Evaluating Potential Open Source Applications:
A Case Study of the Network Attached Storage Industry

By

Martin Glynn

Submitted to the System Design and Management Program
In Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

At the

Massachusetts Institute of Technology

May 2007

© 2007 Martin Glynn
All Rights Reserved

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part.

Signature of Author

Martin Glynn
System Design and Management Program
May 2007

Certified by

Eric A. von Hippel
Professor of Management, Sloan School of Management

Certified by

Patrick Hale
Director, System Design and Management Program
Evaluating Potential Open Source Applications:
A Case Study of the Network Attached Storage Industry

By

Martin Glynn

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of Master of Science in Engineering and Management

The open source model for software development has been established as a legitimate competitor to the largest closed source software firms in the world. The purpose of this thesis is to investigate the potential impact that the open source model can have within the information storage industry.

The existing literature regarding open source is reviewed and used as a framework for analysis of the storage industry. Additional analysis of the FreeNAS open source storage project is also conducted using the framework to provide context.

The research concludes that the information storage industry meets many of the criteria that have been shown to drive adoption of open source software. Some speculation is made regarding the manner in which open source will continue to expand within the storage industry.

Thesis Supervisor:

Eric A. von Hippel, Professor of Management

Sloan School of Management
# Table of Contents

Introduction .......................................................................................................................... 4
A Brief History of the Storage Industry .................................................................................. 5
  The Early Days .................................................................................................................... 5
  The Arrival of General Purpose Hardware ......................................................................... 7
  The Software ....................................................................................................................... 9
  The Open Source Operating Systems ............................................................................... 10
  Innovation from the Bottom Up ........................................................................................ 12
  Conclusion ......................................................................................................................... 13

What is it that makes a closed-source software industry susceptible to takeover by open source software? .................................................................................................................. 15

Technical Factors .............................................................................................................. 17
  The “ilities” ......................................................................................................................... 18
  Peer Review ....................................................................................................................... 27
  Control ................................................................................................................................ 31
  Interconnectivity ............................................................................................................... 32
  Very Little to Be Gained ..................................................................................................... 33
  A Wide Range of Problem Solvers .................................................................................... 33
  Discussion ........................................................................................................................... 34

Economic Factors .............................................................................................................. 36
  Heterogeneous Needs ......................................................................................................... 36
  “Sticky” Information .......................................................................................................... 38
  “Toolkits” .......................................................................................................................... 39
  The Process of Innovation ................................................................................................. 40
  Nothing to Lose ................................................................................................................ 42
  Cost-Sharing ...................................................................................................................... 42
  Risk-Sharing ....................................................................................................................... 43
  Business Models ............................................................................................................... 44

Strategic Factors .............................................................................................................. 46
  Aligning Price with Cost .................................................................................................... 46
  Disruptive Process ........................................................................................................... 47

What does this mean for the future of closed-source software vendors in the storage industry? ......................................................................................................................... 50

Conclusion .......................................................................................................................... 69

References ............................................................................................................................. 72

Appendix A ............................................................................................................................ 73

Appendix B ............................................................................................................................ 75
Introduction

In January of 2006, I left the storage company at which I had worked for 3 years and entered the System Design and Management (SDM) program at the Massachusetts Institute of Technology. One of the first courses that I took was called “Innovation in the Marketplace”, taught by Professor Eric Von Hippel. Before I took this course, I had experience using open source software; I have education and professional experience in software engineering. However, up until that point, I had considered open source to be the realm of hobbyists and did not consider it truly competitive with the closed-source model of which I had been a part. Professor Von Hippel showed me the power of the open source model, both from a technical perspective and an economic perspective. At that point, I decided that for my thesis, to complete the requirements for the SDM degree, I would combine my experience in storage with a study of the open source model.

What follows is the result of this study. This includes a brief history of the storage industry, a literature review of open source writing, and an analysis of the future impact open source will have on the storage industry.
A Brief History of the Storage Industry

What do I mean when I say “The Storage Industry”? There are many different pieces of the information storage industry. I am focusing on storage systems, which are collections of hard disks, grouped together physically, that are controlled by a combination of hardware and software. Storage systems are essentially big computers with a lot of hard disks inside. Their basic purpose is to provide storage capacity to other computers. Since storage systems are basically big computers, they work the same way as other computers, such as a personal computer (pc). They have an operating system with applications that run on top of that operating system. This relationship is similar to the way that a pc for home use would have an operating system, such as Microsoft Windows, and applications, such as Microsoft Word or Microsoft Internet Explorer, that provide specific functions. In a storage system these applications help the storage system provide other services besides just storing the data, for example copying or backing up the data.

The Early Days

To understand the development of the storage industry I first have to provide some general information about computing. Today’s personal computers contain a central processing unit (CPU) that is capable of performing general tasks. These general tasks can be combined to allow the operating system and applications to perform any necessary higher level computational task. This means that almost all of the logic behind the
function of the operating system and applications is actually contained in the software that runs on the CPU. However, some tasks are very computationally intensive, such as the steps required to produce the graphics on the personal computer. If the CPU were used to produce the graphics, then it would have very few resources left to perform the tasks required for the operating system and applications. For that reason a specialized processor is sold with the personal computer, often referred to as a video card. The specialized processor on the video card will only perform the tasks necessary for providing graphics, which allows the creators to build the processor with specialized tasks that are purpose built for doing the computation required for providing personal computer graphics.

In a manner similar to the video card, the computing equipment once used inside storage systems was somewhat specialized. The early storage systems provided storage capacity to high capacity mainframe computers. The specialized hardware was necessary to produce the performance, such as the number of bytes of data that can be transferred per second, required by the mainframes that were connected to the storage system. However, despite the speed advantage, specialized hardware has some disadvantages. The primary disadvantage is the high cost of developing, fixing and upgrading customized hardware. Returning to the video card analogy, video performance on a pc will be much faster when one has a specialized piece of hardware for rendering the graphics on a pc. However when the video card manufacturers want to develop new functionality, or fix a problem in the field, it is much more costly.
In the early days of the storage industry, storage manufacturers were able to derive competitive advantage by creating customized hardware to generate performance advantages. This custom hardware is difficult to maintain and support over time. For example, if a bug is discovered in a custom piece of hardware, it is very difficult and costly to fix in the field. The hardware in question may have to be replaced at every customer site, which is a costly proposition. A second disadvantage of custom hardware is development time. When new or improved systems are developed, the process of manufacturing and then testing the specialized hardware is long and involved. These disadvantages were acceptable early on because they allowed the manufacturers to maintain a competitive advantage. The specialized hardware, with its higher performance capabilities, was also the only really feasible option, at that time, based on the performance requirements.

**The Arrival of General Purpose Hardware**

Overtime, the performance of general purpose hardware, processors that perform general tasks, has improved dramatically. This is commonly referred to as Moore’s law, which stipulates that processors double in performance ability every eighteen months. I would argue that the storage industry has reached the point at which the performance advantage of specialized hardware no longer compares to the cost advantage of using general purpose hardware.
When a storage vendor chooses to use general purpose hardware, they must implement the functions of the storage system in the software layer. The use of software has two primary advantages. The first is that when a bug in the system is found it can be fixed in the field, at customer sites, with a simple software patch. The cost of making copies of the software patch for each customer is negligible. With the advent of the internet, some patches can even be delivered automatically with no disruption to the customer and near zero transportation costs (in contrast to sending a technician on site). The second advantage is the reduction in development time. Without the need to manufacture new hardware, development cycle time is greatly reduced. New features can be developed in software and tested more easily.

An example of the advantages had by combining general purpose hardware with software level logic, versus that of using specialized hardware, is described in the book Rebel Code, by Glyn Moody. He describes the story of Transmeta, a company that built processors that could be modified at the hardware level using just software (how they actually did this is not necessary for this discussion.). As noted in the book, Transmeta experienced a problem with one of its chips in the field. To fix this problem in a standard CPU, the CEO at the time, Dave Ditzel, explained, “...would take weeks of fabrication time, testing, and shipping it to them.” Instead, “What Transmeta did was to send them a new CPU over the internet. In fact, we simply e-mailed it to them,” noted Ditzel. This is the power of software combined with general purpose hardware, and it is that power which the storage system manufacturers would soon hope to capture.
However, software based solutions also have their disadvantages. When manufacturers begin using general purpose hardware they lose the competitive advantage, and substantial barrier to entry for new vendors, that they developed using their own specialized hardware. They are forced to compete based on the software functionality they can develop and the service they provide, which is not necessarily an easy transformation for any company to undertake. The shift will involve changing the primary skill set of much of the engineering workforce. Nonetheless, many storage system manufacturers shifted to general purpose hardware and began competing based on software and service.

**The Software**

Returning to the idea that storage systems are basically big computers with a lot of hard disks, the software they run breaks into two functions: 1) the operating system and 2) the applications that provide specific functionality.

Whether or not the storage vendors were using customized hardware or specialized hardware they needed an operating system. Most of the early storage vendors built their own operating systems. One example is the Data OnTap™ operating system. Data
OnTap™ is an operating system developed by Network Appliance (NetApp) to run its NAS appliances.

Building your own operating system has advantages and disadvantages. The primary advantage is that you can customize the code to do exactly what you need it to do. This may allow you to provide better performance. The primary disadvantage is that you have to train and carry a large engineering organization to support it. This problem is larger than it may sound. The engineers required to build the operating system will eventually turn over, at which point new engineers will have to be hired and spend considerable time learning how the operating system functions in order to help maintain and enhance it. This kind of job is also undesirable for many engineers, as the extensive expertise they have developed in the operating system may not be transferable to work at other companies and will not be particularly marketable. Therefore, this could be considered a dead-end position.

**The Open Source Operating Systems**

What options does a vendor have besides constructing their own operating system? Until the mid-1990s the only other option was to license an operating system from another company. This can be quite expensive and ties the storage manufacturer to a single supplier. This is not a desirable position to be in. However, in the mid-1990s the open

---

1 This information was gathered from an interview published in the ACM Queue found here: [http://www.acmqueue.com/modules.php?name=Content&pa=printer_friendly&pid=378&page=1](http://www.acmqueue.com/modules.php?name=Content&pa=printer_friendly&pid=378&page=1)
source movement began to make serious progress and with it came an alternative to licensing an operating system: open source operating systems.

In the early days open source operating systems were not seriously considered for business applications. However, just as general purpose hardware did, over time the open source operating systems improved in performance. Also, as more people began to use these operating systems the systems became more robust generally improved in quality (this quality of software will be discussed in depth later in this paper). As this progression happened the attitude towards open source improved within large manufacturers in many different industries. Though it is not being advertised by the manufacturers at this point, it would make sense for many of them to shift to open source operating systems or at least derivatives of such. There is anecdotal evidence that this is happening already.²

There are several reasons that it makes sense for a storage manufacturer, in particular, to switch from their own internally developed operating systems to open source operating systems. The operating system offers functionality that is now basic to all storage systems. This is very similar to the transition that occurred between customized hardware and commodity hardware. Now that the open source operating systems offer the same levels of functionality that the manufacturers can develop in-house, they can no

² Wikipedia indicates that the latest version of the Data OnTap from NetApp is based on Freebsd:
 In 2007 NetApp became a premier donor to the FreeBSD foundation:
 http://www.freebsdfoundation.org/donate/sponsors.shtml
longer maintain a competitive advantage by developing an advanced operating system. However, if the storage platform consists of general purpose hardware, or commodity hardware, and an open source operating system, it will force the storage manufacturers to compete based on the software applications that they can offer. Additional competitive advantage can be gained through services supplied around the use and support of the storage system.

**Innovation from the Bottom Up**

The ability to develop a storage system based on commodity hardware and an open source operating system had significantly lowered the barriers to entry for innovation from areas other than the large manufacturers; or what I am referring to as innovation from the bottom up. There are two aspects to this innovation from the bottom up, the first is the arrival of entirely open source storage systems, and the second is the ability by small competitors to compete with the large manufacturers.

As previously discussed, a storage system is just a computer. A software engineer by the name of Olivier Cochard-Labbé had a personal computer he decided he wanted to use as a storage system. He could not find free software that would allow him to use his computer as a storage system, so he built FreeNAS. FreeNAS is software based network attached storage (NAS) solution that can be installed on any PC and convert that system into a storage system (commonly called a storage appliance). Amazingly, he was able to build FreeNAS even though he had no previous software engineering experience building storage products. This was possible because he was able to use existing open source
software projects to provide most of the basic functionality, such as the operating system and framework to develop a graphical user interface. The initial version took him only two days to create.³

The second area of innovation is one with which I have first hand experience. The company at which I worked before I began the SDM program was building a storage appliance that offered new functionality for storage users. Where does a startup begin, when it is handed funding and tasked with constructing a new storage appliance from scratch? We chose the shortest path to delivering the novel functionality of our new products. That meant combining general purpose hardware with open source software and our own proprietary layer of functionality. This type of rapid development would not have been feasible were we required to develop our own hardware or operating system. We were able to construct a highly-available, i.e. no single point of failure, storage appliance, which several medium sized businesses purchased. The fact that a startup was able to construct such a system demonstrates how the barriers to entry have been lowered through the improvement in performance and quality of commodity hardware and open source operating systems. These two factors will continue to enable new players to enter the storage industry with innovative new products.

Conclusion
Clearly open source software and the hardware upon which it executes is having a

³ In January 2007, FreeNAS was named the sourceforge project of the month. Details provided are taken from the profile created there: http://sourceforge.net/potm/potm-2007-01.php
dramatic impact on the storage industry. These two elements are driving innovation from
the top-down, within the major storage manufacturers, and enabling innovation from the
bottom up, with innovations by open source communities and startup storage companies.
This evolution has been happening for the last ten years and continues today. The
questions remains, will this evolution continue? Will open source maintain the initial
impact it has had and what additional impact can open source have? In the next section I
will discuss the general factors that allow open source to have an impact on any industry.
What is it that makes a closed-source software industry susceptible to takeover by open source software?

To develop an answer to this question, I will identify the factors, in any given closed-source software market, that allow the open source model to be successful. Once this set of factors has been identified, it can be applied as a framework to determine if open source will be successful at any given time and in any given closed-source software market. In the final section of this paper, I will apply these criteria to the Network Attached Storage (NAS) market and will attempt to make some predictions about the impact open source will have on that market.

Broadly speaking, I have identified three categories of analysis: Technical Factors, Economic Factors and Strategic Factors. Almost all of the technical and economic factors come from the two monumental works on open models of innovation, *Democratizing Innovation* and *The Cathedral and the Bazaar* by Eric Von Hippel and Eric Raymond, respectively. The exceptions are indicated below. The two strategic factors come from Eric Raymond and Clay Christensen. I've tried to provide enough technical detail to make the explanations clear, but at the same time it should not require a technical background to follow the arguments presented. Where possible, I have added case studies and my own experience to support the conclusions. In the final section of this paper, I will analyze the NAS market against the factors below. For ease of reference, I have numbered, cited and listed them here:
**Technical Factors**

1. “Reliability/stability/scalability are critical.” (Raymond, 2000)

2. “Correctness of design and implementation cannot readily be verified by means other than independent peer review.” (Raymond, 2000)

3. “The software is critical to the user's control of his/her business.” (Raymond, 2000)

4. “The software establishes or enables a common computing and communications infrastructure.” (Raymond, 2000)

5. “Key methods (or functional equivalents of them) are part of common engineering knowledge.” (Raymond, 2000)


**Economic Factors**

7. The users have heterogeneous needs that they each find valuable, which the manufacturer is unable to economically meet. (Von Hippel, 2005)

8. The information needed to develop the software is “sticky”. (Von Hippel, 1994)

9. The open source software meets the definition of a “toolkit” developed by Franke and Von Hippel. (Franke and Von Hippel, 2003)

10. The software users value the process of innovating themselves. (Von Hippel, 2005)

11. Competitors that have little to lose economically by open sourcing. (Von Hippel, 2005)

12. Users wish to share the cost of software development. (Raymond, 2000)
13. Users wish to share the risk involved in maintaining software over-time (future proofing). (Raymond, 2000)

14. The industry fits one of the identified open source business models.

**Strategic Factors**

15. The software pricing model does not align with the cost pattern. (Raymond, 2000)

16. Software vendors focus on meeting customer needs at the expense of process innovation (Christensen, 1997).

**Technical Factors**

In Eric Raymond's essay, *The Magic Cauldron*, which was published as part of his book, *The Cathedral and the Bazaar*, he identified five criteria for determining whether or not a software product should be developed and maintained under an open source model. He makes that case that when these criteria are met the open source model\(^4\) for developing software is more effective than the closed-source model\(^5\) for developing software. Those five criteria will be the subject of the next five sections of this paper. One way to frame these criteria is to ask, “Why would a user choose to use an open source product over a closed-source product?” One answer, based on the first criteria listed above, is that a user should choose an open source model whenever a user finds reliability, stability and scalability to be critical. What follows is a deconstruction of each of these points and the

---

\(^4\) In Raymond's metaphor for software development the open source model is represented by the Bazaar.

\(^5\) In Raymond's metaphor for software development the closed source model is represented by the Cathedral.
The “ilities”

1. Reliability/stability/scalability are critical. (Raymond, 2000)

Why does Raymond believe that if reliability, stability, and scalability (referred to as the “ilities” henceforth) are critical to a software product, then open sourcing that product is advantageous? The answer to this lies within his argument that there are inherent advantages to an open model for product development versus that of a closed model. As an open model for product development, the open source model has several basic tenets beyond the simple idea that the source code is visible and free.

The basic tenets of the open source model are: 1) source code is freely available under one of the legally-binding recognized open source licenses, 2) communication between users and developers is open and visible to anyone, and 3) updates to the source code are released frequently (“release early, release often”).

In order to be certified by the non-profit Open Source Initiative (OSI) as an open source project, a piece of software must be released under one of the OSI approved licenses. The “gold standard” of these licenses is the GNU General Public License (GPL). Of the 142,078 open source projects hosted on sourceforge.net, a widely used platform for open

---

6 Of course, these technical factors are closely related to the economic factors which follow, however, to make the arguments as clear as possible, I will initially discuss these criteria from a purely technical standpoint.

7 www.opensource.org

8 www.opensource.org/licenses/
source projects, 59,703 or roughly 42% use the GPL. For a license to be approved it must meet the definition of open source created by the OSI. Some licenses offer more flexibility than others, but they all meet the basic requirements of the definition. The definition has ten criteria, each serving a specific purpose. Some of the criteria prevent discrimination on any grounds, while others are included for purely legal reasons and ensure that the licenses have no loop holes. However, it is the first three criteria that are critical to the technical aspects of the open source development process: 

1) the source code must be freely distributable by anyone, 
2) the source must always be available with any distribution, and 
3) the source must be modifiable by anyone.

The open source licenses that must meet these criteria therefore serve two critical purposes that create a protected and open model for users to contribute too. The first purpose is to prevent any entity from incorporating the source code into some other product without making the source freely available (or very cheaply available if for some reason freely downloading over the Internet is not an option). This protects the developers and contributors to the open source project from having their work unknowingly profited from by a third party that incorporates their code and then releases only binary versions of a product. It also ensures that the original creator of the open source project is given credit for his or her work, even though the credit may not be advertised, in the traditional sense, the developer will recognize his or her work and, perhaps more importantly, other developers will recognize his or her work (we'll come

---

9 www.opensource.org/docs/definition.php
10 A binary version of a software product can be run by the user, however the source code cannot be seen. Closed source software companies only release binary versions of their products.
back to the topic of peer recognition when we examine the economic factors behind open source). The second, and equally important purpose of the licenses that meet the OSI definition, is to prevent the creator of the open source project from closing the source. That is not to say that someone could not build a closed extension or new feature onto an open source project, but the original open source code can never be re-closed or prevented from being freely available. This clearly protects the users from ever losing access to the software, or being charged for its use, at some point in the future.

There is second less obvious benefit to this purpose is that it allows anyone to “fork” the project when he or she so choose. “Fork” is a term used in the software industry when the source code for a given piece of software is copied and two separate versions of the software are developed going forward.\textsuperscript{11} There are a lot of reasons that a software product can be “forked” and it is an extremely easy process given the ease with which digital information can be copied. If the users of open source product disagree with the direction that the current developers have taken with the project they can simply “fork” the project and begin a new version. There will need to be new developers to maintain the new project, but often times the users of open source are capable developers themselves and can undertake development if they so choose. One can imagine that software users that are initially happy with a product, be it closed or open source, may overtime become dissatisfied with a piece of software if the developers, or company that distributes closed software, are unresponsive. In a closed software model, users are forced to find a replacement piece of software, or build their own from scratch. Open

\textsuperscript{11} More information about “forking” can be found here: http://www.bellevuelinux.org/project_fork.html
source users are protected from this scenario, because they can “fork” the product and build onto the original version that they were happy with.

One example of a “forked” project is the XFree86 project. XFree86 was the dominant X Windows System implementation from 1992 to 2004. The X Windows System is the underlying protocol that provides a graphical interface on almost all Unix-like systems (including Linux and FreeBSD). XFree86 was the open source implementation of the X Windows System that was distributed with almost all versions of Linux and FreeBSD for 12 years. Then, in 2004, the core developers of the project decided to change the license under which the code would be released. The justification provided for the new license was that it would ensure that the core members of the technical team would be given proper credit for their work. This prompted some lively debate within the open source community. The primary complaint was that the new license did not appear to be compatible with the GPL (it is also not certified by the OSI, though it was the conflict with the GPL, which is more strict than the OSI definition, not a discussion of whether or not OSI would certify the new license, that seemed to cause all of the problems). What was the end result? The last version of XFree86 released under the original license was “forked” and became the first version of a few brand new open source X Windows System implementations. X.org quickly became the new dominant X Windows System.

---

12 www.xfree86.org
13 The new XFree86 license can be found here: http://www.xfree86.org/legal/licenses.html
14 Some of the debate surrounding the XFree86 license change can be seen at the following sites:
   www.xfree86.org/pipermail/forum/2004-February/003945.html,
   www.xfree86.org/pipermail/forum/2004-February/003974.html,
   www.linuxtoday.com/developer/2004021803026NWDTLL
implementation and was adopted by the major Linux distributions. Anecdotal evidence also indicates that nearly all of the XFree86 developers migrated to X.org (with the obvious exception of those developers that were behind the license change).\textsuperscript{15} While this kind of "forking" and disagreement may not occur frequently in the open source world, it is a powerful option made available by the licensing system that is a part of the open source model.

One final note concerning the Open Source License: the licenses require a legal system that can enforce such restrictions, without it the open source model cannot function in the same form that has flourished thus far.\textsuperscript{16} The licenses manage to protect both the work of the developers and the contributions by the users, as well as the long-term availability of the software to both users and developers. The legal licenses ensure that the open model remains open and allows all those who participate, whether as users or developers, to do so without fear that the model will someday close, robbing them of value they have helped create. This is critical for the creation of an open source community, which directly relates to the development of the "ilities" for a software product. The following will demonstrate how this occurs.

How is the community built and maintained over time? This is where the second tenet,
open communication, which means that almost all communication concerning the project is done using written messages (e.g. email or discussion forums) that are visible to anyone and permanently archived, plays an important. Open communication, unlike the licensing described above, is not a requirement for becoming a certified open source project, however most successful projects practice it in some form. This transparent communication occurs between the primary project developers and the users and the frequency, tone, and nature of this communication is what characterizes the development of any one specific open source project. The strength of this community of developers and users, especially its core developers, will have a strong impact on the success of the project. For this reason, successful projects usually have core developers who are responsive and cultivate the community of users. The members of the community will each bring different skills to the development process: the primary developers have a deep understanding of the code and control what changes to the code are added to the recognized distribution, other users will also have an understanding of the code (because they can look at the source), yet other users will have expertise in areas that are relevant to how the software is used, and finally, some users will be long-time users of the product while others will be novices.

So how does this community of users, enabled by the licensing system and developed through open communication, help to improve a product's “ilities”? First, a diverse set of users will subject the software to different environments and uses, especially if the group continues to expand over time. This process of testing the software in varied ways is
really the only way to improve the “ilities” of any piece of software, since any software product of reasonable complexity cannot be “proven” to work properly. By exposing the software to different environments and uses more and more issues will be exposed, and, as these issues are fixed, the software will become more reliable and stable. Second, because this community has access to the source, it can often speed the process of fixing issues by actually finding the incorrect source code themselves (again, the source code is visible) or by simply narrowing the number of possibilities. This is summarized with an observation often attributed to Linus Torvalds, the creator of Linux, that “Given enough eyeballs, all bugs are shallow” (Raymond, 2000). The implication of this statement is that, with a large enough community of users (some of whom will be capable of reading the source code themselves), any bug will be easy to fix, because there will be someone who has the right frame of mind or set of experiences for whom the problem will be obvious.

Yet, the benefits of the community with respect to the “ilities” cannot be had if the software is released every six months, or three months, or even three weeks. For the beta-testers to continue exercising the product, they need the issues they found fixed and fixed quickly. Hence, the third tenet of the open source development model, the need to release early and often, the third basic tenet of the open source software model. This tenet, like transparent communication, is not a requirement for an open source project, but it seems to be one in practice with the successful projects. It is not uncommon for projects under heavy development to have daily or even twice daily releases. This has
the added benefit of keeping the community excited and involved (Raymond, 2000).

So what happens when a project is not “released early and often”? There are many examples of this from the closed-source world, most recently Microsoft’s Windows Vista operating system. However, there may be an equally long suffering example from the open source world. The Perl programming language is an oft cited example of a successful open source project (Lerner and Tirole 2002, Von Hippel and Krogh 2003).

The development history seems to follow the “release early, release often” mantra with releases at the very least every few months from Perl 1 to Perl 5. As indicated above, this approach was widely successful. Then, in 2000, the creator of Perl, Larry Wall, decided that in order to implement Perl 6, the next version of the language and the accompanying interpreter, the entire Perl implementation would have to be rewritten. It is now 2007 and there has yet to be an alpha release of Perl 6. Anecdotal evidence found on the web and, more importantly, discussions with veteran Perl users indicate that the Perl community is no longer excited about Perl 6. It is also no longer clear how widely adopted Perl 6 will be when it is finally released. Many users may continue to use and make contributions to Perl 5 instead of switching to Perl 6. Some might counter argue that rewriting an entire programming language, such as Perl 6, will inevitably take

17 The history of the development of Perl can be seen at http://search.cpan.org/~jhi/perl-5.8.0/pod/perlhist.pod.
18 The announcement concerning Perl 6 can be seen here: www.perl.com/pub/a/2000/07/perl6.html
19 There are bits and pieces of Perl 6, such as the Parrot runtime environment that will play a role in the final release, but none of these pieces represents an alpha version. Latest status can be seen at: dev.perl.org/perl6/status.html
20 Evidence on the web dates all the way back to 2004: http://www.oreillynet.com/digitalmedia/blog/2004/07/oscon_worried_about_perl_6.html. Discussions about Perl 6 were held with two individuals that have been professional Perl programmers since the mid 1990's.
a long time and this is the exception to the “release early, release often” rule. In response to that, I would point to two projects, Freenet and Linux. Freenet is a software product that enables decentralized publishing of content with the goal of preventing censorship. In their examination of the Freenet project Von Krogh et. al. indicated that the release of the initial code, despite immaturity and defects helped to build a community around the project (Von Krogh, 2003). The Freenet project today has 61 registered developers on SourceForge.21 As discussed up until this point that community is critical to the advantage that open source possesses over closed source. The Linux operating system is an even stronger example of the power of releasing early and often. Linus Torvalds posted the first revision of Linux after just 6 months of work (Lerner and Tirole, 2002). At the time he was a graduate student, hardly an experienced software engineer. He was building an operating system, which is arguably as complex, if not more, than a programming language. Only time will tell if Perl 6 is able to recapture the community that Perl 5 was able to build. The beauty of the open source model is that the community that formed around Perl 5 can continue to use Perl 5 indefinitely. If it is valuable enough they can build their maintenance releases and effectively “fork” the project. No single entity has the power stop supporting the software, such as Microsoft has done with earlier versions of its Windows Operating System.22

To summarize, I have described several ways in which the open source model for software development holds technical advantages over the closed source model. This is

21 As seen on sourceforge.net.
22 An example of this can be found at: www.microsoft.com/windows/support/endosupport.mspx
achieved through three basic tenets of open source software, licensing, open communication, and frequent software updates. Returning to Raymond's original point, this gives open source an advantage, specifically with respect to the “ilities” that any given software product develops over time. In the following section, I will examine further technical advantages created by having a community of open source software users.

Peer Review

2. Correctness of design and implementation cannot readily be verified by means other than independent peer review. (Raymond, 2000)

Again, why does Raymond believe that open sourcing is advantageous when the above condition exists? The answer here is simple. If the design or implementation of a piece of software can be verified using some method other than peer review, such as model abstraction that can be verified automatically, then the benefits of the large community and all their “eyeballs” looking at the design is lost. As a program becomes more complex, and almost any program of any value is complex, it becomes more and more difficult to verify the design or implementation using some form of proof or automatic tool. Strides are being made in this area, such as the Alloy tool developed at MIT by Daniel Jackson. Alloy is a tool that allows for the automatic verification of the design of a piece of software, using abstractions of the design model. It has been used to find issues with some very complex software products, though at this point it has not been widely adopted by the software industry (Jackson, 2006). Alloy also cannot verify the
implementation of a piece of software, i.e. the actual source code; it only verifies the
design model. The two most widely used methods for verifying both the design and the
implementation are peer review and testing.

At this point I think it is useful to combine the first two of Raymond's points together
logically and examine why the open source model has advantages over the closed-source
model. To develop a piece of software that is reliable, stable, and scalable, it is very
important for the design to be correct. In addition to a correct design, the
implementation, the actual source code, must also be correct. There are two methods for
verifying correctness of the design and implementation: peer review and testing. In the
open source model, the number of potential peer reviewers is much greater than in the
traditional closed-source model. In the open source model, the number of potential beta-
testers who are willing to provide detailed feedback is also much greater than in the
traditional closed-source model. In the open source model, the quality of the feedback
from these users has the potential to be much more effective because users can help to
diagnose any issues they see using the source themselves. Of course, all of these are only
potential advantages; they depend largely on the quality and size of the community
involved.

Developing an involved community around a piece of open source is enabled and
facilitated by the basic tenets I have described thus far: the licenses that protect both the
developers and the users, the transparency of communication and the frequent releases.
The protection from the licenses ensures that everyone involved knows that their contributions will be protected. The safety of this environment cannot clearly be duplicated by the closed-source model. In the closed-source model, the vendor needs the same contributions from the users, but once the information is received it will be used to increase the sale value of the product to the same users who helped to build it. The users clearly recognize this and usually require some form of incentive to spend any significant time helping the vendor. Over time this may or may not be cost effective for the vendor. Also, again because the users do not have access to the code, their contributions will be more superficial.

The second tenet, transparency of communication, helps to create a meritocracy in which all ideas are examined based on their merits and then acted upon with the understanding of all involved. In a closed-source model, the vendor is motivated to protect the value of the software it is selling, and as such, will try to prevent competitors from understanding the ideas that are going into building that software. However, closing the process like this opens the door for inefficiencies that may be driven by biases, personal agendas and preferences. That is not to say that open source projects do not suffer from the same inefficiencies, but those inefficiencies will be much shorter lived; as Supreme Court Louis Brandeis once said, “Publicity is justly commended as a remedy for social and industrial diseases. Sunlight is said to be the best of disinfectants.”

Lastly, the frequent releases have two effects. The first is to keep the community

23 The quote can be found at: en.wikipedia.org/wiki/Louis_Brandeis
involved and excited. Of course, this, in and of itself, may not be an advantage over closed-source vendors, because they too could release often with incremental new features. Some closed-source vendors have pursued this model. To drive revenue they only charge for major releases with large new features. However, the second effect is to ensure that most bugs are found and, hopefully, fixed rapidly. As discussed previously, most users of closed-source software will not freely use frequent new releases while at the same time dutifully reporting on new bugs without receiving some return compensation. For that reason, a model with user-reported bugs is likely not cost-effective for a closed-source vendor. Thus, most closed-source vendors hire a quality assurance team to act like customers and find the bugs before the customer gets the product. However, the quality assurance team will be limited by two factors. The first is size. For cost reasons, there simply will not be enough quality assurance people to make sure that the product is tested in every, or even most, reasonable situations.

The second, and more important reason, is the problem of “sticky” information. The term “sticky” information was coined by Eric Von Hippel in 1994 (Von Hippel, 1994). “Sticky” information is information that, for various reasons, is difficult to transfer from one location to another. The net effect is that because the information is difficult to transport, not all of it is, and thus the solution that requires the information is likely to be lacking in some way. The “sticky” information problem faced by the quality assurance team understands how the user will actually make use of the product. I have first hand experience with this problem as I spent several years in my career as a quality assurance
engineer for several closed-source software companies. One aspect of how the user makes use of the software is the computing environment into which the user will incorporate the software. There are a very large number of combinations of patches and conflicting softwares that could impact the software and produce bugs that are otherwise irreproducible. The sheer amount of possibly relevant information in each user's computing environment causes the information to be difficult to capture, making it "sticky". In the open source model, the beta-users will actually test the software themselves. Combining this with the fact that they have access to the source gives them superior debugging capability. The end result is software that is more reliable and stable in exactly the way it should be: as it functions with respect to the customer environment.

In analyzing the first two points behind Raymond's criteria for open-sourcing software, I have discussed many of the concepts that underlie the open source model and its advantages. Progressing through the remaining points will happen more quickly with this foundation.

**Control**

3. The software is critical to the user's control of his/her business. (Raymond, 2000)

Again, why does Raymond believe that there are advantages to open sourcing software that are critical to the user's control of his/her business? From a technical perspective, software controls access to a resource; that resource is usually information of some sort, or the actual function that the software performs. When software is closed-source, the
company that makes the software can effectively control access to the resource or the function for which the software is responsible. The company can do this by refusing to perform maintenance or fix bugs over time. If there is no method for the user of the software to extract the resource in question, he or she is effectively held hostage by the software vendor. This is no problem for the user if the resource in question is not critical to the business. However, the more critical the resource, the more concerned the user will be about the potential for the closed-source vendor to take advantage of its controlling position. If the software is open source, the user will almost certainly have other options. He or she can choose to perform the maintenance or bug fixing work him or herself without the help of any vendor, or he or she can simply add a feature to extract the resource from this particular software product and transfer it to another. As such, any vendor that is distributing an open source option will have a significant technical advantage over the closed-source vendor. In this sense, the fact that the software is open source is a feature, like any other.

**Interconnectivity**

4. The software establishes or enables a common computing and communications infrastructure. (Raymond, 2000)

When software is part of a larger communication infrastructure, or it is highly interconnected, then it must be able to communicate with other software and devices. Raymond's argument is that when users are able to examine the source code, they can ensure that no vendor is attempting to build-in proprietary extensions to an otherwise
open communications system. If that were the case, then a single vendor might be able to
gain control of a large network of systems. Open source code helps to prevent this by
allowing the users to examine the communication code. This is especially important
when different users are attempting to connect systems between each other. The
cooperation between users requires a certain level of trust open source simply reduces the
amount of trust required.

**Very Little to Be Gained**

5. Key methods (or functional equivalents of them) are part of common engineering
knowledge. (Raymond, 2000)

When the key methods that constitute a piece of software are already understood by a
wide range of engineers, there is little to be gained by trying to sell software that
implements these methods. The reason is that it is not difficult for competitors to develop
the same product, thus there is no competitive advantage derived from hiding the source
code. Raymond's argument is that there is much to be gained by open sourcing, as
described above, and nothing to be gained by remaining closed-source. Therefore it
makes sense to open source the code.

**A Wide Range of Problem Solvers**

6. The software can benefit from a wide set of potential problem solvers. (Von Hippel,
2005)

Von Hippel makes the argument that some products benefit from having a wide set of
potential problems solvers. One example he provides actually comes from somewhere other than the open source world: kitesurfing. By opening up the design of the kites to designers outside the traditional manufacturer's, kitesurfers were able to take advantage of the relevant knowledge possessed by experts from other industries, such as aerospace (Von Hippel, 2005).

Discussion

I have made the case, citing the work of others in this field, that there are certain types of software products for which the open source software development model holds technical advantages over the traditional closed-source model. Thus, any industry that produces software that meets some combination of the six criteria identified above will find that the open source model will present a superior product and stiff competition to any closed-source vendors. But, even if you have agreed with the arguments thus far, you are still left wondering why an open source project springs to life and then, on top of that, how the community necessary to make the project a real success remains involved. Especially since it is now clear how those involved will be economically compensated! Why don't the developers and users of these products simply buy the equivalent version from a closed-source vendor and save themselves all the work of beta-testing, coding, reading other developers code, submitting bugs, making feature requests, etc.?

The benefits of the open source model sound almost too good to be true for those who have developed closed-source commercial software. Communities where users download, use, and dutifully report bugs with beta software even help debug that beta
software on a regular basis without being paid for it. If you have developed software commercially, this sounds a bit like fantasy land. I have developed software commercially and I probably would not have believed such communities were possible, if there were not vibrant examples of such communities already. The blueprint described above was not written down and then formally adopted by an open source organization. If historical accounts, such as those provided in the Cathedral and the Bazaar, are accurate, this model formed organically based on the needs of the original Linux development project. I will get into why these communities actually form and exist in the next section when I discuss the economic factors that make an industry susceptible to open source.

Finally, perhaps the most inspiring aspect of the open source model this is the action and open discussion that occur, like that which led to the “forking” of the XFree86 project. I would suggest that it is the strength of this community that will continue to power the Open Source movement as a whole. In the event that any single element, even one as influential as the OSI, was to compromise the interests of Open Source, I believe that the community would not stand for it and would take steps to prevent or workaround the issue. The following quote from Linux Torvalds sums it up nicely:

*The power of Linux is as much about the community of cooperation behind it as the code itself.* If Linux were hijacked—*if someone attempted to make and distribute a proprietary version*—*the appeal of Linux, which is essentially the open-source development model,*
would be lost for that proprietary version. (Torvalds, 1999)

**Economic Factors**

The analysis of the economic benefits of open source over closed-source, and, following from that, what makes a closed-source industry susceptible to open source, can be broken into two parts: what economic reasons does a user have for choosing an open source product, and what economic reasons does a software developer have for open sourcing its software? In the last piece of this section, I will quickly examine economic models to create a sustainable business around an open source product, as that may have an influence on how quickly any given open source product can overtake a closed-source industry.

What economic reasons exist for a user to choose open source software over closed-source software? The first answer to this question has to do with how well the existing solutions (either open source or closed-source) meet the needs of the user.

**Heterogeneous Needs**

7. The users have heterogeneous needs that they each find valuable, which the manufacturer is unable to economically meet. (Von Hippel, 2005)

As Eric Von Hippel described in his book, *Democratizing Innovation*, one of the first reasons for choosing open source will be a function of how heterogeneous the needs of the users are and how well the manufacturers can meet those needs (Von Hippel, 2005).
Von Hippel makes the case that every user faces the make or buy decision and will make a choice based on whether the solutions that can be purchased meet their needs. Most importantly, if the manufactured versions lack features that the user needs, the user will calculate the value (calculate is used loosely here) of those features. If the value of the missing features compares favorably to the effort required to build a solution that includes those features, than the user will choose to build. To prove this point, let's examine an extreme scenario. Suppose that every user in an industry had completely different, absolutely heterogeneous needs. The manufacturer would have to make different versions of his or her product for each and every user, which is hardly a profitable business model. Therefore, the manufacturer will search for users that have a subset of common needs for which a product can then be manufactured and easily duplicated, a much more profitable scenario. Obviously, most industries fall somewhere on a spectrum that includes some heterogeneity of user need and some ability by the manufacturer to meet different sets of needs. So, how many user needs can possibly go unmet? Most manufacturers work hard to make their customers happy, right? This may be the case, but even with that Franke and Von Hippel found that once a market has been segmented “fully 50% of the total variation in customer or user need is typically left as (unaddressed) within- segment variation” (Franke and Von Hippel 2003). This inability by manufacturers to meet all the needs of their users opens the door for open source. The advantage that open source holds is that it provides the ultimate in flexibility: it can be modified in whatever manner the user desires.

The Apache Webserver serves as a good example of the benefits that users can obtain
when they have the power to make their own modifications. Franke and Von Hippel conducted a survey of technical Apache users to determine their levels of satisfaction. They concluded that those users who made their own modifications to the software were more satisfied with Apache as a whole (Franke and Von Hippel 2003).

"Sticky" Information

8. The information needed to develop the software is "sticky" (Von Hippel, 1994). Compounding the fact that users' needs often vary is the fact that the information required to develop a solution to user needs is often "sticky". The concept of "sticky" information, coined by Von Hippel, was first described above with respect to testing software the way users will use it (Von Hippel, 1994). My use of "sticky" in this economic factors section refers to the fact that much of the information needed to fulfill a customer need is not easily transferable from the customer to the manufacturer. There are many reasons this information is "sticky". One reason that is especially applicable to the development of software is the problem of too much information. Simply put, modern computing environments are complex, and it is difficult to capture all of the necessary requirements. Some might argue that this is simply a process issue, but it is often difficult to recognize which information is actually relevant to the customer requirements (Von Hippel, 1994) especially when developing new features or functionality with no previous comparison point. This can be magnified by the amount of "tacit" information (Von Hippel, 1994), or information that is not explicitly stated.

anywhere (such as that information stored only in the mind of the system administrator).

The open source model for innovation holds an economic advantage over the closed-source model: the information does not need to be captured by the “vendor”; the user has it all and is free to make the innovative changes he or she sees fit.

“Toolkits”

9. The open source software meets the definition of a “toolkit” developed by Franke and Von Hippel (Franke and Von Hippel, 2003).

And yet, not every open source project has the same characteristics. One of the key determinants in the user's decision-making process will be how much work needs to be done on the open source project to make it meet his or her requirements. There was little success after the Netscape browser was initially made open source as the Mozilla distribution. Jamie Zawinski, one of the early Netscape employees and Mozilla contributor, famously pointed out that “you can't take a dying project, sprinkle it with the magic pixie dust of ‘open source,' and have everything magically work out”. Raymond speculates that one of the reasons for Mozilla's early struggles was that the initial shipment did not allow users to easily run and see the browser work. The software at that point still required a proprietary license for the Motif library (Raymond, 2000). I would suggest that this falls under the more general guidelines developed by Franke and Von Hippel regarding the qualities required for a user “toolkit”. Franke and Von Hippel make the case that, in order for a user to benefit from the ability to modify a product

25 Franke and Von Hippel,
26 This quote can be found at: www.jwz.org/gruntle/nomo.html
(open source or otherwise), the product “toolkit” should have several qualities. These include: the ability to execute complete cycles of experimentation (trial and error), user-friendliness, libraries with useful and reliable components, and information about how to make a production version of product.\footnote{This last quality is included to make sure that users don't make versions of the product using the toolkit that cannot be produced by whatever production means exist. This particular requirement is probably more important for physical products, however it is still a necessary requirement for software.} When an open source software product possesses these qualities, the product will be more easily adopted by users and will pose a more significant threat to any competing closed-source vendors.

\section*{The Process of Innovation}

10. The software users value the process of innovating themselves (Von Hippel, 2005). Some users simply value the process of innovation. The open source model provides an excellent platform for innovation. The value of the innovation process to a user is certainly hard to quantify, but it is often provided in surveys of open source contributors (Von Hippel, 2005). I am not sure that any given industry can quantify whether its users will shift to open source because they desire to innovate themselves, however it does seem likely that certain environments (such as academia) will place great value on this capability.

At this point, some brave readers that have made it thus far are wondering aloud, what about the fact that open source software is free and closed-source software costs money? Shouldn't that be considered an economic factor for why a user would choose open source? While this is a factor, at least initially, I don't believe it is an important factor for
any software that will be used for a significant period of time. Raymond demonstrates that it is often the case that 75% of the cost of a piece of software is incurred in the maintenance and bug-fixing that occurs after the initial purchase. This makes sense, as one of the primary arguments that closed-source vendors make against open source competition is that the total cost of ownership (TCO) between a closed-source solution and an open source solution are closely matched. This argument states that the cost of maintaining the open source software overtime is actually greater than the upfront cost of a piece of closed-source software. I will not attempt to take either side of that debate, but the argument made by closed source vendors seems valid enough to prevent upfront cost from being one of the primary deciding factors of any savvy software user.²⁸

The flip side to the question of why a user chooses open source asks why a user or vendor would make a piece of software open source. (Again, at this point we're only concerned with the economic benefits, the technical reasons have already been discussed above.) When it makes sense for a single user or vendor to open source a piece of software, then an entire industry will be forced to compete against that open version and all of the advantages that accompany the opens source model. Also, it is important to keep in mind, as Von Hippel notes in *Democratizing Innovation*, that it may be the vendor or user with the least to lose that takes advantage of such a situation.

²⁸ Raymond makes an additional related point, that the popular open source licenses make it difficult for anyone to capture "sale-value" with an open source piece software. Therefore any industry in which a competitive product is open sourced will have to compete against this virtually "free" product (with respect to upfront purchase price) and it will not be possible for anyone within the industry to package up and sell the open source product.
Nothing to Lose

11. Users or vendors have little or nothing to lose economically by open sourcing. (Von Hippel, 2005)

This was clearly demonstrated when Netscape made its browser open source. Netscape was losing market share and could no longer effectively make money selling its browser due to the intense competition from Microsoft, which was freely distributing its closed-source browser. Lacking the same distribution method, i.e. the Windows operating system, to distribute their own browser, Netscape made the decision to open source their browser, now known as Firefox. This forced any company in the browser industry to compete against the open source version released by Netscape.

Cost-Sharing

12. Users can share the cost of software development. (Raymond, 2000)

The open source licensing system provides a safe environment in which multiple users may band together to develop a software solution they all jointly require. The idea is that there are three options for obtaining software needed within a company:

1. Buy it from a vendor
2. Build it yourself.
3. Use an open source solution. (Raymond, 2000)

In option three, the cost of developing the solution is shared by all of the developers of the project. The example Raymond provides of a successful cost-sharing project is the
Apache software group. In that case, several developers who were responsible for developing internal solutions for the company's web serving needs (option two in the list above) pooled their efforts and ended up with a product that has been extremely successful. From that modest beginning the Apache server software grew, while competing against corporations such as Microsoft, and currently has a market share of 58.7%.29

Risk-Sharing

13. Users wish to share the risk involved in maintaining software over-time (future-proofing) (Raymond, 2000).

Risk-sharing, or future-proofing, is another concept discussed by Raymond. In this analysis, open sourcing a closed-source solution makes sense for a user because the engineers who build a product for internal will use not be employed indefinitely by that company. Unless significant efforts are undertaken to document and train successive engineering staff, the internally developed software will become less and less useful as there is turnover of the engineering staff. Raymond provides an example from Cisco. In the example the software in question is a print-spooling manager. It is again important to emphasize the impact that the licensing system has on the future-proofing of the software. Releasing the software under one of the open source licenses will ensure it is always free (or that version of it will be), which is critical for attracting other potential users. There is no point in the future when the company could simply remove the open sourced version of the software.

29 The source for this data can be found here: news.netcraft.com/archives/web_server_survey.html
Business Models

14. The industry fits one of the identified open source business models.

Raymond identified six known open source business models and one suggested potential model for an open source business. Perhaps more interestingly he revealed the reasons for undertaking such a business model:

1. "Loss-Leader/Market Positioner": In this model, the vendor open sources a part of the product to help drive sales of a closed-source aspect of the product. Netscape provides a good example of this model. The Netscape browser was threatened by the introduction of Internet Explorer by Microsoft. Netscape feared that over time Microsoft could gain de facto control of the hypertext markup language (HTML) standard by developing closed extensions to the browser. In response, Netscape open sourced the browser and continued to sell the server software. The motivation here was two-fold: drive sales of the server software and prevent loss of market share that would make the server software irrelevant.

2. "Widget Frosting": In this model, a hardware vendor open sources any software that accompanies the hardware that is being sold. If the hardware vendor was not deriving any sale-value from the software, then this approach has almost no downside. The direct motivations here are three-fold: reduce costs of software, drive sales of hardware, and provide a better customer experience.

3. "Give Away the Recipe, Open a Restaurant": In this model, the software is the "recipe" and the "restaurant" provides the services around the "recipe."
motivations behind this model are two-fold: adoption of the chosen software, and sell the expertise that the “restaurant” possesses.

4. “Accessorizing”: In this model, the company sells the accessories that accompany an open source solution, including everything from t-shirts to high quality documentation. The motivation behind this model is to capture value associated with an open source product.

5. “Free the Future, Sell the Present”: In this model, the vendor sells the current version of the software with a license that guarantees that, at some point in the future, it will be opened. The motivation behind this approach is to capture sale value and at the same time provide some of the benefits of open source, such as future-proofing.

6. “Free the Software, Sell the Brand”: In this model, the vendor sells a brand name certification for user implementations of a product. The motivation behind this is to create awareness of the underlying solution, as well as its quality, and thus drive its adoption and further brand sales.

7. “Free the Software, Sell the Content”: In this model the software is used to disseminate valuable content. Subscriptions for the content are what create revenue. At this point his model is only theoretical. The motivation behind this model is to drive adoption of the software by making it easy for users to modify it to fit their needs (perhaps new platforms) while deriving revenue from the content.
If a given industry matches one of these business models, the open source approach may become competitive more rapidly.

**Strategic Factors**

Up until this point I have evaluated advantages that open source software and the open source development model have over existing closed-source software and the closed-source software development model. At this point, I will shift gears briefly and discuss two more strategic advantages that open source may hold over its closed-source counterpart. The first relates to matching the pricing of software to the cost pattern in order to align user needs with the vendor resources. The second is a strategic reason as to why open source may overtake an industry at the expense of the incumbent closed-source vendors.

**Aligning Price with Cost**

15. The software pricing model does not align with the cost pattern (Raymond, 2000). Raymond makes the point that 75% of software product's costs come from maintenance and improvements over time (Raymond 2000). However, it is often the case that software is sold for a large upfront sum and a much smaller amount for services over time. This mismatch between the cost and price associated with the service will create motivation for the software vendor that is counter to the needs of the software user. The software vendor will want to sell more software and spend less money delivering service, while the software user will be more concerned about receiving service. In the open source model, the software is free, and most companies charge for service delivered over
the life of the product. This more accurately, though not perfectly, aligns the cost of the software with the pricing model and, because of this alignment, should over time deliver a better quality product to the user.

**Disruptive Process**

16. Software vendors focus on meeting customer needs at the expense of process innovation (Christensen, 1997).

In his book, *The Innovator's Dilemma*, Clayton Christensen defined a specific category of innovation: disruptive innovation. In his study of the hard disk industry, Christensen observed that there were certain types of innovations, usually made by an upstart innovator, that would overtake the incumbent hard disk manufacturers and often put them out of business. The innovations held a set of characteristics: they began with much smaller markets than the one they would eventually overtake, they were usually cheaper and held a technological advantage over the incumbent technology, and they usually began with much lower performance capability than the incumbent technology. These technological innovations would begin by succeeding in a small market where their technological advantage and cost were more important than their lack of performance. Over-time the performance would improve until it could compete with the incumbent technology in the primary market. At that point, because it held price and technological advantage, and could now compete based on performance as well, the incumbent companies could no longer compete. In most cases, the incumbents did not realize that they would have to compete with the upstart innovation for two reasons: they were too focused on listening to their customers and making the improvements that their customers
desired, and two, the new innovation lacked the performance to compete initially, and so it was not clear that they would ever be able to compete; therefore the innovator was ignored.

Is open source a disruptive technology? Yes, though in my perspective it is better described as a disruptive development process. Open source holds two of the characteristics of a disruptive technology: 1) most open source projects begin with very limited features which cannot compete with the established software vendors, and 2) open source holds a price advantage (at least in upfront cost) over closed-source. The initial cost difference will attract a small subset of the market and, presumably, over time, the open source product will improve and eventually will be able to compete with the closed-source vendors. At that point, open source holds a process advantage, not a technological advantage. This will make it very difficult for the incumbent closed-source vendors to compete.

The key to this argument concerns whether or not incumbent vendors pay attention to the progress made by open source competition. If they listen only to their customer's, they will likely be told repeatedly that the open source software cannot compete. That is, until the day that the open source software can actually compete. This day will arrive due to the process advantages that open source holds, at which point the customer will shift to the open source solution. At this point, the open source model is relatively well known
and, as such, this type of strategic mistake by a closed-source vendor may be unlikely to occur. However, historically it has proven to be quite possible.
What does this mean for the future of closed-source software vendors in the storage industry?

Up until this point, I have examined the evolution of the storage industry and the factors that make an industry susceptible to the open source model for software development. By combining these two, I will now make some predictions about the impact open source will have on the future of the storage industry.

Technical Factors

1. “Reliability/stability/scalability are critical.” (Raymond, 2000)

Does a storage appliance need to be reliable, stable and scalable? The answer is a resounding yes, perhaps more so than any other element of the modern data center. The storage system makes accessible the data that many applications will use. If the storage system is not reliable, stable and scalable, then the applications will not be reliable, stable, and scalable. In many data centers, different servers will handle different applications, yet they will share the storage made available through the storage system. The implication is that an unreliable, unstable, or un-scalable storage system will affect the entire function of the data center, not just one application. This will inevitably lead to unhappy users. With that in mind, I will now examine each of three characteristics in depth.

There are two aspects, the hardware and the software, to providing the first two “ilities”: reliability and stability. As I have discussed, commodity hardware has
continued to make strides in performance and quality. Still, no piece of hardware will function reliably and stably indefinitely. For that reason, most storage systems contain redundant, or backup, hardware elements that allow the system to survive any single hardware failure. The failed piece is then replaced; typically the system continues to run through the failure and replace process with no interruption in service. This is generally how reliability and stability are built into the hardware aspects of storage systems; it is a method that has proven itself successful. Building reliability and stability into the software is not as simple. One might suggest using the same model as with the hardware: include a backup or spare copy of the software and run it when a failure occurs. This cannot be done for several practical reasons. The first reason is that software does not fail for the same reason as hardware. Hardware usually fails because it wears out, has a flaw from the manufacturing process, or an accident occurs, such as someone dropping it. This is similar to the tires on your car; use them and they will wear out, or if a mistake was made during manufacture perhaps they will burst unexpectedly, or lastly, you might drive over a nail. Software does not wear out, nor can one copy differ from the other as a result of the manufacturing process - each version is identical. Software fails because it contains a flaw in logic, which is commonly referred to as a “bug”. There are many reasons this bug may be encountered, such as strange timing that the software engineer did not predict, or an unplanned for sequence of events. In the software engineering process, the beta-testers attempt to uncover all of these flaws so that they can be fixed before the software is used by the general public. However, as I am sure each reader has encountered while using a computer, some bugs will slip through.
Keeping this model of software failure in mind, let's return to the idea of maintaining a backup copy of the software in the event of failure. If the bug is encountered by the primary software, there is no reason to believe that the backup or spare software (which is an *exact* copy of the primary software) will not encounter the same issue as soon as it is invoked. Perhaps a second, different, version of the software could be built that has separate logic that would act differently than the primary version? This would require two separate development processes and is simply cost prohibitive.

So, we have returned to the original case, a single version of the software that executes on top of redundant pieces of hardware. Clearly, this single version of the software has to be reliable and stable. The arguments presented earlier in this paper make it clear that open source has advantages over closed source models for developing reliability and stability into software. Therefore, with respect to the reliability and stability requirements for storage system software, it seems likely that open source products will be adopted for storage systems.

The final element, scalability, is also a critical element of the storage system. The amount of data that is being generated by modern applications is growing at an astounding rate. It is the job of the storage system to provide the capacity for all of this data. If the system cannot scale, then the applications cannot function and new data cannot be stored. Again, based upon arguments, such as the power of peer review, presented earlier in this paper, open source will present an advantage over closed source.

Thus, given the requirements placed on a storage system, the "ilities" are critical. Based on earlier arguments, open source holds a clear advantage in providing the
"ilities". Therefore, it seems clear that the use of open source software within the storage industry will continue to expand.

2. "Correctness of design and implementation cannot readily be verified by means other than independent peer review." (Raymond, 2000)

As Raymond points out, most software that performs any useful function meets this criterion. Storage systems are certainly no different. In fact, storage systems are ultimately quite complex. The storage system consists of an operating system and application code that provides specific functions. The software often performs functions that span from very low-level operations, such as drivers that interact with the server systems, to very high-level operations, such as interfaces that take input from human operators. This combination of functions, and the interaction between these different sub-systems, can only be verified through examination by equally skilled software engineers. In this regard open source holds a clear advantage over closed source and, as such, will drive the adoption of the open source model into storage systems.

3. "The software is critical to the user's control of his/her business." (Raymond, 2000)

The previous two points involve improving the technical quality of the product, while this point shifts to a specific aspect of the customer use scenario. This point does not affect storage systems as directly as other more obvious software products, such as those that manage a manufacturing line or handle transaction processing. Those software
products are clearly critical to the control of a business. However, this guidance does not apply to storage systems on a limited scale. Storage systems connect to other systems, such as servers, that are usually designed and built by a manufacturer other than the storage maker. This requires that the products be able to interact with each other, or to interoperate. This is commonly referred to as interoperability. The storage system function that performs this operation is called a driver. This driver must be able to interoperate with other third parties in order for the data to be accessible to the business critical applications. To ensure interoperability manufacturers have adopted open standards for communication, such as the small computer system interface (SCSI) which is a commonly used protocol for connecting storage systems to host operating systems. The software drivers then implement this standard. However, the standards are often open to some interpretation, which leads to different implementations and problems with interoperability. On top of that, manufacturers often construct their own extensions to the standard to add-in functionality that they hope will give them a competitive advantage. This creates more opportunity for interoperability issues. Interoperability is such a big concern that business and universities offer services to help analyze and certify each vendor product; one example is the Interoperability Lab at UNH.30 The fact that this aspect of the storage system software controls access to data makes it critical to the operation of the users’ business. When the software is open source, the user has more options if a problem should occur that prevents access to the data. As in the first two cases, this flexibility will help to drive open source adoption in the storage industry. However, this flexibility does not appear to be as compelling for the storage industry as

30 http://www.iol.unh.edu/
the argument made, for overall improved quality of the software produced, by the first
two of Raymond’s points.

4. “The software establishes or enables a common computing and communications
infrastructure.” (Raymond, 2000)

The storage system often functions as a central point from which multiple servers and
their applications access data. There is no doubt that the storage system enables a
common infrastructure, and that the system performs a crucial role in that infrastructure.
This discussion involves some of the same issues raised under point three that concern
interoperability. The interfaces between the storage systems, one example of which is the
SCSI standard, are designed to enable communication between products from multiple
vendors. However, it is not uncommon for a vendor to make extensions to the standard,
or worse, not to implement the entire standard. Vendors do this for a variety of reasons,
most often to gain some form of competitive advantage with respect to what choices the
customer will be left with for interoperating with the product. The customer often has
little means for validating upfront that the vendor truly implements the open standard in
its entirety, ensuring that any other product from a competing vendor that implements the
same standard will interoperate correctly.

However, under the open source model, the user can validate the implementation
of the standard upfront. Alternatively, the user can trust the certification of an unbiased
third party. Also, as the product evolves over time, the user can ensure that any changes
maintain required interoperability. This clearly presents an advantage for open source.
5. "Key methods (or functional equivalents of them) are part of common engineering knowledge." (Raymond, 2000)

This final point by Raymond is perhaps the most compelling reason that open source has a specific advantage over closed source for storage system solutions. When the key methods of a software product are part of common engineering knowledge no competitive advantage can be gained by keeping them a secret. For example, if a company uses a well known algorithm to build a data compression product there is nothing to stop another company from developing a competing data compression product using the same algorithm. However, if the company is using an algorithm that is not well known and is kept a secret, the competitor will be forced to develop their own different algorithm, which may or may not be as powerful. In this situation it makes sense for the company with the secret algorithm to keep their software closed source, as this will protect the competitive advantage they have. When the product, such as a storage system, is built on well known algorithms the competitive advantage maintained by keeping the source closed is small.

Also, as demonstrated by points one and two, there are significant advantages to making the source open. The basic structure of a storage platform, the operating system and storage of data, is well understood and is often referred to as a commodity in the business press. There is therefore little competitive advantage to keeping this aspect of the storage system closed source. Most vendors compete based on additional functionality that they develop above and beyond the basic service, such as replicating data between remote sites or providing special backup operations. It seems that it will
only be a matter of time before the basic platform aspect of the product is made completely open by one of the storage vendors. This will likely be done by the vendor with the least to lose, similar to the Netscape case and the opening of its browser software.


The impact that a large and knowledgeable user base can have on the fixing of bugs in the software is important; however, fixing bugs is only a subset of the potential impact that Von Hippel was describing. The effect that a wide set of potential problem solvers will have on the development of storage systems beyond bug fixing is not immediately obvious. In the kite-surfing example, discussed earlier, the users benefited from design expertise introduced by experts from a wide range of industries. It seems plausible that this same effect could be observed in the storage industry. Experts from other industries, such as the telecommunications industry, might have insights into building reliable software that are not obvious to developers in the storage industry, though I am unaware of any explicit examples. My own experience in developing software leads me to believe that the primary advantage is in finding and fixing bugs and that additional advantage gained by introducing expertise from other industries will be minor. I say this with respect to the adoption of open source within the storage industry. Von Hippel’s perspective on this point encompasses the larger field of open development models in general. Certainly experts from other industries could have insight into the construction of the storage system as whole that could add great value, including aspects of the technology and development process that could also be improved. However, this
type of analysis is beyond the scope of this discussion.

**Economic Factors**

7. The users have heterogeneous needs that they each find valuable, which the manufacturer is unable to economically meet. (Von Hippel, 2005)

This economic factor is perhaps the single most important economic reason that businesses will adopt open source software. Determining if business storage users have heterogeneous needs is a difficult task. In an attempt to answer this question, I have done some limited analysis on the previously discussed FreeNAS project. The FreeNAS project is coordinated using the Sourceforge open source software management platform. This platform allows users to submit requests for new functionality to the developers that build and maintain a project. The following graph displays the number of feature requests that have been submitted and closed each month over the last twelve months of the FreeNAS project.

![Tracker Traffic for Feature Requests For FreeNAS](image)

*Generated 2007-04-21 21:16:29 UTC*
This seems to indicate a reasonably large number of users desired additional functionality - enough to take the time to make a request for more functionality. For comparisons sake I have included a graph below for the TCL project. The TCL project is one of the top 50 active Sourceforge projects of all time, according to the Sourceforge statistics, while FreeNAS, despite its popularity, does not even fall into the top 1000. Further, the TCL project outranked FreeNAS in overall activity for the previous seven days as of the writing of this document. These two measures indicate that the TCL project is a relatively active project and a useful benchmark for comparison. The following graph describes the Feature Requests made of the TCL development team, for the same previous twelve months described in the FreeNAS graph above.

![Tracker Traffic for Feature Requests For Tcl](image)

The TCL project ranges in number of open feature requests from 0-7, while the FreeNAS project has a range from 11-23 in any given month. While it is difficult to compare any two projects because they will likely come from different technical areas or exist for
different time periods, the above comparison does seem to indicate that FreeNAS users strongly desire a variety of features, relative to a well established software product.

The question then becomes whether or not these feature requests are supportable by a manufacturer. This is an even more difficult question to answer. In my limited analysis of the FreeNAS feature requests, I repeatedly saw requests for FreeNAS to support different additional types of hardware. In many cases, the requestor provided links to outside software vendors that provided the code needed. This means the developers only have to integrate some third party driver code, which is usually not a very difficult task. There is no reason to believe that a manufacturer could not support the same process, yet most manufacturers deliver their products using a standard set of hardware and will not support the use of their storage operating system software with third party hardware. At the time of the writing of this document, I am unaware of any storage company that sells standalone software to be used on hardware that is purchased separately by the customer. However, the activity around FreeNAS indicates that there could be a market for such a business model.

8. The information needed to develop the software is “sticky”. (Von Hippel, 1994)

As was the case with the analysis of the third element in this framework, storage systems do not fit the standard model for requiring information that is “sticky”. As described in detail earlier in this paper, information is “sticky” when it is difficult to transfer from where the solution is needed to where the solution is created. Storage system functionality is reasonably well-defined; however there is one aspect of using

31 A link to the full text of the FreeNAS feature requests can be seen in the Appendix.
storage systems that has “sticky” information. Storage systems are used as a part of a
closer computing infrastructure. Modern data centers often contain a variety of different
systems, such as legacy systems or systems brought in through acquisition of other
companies. Many of these systems will come from different vendors. On top of that,
many of these systems will be running different levels of operating systems or patch
levels. Each of these operating systems or patch levels implies that the host system could
act slightly differently when connected to the storage system. The number of
combinations becomes unmanageable quickly. The bottom line is that connecting a
storage system to the other elements operating inside the data center is different in every
data center. Most storage vendors publish interoperability requirements, but customers
are not always able to comply. Even if they desire to comply, it may be difficult to verify
every system. As such, when a storage system is installed in a customer environment,
there are almost always some surprises. These surprises represent the “sticky”
information. Storage vendors often take significant time to produce fixes that overcome
interoperability problems. Open source software would potentially allow the users to
make their own changes to overcome problems that are specific to their environments.

9. The open source software meets the definition of a “toolkit” developed by Franke
and Von Hippel. (Franke and Von Hippel, 2003)

As described above, the requirements of a “toolkit” are as follows: the ability to
execute complete cycles of experimentation (trial and error), user-friendliness, libraries
with useful and reliable components, and information about how to make a production
version of product. Raymond made a similar observation when he pointed out that the
number of contributors to an open source project is inversely related to the number of hoops required to make contributions. This relationship is based on the idea that a contributor may or may not gain something from contributing; therefore, when the barriers to contribution are high, the work-to-potential-reward ratio is too far skewed even if the contributor has actually already completed a code contribution.

Storage products, perhaps more than any other product in the information technology ecosystem, cannot have bugs within their basic functions. A serious bug in the basic function of a storage product results in irretrievably corrupted data. This type of bug only has to occur once for an intelligent storage user to decide to use a different product. This high risk relationship requires that code changes to this aspect of a storage product are heavily reviewed and tested. This will inevitably result in significant hoops that must be jumped through by contributors to this aspect of an open source project. As Raymond pointed out, the more hoops required, the less participation from actual code contributors will occur. This could serve as a potential barrier to the growth of the product, but it is also inherent in storage products and thus does not pose a competitive disadvantage to commercial closed source products.

As the project manager for FreeNAS, Cochard-Labbé has first hand experience with this issue. He indicated that there are only two regular contributors to the project, but they do receive some patches from outside contributors. This number will likely increase over time. He is also aware of users who have chosen to make their own modifications but not to share them with the wider community. One user modified the FreeNAS code to work with a different operating system. This is a serious modification,
but it seems to indicate that FreeNAS can be modified almost like a toolkit. Modifying
the open source code of a storage system will likely never be something that the average
user is capable of doing; however, there is some indication at this point that expert users
are capable of modifying the code.

10. **The software users value the process of innovating themselves.** (Von Hippel,
2005)

   This is another very difficult metric to measure. If we frame this using the
FreeNAS project, the question is: “Have the users developed this tool because they need
it, or because they value the process of developing it?” My early discussions with
Cochard-Labbé indicate that he had a need for such a product. However, all of the
contributors, including Cochard-Labbé, hold other full-time jobs. It would seem that they
value this process enough to spend their leisure time working on it.

11. **Competitors that have little to lose economically by open sourcing.** (Von Hippel,
2005)

   There are several large players in the storage industry and this scenario may apply
to those that are lagging and trying to gain a competitive advantage. One possible player
is Sun Microsystems. Sun is not primarily a storage vendor and currently lags behind
other vendors that are more closely focused on storage products. There is likely little for
Sun to lose by open sourcing aspects of its storage products. Indeed, there are several
examples of Sun open sourcing parts of its storage system offering.32

32 http://www.cbronline.com/article_news.asp?guid=534E0F91-0941-41B4-8697-D52D98C7FCB6
12. **Users wish to share the cost of software development. (Raymond, 2000)**

In this section, I will analyze Raymond's cost-sharing within the NAS storage industry. When a user is pursuing a storage solution, there are three options available:

1. Buy hardware and software from a vendor.
2. Buy hardware parts from a vendor and build your own software.
3. Buy hardware parts and use open source software.

(A fourth option in which the user builds his or her own hardware is not feasible in most user contexts (perhaps a large company like Google is different), and will not be discussed here). Option 1, in which the user purchases the entire product, is the most common practice. Option 2 is somewhat blurred in this analysis. The reason for this is that it is often the case that end-users build additional functionality on top of their storage solutions, usually in the form of customized code that performs repeated tasks specific to their environments. This raises an intriguing point, because buying hardware and installing open source will still also require the development of custom scripts that perform tasks exclusive to the user environment. However, in the open source case, I would argue it is more likely that the script will be released to the public. The release of such a script and its potential use by others will be discussed in the following section on risk-spreading. For the purpose of this analysis, option two represents building *all* of the required software for a storage solution, including the custom scripts.

Option 2 can easily be eliminated for the vast majority of NAS storage users (again, with the possible exception of Google). NAS storage, by its definition, is attached
to the network that the user deploys. The modern data center network uses openly defined standards for communication. Developing the software required to connect the storage to the network and perform the standardized tasks (such as RAID 5) is, based on my experience, costly and risky. The cost comes from the months, perhaps years, necessary to develop the software. The risk comes from the required testing of such a solution. For these two reasons, option 2 is never considered by the vast majority of NAS storage users. Option 1 is currently the dominant choice from vendors such as Network Appliance (NetApp). In Raymond's original analysis of cost-sharing, users will choose to use open source when the costs of developing a solution that matches their needs compares favorably with the cost of purchasing a solution from a vendor, which may or may not be customized to meet their needs. It is important to remember that the primary cost of a piece of software will be the maintenance and extensions developed over time. How the costs can be shared hinges on the needs of the user. If the needs of a group of users are homogeneous, then a manufacturer can absorb the costs of developing the solution by delivering an end-product that meets the needs of the users. However, if the users' needs vary significantly enough, then the manufacturer will not be able to absorb the costs of developing a solution that matches their needs. In this case it makes more sense for the users to develop an open solution and to spread the costs amongst themselves.

What are the needs of the NAS storage user? First and foremost the product must connect to the network through standard interfaces, and this clearly can be handled by
manufacturers. Second, the product must perform the basic RAID functions required of storage; this, too, can be handled by manufacturers. If a storage product consists of only these functions, and many do, then from a cost-sharing perspective manufacturers seem to match up well with the open source cost-sharing model, purely based on the size of the market and the homogeneous nature of the requirements. Unfortunately, understanding the additional needs of the storage users is not an easy task. As such, I am not certain what is truly important beyond these basic functions. There are a myriad of additional storage functions, such as backups and replication, that storage vendors have moved to provide such functionality. However, it is not clear if these additional functions will prove to be heterogeneous enough that an open source cost-sharing method will trump a manufacturing model.

13. Users wish to share the risk involved in maintaining software over-time (future proofing). (Raymond, 2000)

There is a direct correlation between the Cisco print spooling type of enhancement and the types of “scripts” that are developed around storage platforms to perform basic and repetitive tasks. When the developer(s) who create the additional functionality move on the software often quickly becomes outdated and un-maintainable. When the storage is eventually upgraded the old scripts will often break, and without in house expertise entirely new software will have to be developed. By open sourcing these scripts, the user (company) can hedge the risk that this will happen by enticing other users to adopt the script and help maintain it over-time. This can be done by open-
14. The industry fits one of the identified open source business models.

1. "Loss-Leader/Market Positioner": There are examples of storage vendors that are attempting this strategy already. One prominent example is that of IBM, which has recently open sourced its storage management software.\(^{33}\)

2. "Widget Frosting": Many storage vendors consider themselves to be hardware companies. The prospect of using open source software to drive the sales of the hardware seems very attractive. As stated above, the direct motivations here are three-fold: reduce costs of software, drive sales of hardware, and provide a better customer experience.

3. "Give Away the Recipe, Open a Restaurant": This model could work in the storage industry, though it does not seem as attractive as widget frosting.

4. "Accessorizing": There is not enough revenue in this model to support any of the major storage vendors.

5. "Free the Future, Sell the Present": This model could be adopted, especially if some large customers become fearful of the amount of control that a given storage vendor has over their data.

6. "Free the Software, Sell the Brand": This model also could be applied, though it does not seem to offer the level of revenue required to support one of the large storage vendors.

7. "Free the Software, Sell the Content": This model is not applicable to the storage

---

\(^{33}\) [Link](http://news.zdnet.com/2100-3513_22-5912912.html)
Strategic Factors

15. The software pricing model does not align with the cost pattern. (Raymond, 2000)

Sales in the storage industry still revolve around large up front costs followed by service fees. This is somewhat justified by the purchase of the hardware that accompanies the storage system. However, it does seem likely that an adjustment will need to occur to the software pricing that accompanies the hardware. This does not seem to be as large a factor as it would be in other pure software sales examples.

16. Software vendors focus on meeting customer needs at the expense of process innovation (Christensen, 1997).

The major storage vendors have publicly acknowledged the importance of open source. Examples provided earlier indicate that some have even been willing to make early steps towards open sourcing some of their products. However, these initial steps are minor when compared with the primary products and competencies that the storage vendors have developed. If open source continues to infiltrate the storage industry the large storage vendors will have to undergo difficult transformations. Christensen’s work in this area indicates that many of the established players will not survive this transition. This is not because the vendors cannot make their products open source. The primary issue will be the forced transition in sources of revenue and the adjustment in skills that
these companies will have to undergo to support a business model based on an open
source product.

Conclusion

The storage industry has come a long way in a short time. Over the past 20 years,
the units used to measure system capacity have grown from Megabytes to Gigabytes to
Terabytes to Petabytes. Humankind’s capacity for producing new information is growing
as quickly as these systems can support it. The brief history of the storage industry
provided above delivers some insight into how these systems have evolved to handle new
requirements.

In even less time, the open source movement has established itself as a legitimate
model for software development, capable of competing with the largest software
corporations in the world. The work done by Raymond and Von Hippel in identifying
the factors that have allowed open source to be so successful has been summarized above
and provides a framework for understanding this success.

Combining this historical context with more current projects, such as FreeNAS,
and then evaluating it all using the ideas developed by Von Hippel and Raymond formed
the basis for the study presented. My own conclusion after having done the above
analysis is that open source will continue to grow in use and influence within the storage
industry. I hope that most readers draw the same conclusion. However, this analysis
only covers why open source will grow within the storage industry and in what form the growth will likely take place. There still remains the “who, when, where, and how” open source will continue to expand. Predicting when and where any event will occur has always been a difficult thing to do, and I will not do so here. However, the “who and how” have already begun. The work being done by users such Olivier Cochard-Labbé on the FreeNAS project is one example of who and how open source is making inroads. The initial steps taken by companies such as Sun and IBM are further examples.

Were I to continue study in this area, my focus would revolve around determining which factors identified are most critical for the success of open source in a specific industry. Even having done the above analysis, it is difficult to say which factor is most important for driving adoption within the storage industry, though one can find my own educated estimates above. Identifying a method for determining which factors are most critical in a given industry would provide significant insight into answering whether or not open source is relevant and, if so, the “who, what, when, where, why and how” for that industry.

To conclude, I feel privileged to have had the opportunity to do the above analysis at this moment in time. The Linux operating system and its creator, Linus Torvalds, proved what open source is capable of doing. The parallels between Linux and the FreeNAS project and its creator, Olivier Cochard-Labbé, are striking. I will not be surprised to see FreeNAS and Olivier achieve the same astounding success as Linus and
Linux. The open source model and the community that has embraced it will continue to have enormous success in the software industry and especially within the storage industry. Simply put, open source makes sense.
References


Appendix A

License FreeNAS
FreeNAS is © 2005-2007 par Olivier Cochard.
FreeNAS documentation is © 2005-2007 FreeNAS project.
Logo is © 2005-2007 Youri Trioreau.
FreeNAS is under BSD license.
All rights reserved.

Portions from m0n0wall, both documentation and source code, © 2002-2005 Manuel Kasper, and m0n0wall Documentation Project. Please refer to source code header for exact copyright information. Portion of logo (hard drive) from tdkdesign.net.

Logo Usage
The FreeNAS logo is not released under the BSD license. Usage in any commercial context is strictly prohibited without prior written consent of Youri Trioreau.

Code and Documentation
FreeNAS code and documentation are released under the BSD license, under terms as follows.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* Neither the name of FreeNAS nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Appendix B

The data used in this analysis was collected using a custom software program. Readers interested in seeing the collected data should contact the author. The data can also be found online at www.sourceforge.net.