for the individual. Hackers almost always feel that Microsoft's general presence limits their abilities to produce code that can be shared. I think this feeling is based on some rather broad assumptions, but my experience with talking to students is

¹² I'll make a point here: hackers do not typically engage in illicit activities such as stealing secret government files or breaking into bank databases. These are the province of "crackers", individuals the typical hacker looks down on as a menace.

that they treat Microsoft's license philosophy with a religious hatred. Religion is a good word to use, as the zealotry encountered when speaking to some students is frightening. Microsoft is seen as the overbearing Goliath to the David of Linux. Bill Gates is a literal Satan who seeks to tempt the populace to use inferior products. The analogies are numerous, the feelings are real.

The notion that Microsoft is somehow evil is reflected in a controversial topic: software piracy. Because many students use Windows, but are disgusted with the notion of closed-source products, many of them "stick it to the man", or actively steal software. Windows is pirated by many students, even though they can purchase it for a mere \$5. Piracy of other software that operates on the Windows platform runs rampant. The cost of such software might be seen as a reason to copy it illegally, but how the software is actually used gives a different answer. I spoke with many students who simply enjoy "collecting" programs. They neither use nor care to use most of the software they download, but instead archive it into a collection, much like baseball cards. Software is traded amongst students, and the one with the largest assortment of programs, or who has a newly released program, is the king. When I asked why they even cared to collect software in this way, I got the sense that it was specifically to demonstrate that charging money for closed-source software is wrong. Evidence that supports this supposition is shown by the amount of respect Opensource projects are given. When Mandrake 9.2, a Linux distribution, was released, only a select few of Mandrake's paying customers were allowed to download a copy. Non-paying customers had to wait a period of one month before they could download the operating system. Software pirates openly admonished

anyone who gave away Mandrake ahead of time, instead waiting until the appointed date to download. Mandrake is seen as “one of the good guys”, since it is Opensource and provides a valuable resource to the hacker community. Copying Mandrake ahead of time would have perhaps taken income away from the company, something the pirates did not condone.

The students who professed to engaging in software piracy did not see much wrong with it fundamentally. They generally likened it to disobedience of an unjust law. By stealing proprietary software and giving it away to others, they hope to bring about the downfall of closed-source licensing. On an ethical level, many students **did** feel that what they were doing was basically stealing, and unacceptable. This dichotomy is reflected in the sense of reluctance to openly discuss piracy; political activists usually want to be heard, not remain anonymous. From a strictly utilitarian point of view, this makes perfect sense. By keeping the theft of software anonymous, pirates can affect the profit margins of major software companies, but avoid prosecution, which allows the pirates to further affect margins in the future. If we use a more reasonable test of the validity of the morality behind such actions, they become almost absurd. There is no real justification behind stealing software, which I think many students realize. The problem that occurs is they many of them have no real way of expressing their feelings on licensing in a productive manner. Perhaps this is what drives some more technically inclined students to create Opensource versions of popular closed-source software.

Licensing is both loved and hated: loved because it protects the individual from theft, hated because it limits further technical advancement. Of course, when speaking of licensing, it is fundamental to differentiate which sort of license we're referring to, i.e. open or closed. Many students have absolutely no stance on the subject, and in fact some are completely unaware that Opensource even exists. For the brightest programmers, however, licensing is a topic worthy of rigorous debate. It isn't that difficult to see the struggles many students experience when dealing with software development. Giving up their personal ethics in an effort to get a job might be a sort of social evolution where money takes priority over ideals. I did not speak with any students who had accepted jobs with large software companies, but some were worried that working for a company like Microsoft might jeopardize their ability to "show off". Again, for these students programming has aesthetic appeal and is not merely an exercise in producing a product. Taken from the view of an artist, restrictive licensing can be greatly limiting, severely weakening artistic expression.

V. A Personal Experiment

At the beginning of this paper, I mentioned that I took a page out of *The Cathedral and the Bazaar* by Eric S. Raymond and conducted my own personal experiment in order to gain some insight on Opensource. The primary reason for doing this was to avoid the interference a class might impose on forming views about software; if required to use something, students use it for fear of a bad grade. If a professor will only accept programs written with Microsoft Visual Studio, using the Eclipse IDE is a sure way to fail¹³. What follows is a detailed account of the installation and usage of various operating systems, including on from Microsoft and a number from the Opensource community. The conclusions drawn are as objective as personal preference can be.

The first thing I did for this experiment was to procure hardware; being relatively poor, I purchased the cheapest hardware I could get that was compatible with the operating systems I would be installing. I found a Biostar motherboard that had integrated video, sound and network adapter for \$45. A friend sold me a Pentium III-866 chip and 512 megs of ram for another \$60. I had an old CD-ROM lying around, and a flaky hard drive was given to me for a pitcher of beer. The only thing remaining for this lowball machine was a case, which I picked up second-hand for \$15. I borrowed a keyboard, mouse and monitor as I didn't see the necessity of purchasing these items immediately. After spending 30 minutes assembling the parts, I fired it up and was pleased to see that the system booted up into the BIOS. It was time to start the installations.

¹³ No professor I encountered was quite so adamant about what sorts of development tools they required, although there was a definitive pro-Microsoft feel in the classroom.

Since I use Windows XP at home, and Windows 2000 in my office, it should come as no surprise that I am very familiar with both of these operating systems. Windows is quite popular because it runs on just about any modern hardware, so I decided to start off with XP to test my piecemeal system. Inserting my XP CD, I started the computer and let it boot from the CD-ROM drive. After a few minutes, a text-based Windows installer asked me what drive I wished to install the OS onto. I selected the hard drive and let it do its thing. The installer copied some files to the hard drive, then rebooted the computer into a graphical installation mode. Roughly 45 minutes later, I was asked for my time zone, internet connection information, and a few other tidbits. The system rebooted itself then started up XP for the first time and asked me my name. After entering my name, I was presented with the generic XP desktop. Approximately one hour after I had begun the process, I had a working Windows XP system. The video worked, so did the sound and network connection. I could easily see why Windows was so popular with the average user: it was essentially idiot-proof.

Turns out I was mistaken. Mere moments after the system was up and running, I began receiving messages about an imminent shutdown. Hardware failure, perhaps? Having some experience with viruses and worms, I booted into Safemode¹⁴ and checked the logs. Sure enough, I had gotten the Blaster Worm, a malicious piece of software that reboots a Windows machine and seeks to infect other systems. I had to laugh about this; we always hear about viruses being a major problem, but generally ignore such remarks. In around one minute of being connected to the internet, my brand new system had been infected

¹⁴ Safemode is a method of starting the operating system without any drivers loaded, which allows a user to diagnose any potential problematic software.

with a virus. I used Google on another system to find out how to remove the virus, then immediately installed an anti-virus program. Then, I ran WindowsUpdate, a utility that downloads the latest bug fixes and upgrades for Windows. Three HOURS and five or six reboots later, my new system was fully upgraded and patched. The idiot-proof part of Windows seemed to be less of a reality. If I had been using a slow modem (I was connected via a cable modem) the three hour figure might have been closer to fifteen or twenty hours, essentially an all day affair. It took roughly four hours in total to get Windows completely installed and functioning properly, but no additional software besides a virus scanner was installed. To ensure the stability of the hardware, I did a burn-in test, a test where several programs are left running for a length of time to check for errors. After 24 hours, the system was still running strong, so it appeared the hardware was fine.

The next step was to install Microsoft Office. I inserted the CD and Windows automatically started the installation program and began installing after I selected a few options. After Office was installed, I created a new Word document with no problems. I downloaded AOL Instant Messenger, installed it without a hitch, and chatted with a few friends. Using XP wasn't difficult, and even if I had never used a computer I don't think it would be all that hard with Windows. The interface was nice but, except for some minor color variations, unchangeable. Personal preference took a backseat to uniformity; if I used any other Windows system, it would essentially operate exactly the same. Overall, I'd say Windows is a good OS once it has been patched and debugged properly.

The virus problem I experienced is a very common occurrence, and I'm unsure if the average user would be able to fix it alone.

Having used Windows as an almost litmus test for the operability of my hardware, it was time to install Linux. I selected Fedora Core 1 first, a distribution made available by Red Hat that uses entirely free software. I downloaded the ISOs¹⁵, burned them onto two CDs and booted off the first. The installer that Fedora uses is very nice, and purely graphical, unlike the initial installer used by Windows. The number of options was almost overwhelming at first. I was asked how to partition the hard drive; I used the default "Do it for me" option. Then I was presented with a number of software packages to install. I selected "Install Everything", as that seemed to be the fastest. 45 minutes later, the installer asked me to configure the video and network. Configuring X-windows was slightly confusing, and had I no prior experience in doing it I might be somewhat puzzled. The network settings were simple (DHCP, or automatically), as was setting the time zone. Then I was asked to set a root password, the password to administer the system. Selecting a user name and password was the final step before the system rebooted. The Fedora login screen appeared, and I logged in with the user/password combination I had selected. A prompt asked me which desktop I wanted to be default; apparently I had several options, including KDE, GNOME, IceWM and FVWM. Being somewhat familiar with GNOME, I picked it and was presented with a nice desktop.

¹⁵ An ISO is an image file, or a representation of a CD-ROM that can be distributed then "burned" onto a blank CD later.

After logging in, I was surprised at the number of programs Fedora had already configured and installed for me. There was a Microsoft Office clone called OpenOffice that was quite easy to use. The chat program I had used in Windows was not available, but again a clone was provided called GAIM. Most of the “productivity” applications I might require were available in the default install. The fact that this was all free software was quite astonishing. I logged out of GNOME and selected KDE, which presented me with a different desktop. Fundamentally they were very similar, but the functionality and manner in which these two desktops did the same task led me to believe that virtually any personal preference could be implemented if a user had the inclination to do so.

Problems began arising right about the time I saw the coolness of Fedora. The soundcard didn't work very well at all. I played an MP3 to test the sound quality; the popping and poor quality was enough for me to disconnect my speakers. Further, I wanted to change the screen resolution to something a bit higher. This was nearly impossible. The only resolution that worked was the one I had selected during the install (Windows worked just fine in the resolution I was trying to change to), and no matter what I did I was unable to correct this. Some searching on the internet gave me a solution: editing some text files. First, I had to login as root, the official “owner” of the machine. Then I used vi (an arcane editor, that while quite useful, is beyond the skill of any non-technical user) to edit the default .xinitrc file to reflect the allowable video modes. This required looking up some technical details about the motherboard on Biostar's website then translating these details into information Fedora would understand. Unfortunately, it STILL did not work,

and I was unable to get the resolution above 800X600, which is fairly low¹⁶. The sound problem turned out to be problematic, as various websites and usenet messages explained the issue was poor hardware implementation. If that's the case, why did the sound work fine in Windows?

While the customizability and sheer magnitude of software installed was nothing to scoff at, my inability to produce decent sound and have reasonable video resolution gave me a bad taste for Fedora; an average computer user would have no chance to fix these problems. It might be argued that I was using very inexpensive hardware, so I should have expected to run into some problems with it, but again, why did Windows work? Perhaps the proprietary nature of hardware drivers has a lot to do with that. Since hardware manufacturers do not explain many aspects of their products to maintain their trade secrets, they generally issue closed-source drivers for those products. Hence, Windows drivers are released in binary-only format. Linux drivers are produced by the Opensource community, using a sort of "best guess" or reverse engineering. The lack of quality drivers for Linux is reflected in the difficulty of getting an MP3 to play without popping. Still, I was impressed with the OS at an ideal level. It was apparent that the programmers who created Fedora were interested in quality and user choice, things that Windows lacks in some aspects.

I attempted to use another Linux distribution to see if some of the problems I had experienced with Fedora were merely associated with one implementation. Mandrake 9.2

¹⁶ I later found this to be an issue with the version of XFree86 I was using, the program that controls the graphics display. Fedora 1 had installed an older version of XFree86 that did not fully support my video card.

had been described to me as supporting the greatest number of platforms, so I borrowed the CDs off a friend and began the install. Like Fedora, Mandrake's installer was purely graphical (there was an option to use a text-based installer), and very simplistic. The number of installation options was significantly reduced, reminding me of Windows XP. Partitioning the disk was done automatically, and I selected "Workstation" instead of a collection of software packages like Fedora. At the completion of the install, I had to setup the video and network, just like Fedora. Root password, time zone, user name. Upon reboot, I was given a very nice KDE desktop, very similar to the Fedora desktop. Mandrake seemed a bit more user friendly at first glance.

The video resolution problem had really bugged me, so I tried to change it right way in Mandrake. No problem. I merely selected it from a list and the resolution was changed after I restarted KDE. I then played an MP3 and was surprised that the sound was actually decent, although still plagued with some popping. The number of applications was smaller than in Fedora, although OpenOffice and GAIM were both present. A flashing icon at the bottom of the screen revealed an automatic updater. After clicking on it, a utility similar to WindowsUpdate downloaded and installed the newest versions of software that had been released since Mandrake 9.2 had been made. This update program worked without any problems, but did take quite a while to download around 100 megabytes of updated software. Overall, my experience with Mandrake was more pleasant than with Fedora, but I think this had a lot to do with the support for my video card.

Moving away from Linux and to another Opensource platform, I went with FreeBSD. FreeBSD has the distinction of being derived from Unix, instead of a clone such as Linux. In theory, this means it is more stable and well-documented. In practice, Linux is nearly on par. The FreeBSD 5.2 ISO was downloaded and burned, then booted from in El Cheapo¹⁷. The installer was started and I began the process. To being with, the installer FreeBSD uses is the most difficult to understand program I've seen in a long time. I read approximately twenty pages of documentation in an effort to fully comprehend what the installation program actually required of me. It was completely text-based, which is not a problem in itself, but the options are non-intuitive and arbitrary. Consequently, I purchased a book, *FreeBSD Unleashed*, from Barnes and Nobles to assist with the setup and configuration of the operating system. I will not go into great detail, but FreeBSD asked me everything imaginable. The number of options was far beyond the scope of Windows or the Linux systems. After a few days of reading and playing with the installer, I was finally able to begin installation. It is strange that while the initial setup of the installer was quite complicated, the installation process itself was purely automated. Unlike Linux or Windows, there were no post installation steps. Configuring the video card was a painful process: there was no pseudo-automatic detection like that in the other OSes. Still, I was able to get it to work at the resolution I had wanted, unlike in Fedora.

After the initial installation, I was presented with a login screen, albeit in text. Unlike Windows or Linux, FreeBSD assumes that the user wishes to use the commandline only. There is something to be said about a pure text-based computing experience, but having a GUI, or graphical user interface, is much preferred most of the time. I logged in and

¹⁷ By this time, I had named the machine in much the same manner as one might name a boat.

typed “startx”, which told El Cheapo to load up the GUI. After a few moments, KDE started up, and I was in familiar territory. Much like Windows, very little software had been installed with the base system. Wanting to use OpenOffice, I began the installation procedure. The documentation explained that FreeBSD would automatically download and install various programs using a tool called pkg_add. I typed “pkg_add openoffice” at a command prompt and waited for the results. After downloading the package (which took some time as OpenOffice is quite large), El Cheapo output numerous messages, finally ending with “openoffice installed”. And, indeed, it was. It performed in exactly the same way as on the Linux systems, but a bit faster overall. Of all the OSes, FreeBSD was the fastest in general, while Windows was the slowest by far. In any case, I installed several more pieces of software using this simple command and had no real problems getting them to work.

While the video worked just fine in FreeBSD, the sound card produced nothing at all. Several searches on Google and the FreeBSD website revealed that I had to manually load the driver into the kernel. Essentially, I had to edit a text file that told the OS the sound card was present in El Cheapo. After editing the file and rebooting, the sound worked as promised, but again there was significant popping. Further, actions on the system that resulted in audible cues, such as clicking an icon, did not sync. There was up to a one second delay between the time a sound should have occurred and when it actually did. Not good. Overall, FreeBSD excelled in speed, but the installation was involved to say the least.

After the installation and usage of the aforementioned operating systems, I gained some rather interesting insights about Opensource, the primary being the reason for zealotry among Opensource proponents. The reason I believe there exists an almost religious fervency for Linux and its ilk is not due to the technical advantages, as to the casual observer none really exist (the sound card issue proves that). Instead, a user of Linux must be intimately involved with his operating system in order to get it to function properly. FreeBSD demonstrates this even further. Windows works right away for the most part, Linux requires some tinkering. The average Linux user spends time reading documentation in order to do things that Windows users do not worry about. While some of these things arguably make Windows superior, many of them are things that Windows cannot even do. The choice of what window manager to use is a good example. Customizing KDE to perform exactly how you want takes time; Windows XP works one way, and one way only. There is no need to define how certain keys should work in Windows, as these are standardized. KDE allows a user to create key bindings that are tailored to their tastes. I stated that the average user would have some difficulties with Linux, but that seems to come with the territory. Windows tries to be a tool while Linux and other Opensource operating systems truly try to be tools **and** an experience. The philosophical difference creates a boundary between closed-source and Opensource that is not easily discoverable except through personal experience.

VI. Conclusions

I covered a gamut of topics related to computer science, but the general theme was concerned with Opensource software and how it differs from closed-source, or proprietary, software. For the closed-source crowd, Microsoft and various other vendors create tools that are used to accomplish tasks useful in their careers. For the Opensource community, Linux and other projects create an outlet for creativity and to demonstrate talent. The question I had alluded to throughout this paper, and in fact the question the title implicitly asks, is how Opensource drives the software market. The answer is quite simplistic, but profound. Opensource is where the true innovation takes place, where the best programmers gather and exchange ideas, and indeed where many computer science students engage in learning. Many of these programmers go on to take jobs at Microsoft or other companies, but philosophically they still envision software as an art, not merely a means to accomplish a task. Whenever Linux creates an innovative way of doing something, Microsoft generally copies it. Many Microsoft programmers, having had experiences with Opensource software in the past and perhaps still holding onto their art, see Linux not as a threat, but as a source of inspiration. The problem that occurs is not that Windows copies Linux, but that Microsoft does not, rather cannot, give credit to Linux developers. Thus, the rift between the two sides grows because Opensource programmers see this as pure theft. A project like OpenOffice, obviously derived from Microsoft Office, credits Microsoft for much of the design, even if indirectly. Microsoft

gives credit to Opensource projects only when required, and in a manner not easily discoverable¹⁸.

The university further differentiates between Opensource and proprietary software, whether in lecture of the general manner in which those lectures are conducted. It would be literally impossible for an institution, Texas A&M in particular, to totally avoid doing so, and it may not even be a problem. As I've stated numerous times, the artistic merit of software design is often at odds with the pragmatic aspect, and making it apparent to students that a dilemma often exists when choosing the manner to produce software is a good thing. However, perhaps explicating this problem outright instead of vaguely touching on it would be best. A course in Software Philosophy, or maybe Software Aesthetics, could be a viable option.

One final remark in closing: Opensource is not at war with Microsoft. I believe that both must exist for the other to function properly. Without Opensource, Microsoft would lose much of its innovation. Without Microsoft, the Opensource community would devolve into a group of argumentative idealists. These programmers, the hackers, truly need to feel that they are fighting against some oppressive power in order to remain organized. In much the same way that many artists produce works of art to profess some unpopular political statement, the hackers feel that their software is political in nature. Throughout this paper I made it clear that Microsoft's dominance in the marketplace creates problems, which results in the Opensource community's defiance, i.e. their production of

¹⁸ "Based on NCSA Mosaic. NCSA Mosaic(TM)" – This message is displayed when looking at the information about Internet Explorer. Mosaic was the first web browser, and freely distributed.

competitive software. Perhaps the near monopoly that Microsoft has really isn't a problem after all. Without this monopoly, Opensource would not exist, so in a sense Opensource **requires** near overwhelming odds to be motivated. In the not-so-distant past, when the software industry was much more fragmented, programmers of free software were few, although this might have been the function of a much smaller market as much as the nature of that market. Still, free software tools were traded amongst hackers, and there was some resistance to the proprietary Unix systems of the day. As the market became more and more dominated by a single solution, free software become more important. The less competition, the more need there is for Opensource. Microsoft is a catalyst to Opensource, almost to their chagrin.