

Managing a Successful Open Source Software Project: FreeDOS as an Example

Jim Hall

Abstract— This article explains how to manage a successful Open Source Software project. The “Cathedral and the Bazaar” model is explored, and different types of Open Source and Free Software projects are classified. The components that make up a successful Open Source project are itemized, including choice of license and the level of involvement required by project coordinators. The FreeDOS Project is used throughout to demonstrate the success that a marginal project can achieve by following these principles of Open Source Software project management and development.

Keywords— Open Source, Software, Management

I. INTRODUCTION

AN organization’s Open Source strategy is becoming increasingly important in today’s economy. Many users are beginning to expect and demand that much of the software they use be available to them in source form. The availability of source code to a program transforms that project into Open Source Software.

II. THE CATHEDRAL AND THE BAZAAR

The open source software community is fond of using the comparison of cathedrals versus bazaars to demonstrate the differences between how open source and proprietary software is developed. This model was coined by open source software evangelist Eric S. Raymond in his 1996 essay, “The Cathedral and the Bazaar.” Raymond’s comparison is simple:

Linux overturned much of what I thought I knew. I had been preaching the Unix gospel of small tools, rapid prototyping and evolutionary programming for years. But I also believed there was a certain critical complexity above which a more centralized, a priori approach was required. I believed that the most important software (oper-

ating systems and really large tools like the Emacs programming editor) needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time.

Linus Torvalds’s style of development—release early and often, delegate everything you can, be open to the point of promiscuity—came as a surprise. No quiet, reverent cathedral-building here—rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who’d take submissions from anyone) out of which a coherent and stable system could seemingly emerge only by a succession of miracles. [4]

Raymond’s is a very concise description of the open source software development model. I especially like this comparison because it is:

1. Simple enough to propagate
2. Easy enough for non-technical people to accept
3. Catchy

III. WHAT IS OPEN SOURCE SOFTWARE?

Before we go further, it would be a good idea to step back and define exactly what we mean by the phrase “open source software.” This term is used a lot in the news, by developers, by managers. Yet each person seems to have a slightly different idea of what “open source” really means. In some extreme cases, the term “open source” is used interchangeably with “free software.” We will see that the two are not always the same.

A. Open Source Software

Put simply, open source software is any software where users can see the source code. There are remarkably few rules that govern the conditions by which users may view the source code. However, the Internet community at large has come to recognize any software that provides source code as being “open source.”

Jim Hall is the founder and coordinator of the FreeDOS Project (<http://www.freedos.org>) and can be reached at jhall@freedos.org.

Unfortunately, this definition is pretty vague. The Open Source Initiative (OSI) [1] has attempted to document the conditions that define an open source license. These may be summarized below:

1. Free redistribution: Others are permitted to share (or sell) the software, but the license does not require that the author be paid.
 2. Source code: The program must provide source code, and allow others to distribute the source code.
 3. Derived works: The license must allow others to create works based on the open source software, under the same open license terms.
 4. Integrity of the author's source code.
 5. No discrimination against persons or groups.
 6. No discrimination against fields of endeavor.
 7. Distribution of license.
 8. License must not be specific to a product.
 9. License must not contaminate other software.
- [2]

However, the gist of “open source” software is that anyone must be able to view the source code, and both the program and its source code can be shared with others.

B. Free Software

Some free software developers have started to use the term “open source software” interchangeably with “free software”. While free software by any other name would give you the same freedom, it makes a big difference which name we use: different words convey different meanings.

The Free Software Foundation [3] is the body most recognized for its work with free software, and it takes serious issue with the confusion between the two terms “open source” and “free software.” The following is the definition used by the Free Software Foundation:

The obvious meaning for “open source software” is “You can look at the source code.” This is a much weaker criterion than “free software”; it includes free software, but also includes semi-free programs such as Xv, and even some proprietary programs, including Qt under its original license (before the QPL).

That obvious meaning for “open source” is not the meaning that its advocates intend. (Their “official” definition is much closer to “free soft-

ware.”) The result is that people often misunderstand them. Of course, this can be addressed by publishing a precise definition for the term. The people using “open source software” have done this, just as we have done for “free software.” But this approach is only partially effective in either case. For free software, we have to teach people that we intend one meaning rather than another which fits the words equally well. For open source, we would have to teach them to use a meaning which does not really fit at all. [5]

Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. Like open source software, this requires that the source code be made available to the end user, otherwise the user would not be able to study, change and improve the software. In this usage, the “free” refers to freedom, not price. This is also the origin of the much-used motto “free as in speech, not as in beer.”

More precisely, it refers to four fundamental kinds of freedom for the users of the software:

1. The freedom to run the program, for any purpose.
2. The freedom to study how the program works, and adapt it to your needs. Access to the source code is a precondition for this.
3. The freedom to redistribute copies so you can help your neighbor.
4. The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this. [6]

The Free Software Foundation guarantees these freedoms through its GNU General Public License (GPL). [7] The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU GPL is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. The GNU GPL is designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things. The General Public License does not permit incorporating your program into proprietary programs.

So, free software is similar to open source software, but they are not quite identical. The definition of open source software does not quite meet the standards of free software. However, the definition of free software does qualify itself as open source software.

IV. WHAT MAKES A GOOD OPEN SOURCE PROJECT?

An organization's Open Source strategy is becoming increasingly important in today's economy. Many users are beginning to expect and demand that much of the software they use be available to them in source form. Therefore, it is important to understand the qualities of effective open source projects.

There are ten basic qualities that an open source project must possess in order for it to succeed.

A. *Every good project starts by scratching a personal itch*

My first exposure to managing an open source software project started with the FreeDOS Project, and began in 1994. I was a physics student at the University of Wisconsin-River Falls, and I used DOS quite a lot to do data analysis, write papers and physics reports, dial into the University network, and write small programs to help make my life easier. DOS meant a lot to me, and I was most comfortable with using DOS to get my work done.

So it was a bit of a surprise when Microsoft announced that they would stop support of MS-DOS, and that everyone would soon be using their Windows product. Of course, note the timing—this is a little over a year before the launch of Windows 95. I didn't like Windows (then, at version 3.1) because I felt it made my work too difficult. I could accomplish the same tasks in DOS, mostly using the command line, and I could do it faster than in Windows. In Windows, everything was done with a mouse. It just slowed me down, and I felt it was a sloppy GUI.

I wasn't alone. A lot of other people on various DOS newsgroups were shocked to hear that MS-DOS would soon go away. They didn't like Windows any more than I did, and they were just as resistant in being "forced" to migrate to Windows and away from our lovely DOS operating system.

And many people did not have machines capable of running even Windows 3.1, let alone a "next generation" version of Windows. I had a '386 with 4MB memory (later upgraded to 8MB.) A lot of people still had '286 PC's. You just can't run Windows in that. If Microsoft were going to push us toward Windows, we'd need to upgrade our PC's. And that didn't seem right. We felt as though our freedom were being taken away as well when Microsoft decided to take away MS-DOS.

So, on the newsgroups, people started trying to find ways to preserve their freedom. By 1994, Linux had become an underground success story in a lot of Universities. People looked to Linux and asked, "if they can create a free version of UNIX, can we create a free version of DOS?" I had already installed Linux in a dual-boot configuration (with MS-DOS) on my '386, so I knew what a great operating system Linux was. Writing a version of DOS seemed almost trivial next to a multi-tasking, multi-user UNIX kernel.

After a number of weeks, no one on the DOS newsgroups seemed to be interested in *starting* a free DOS project, but a lot of people definitely *wanted* one. I was afraid that if someone didn't at least *try* to create a free DOS, the initiative might be lost, so I posted a note to the newsgroups announcing my intention to create such a project.

People liked the idea! I immediately started to get email from people who wanted to contribute. A few more programs were contributed, and we started scanning the ftp sites for free programs that reproduced MS-DOS functionality and that included source code.

The origins of the FreeDOS Project could just as well apply to any open source software project. In order for the project to exist, there must first be a need. A developer fills that need for himself by writing software, and then turns it into open source software when he gives away the source code for others to share and improve.

B. *Users should be co-developers*

The definition for open source software is that the source code must be open to inspection. A necessary side-effect of this statement is that the users of the software will then be able to view the source code, and make improvements to it. In a well-managed open source software project

will then accept those improvements in the form of patches, and then a new release of the project is made that benefits all its users.

The power of this effect is easy to underestimate. In fact, pretty well all of us in the open source world drastically underestimated how well it would scale up with number of users and against system complexity.

Take, for example, the FreeDOS Project's kernel effort (DOS-C):

DOS-C started in 1988 as an experiment started by Pasquale J. Villani in writing device drivers in C for Microsoft's MS-DOS. Both block and character device drivers were written, along with special C data structures to match the MS-DOS request packet. It was then recognized that using the same techniques, an operating system could be written that would take advantage of the C language features and would require much less time to develop than the traditional assembly language techniques. Although UNIX had proven this earlier, it was not tried with a traditional PC operating system.

At this time, a minimal operating system using the device drivers written earlier along with a new 8086 interrupt API was developed. It was called XDOS and proved to be a functional operating system. This new operating system was used to develop booting techniques and a C library SDK was developed for it.

XDOS enhancements were started in 1989 and MS-DOS was chosen as the new API. A more advanced architecture was also developed. This included the use of an IPL (intermediate program loader) to set up the operating environment prior to loading the operating system itself and re-entrant system calls facilitating real-time applications. This version, known as NSS-DOS, was completed and demonstrated in 1991.

New proprietary techniques were added that allowed the same source to be compiled on a variety of hosts and with a wide range of compilers. This new version, DOS/NT, was the result of this new project. The kernel was redesigned as a micro kernel along with logical separation of the file system, memory and task managers. A new DOS API was designed along with a new DOS SDK to guarantee portability. Additionally, all processor unique code was separated from the core functions. The

result is the highly portable operating system that DOS/NT represents. [9]

After a number of successful commercial applications, DOS/NT became part of both DOSEmu and the FreeDOS Project in 1995. DOS/NT was released under the GNU General Public License, now making it open source software, and re-named DOS-C.

However, it is important to note that, although DOS-C had been quite mature in its development from 1988 to 1995, it had only been maintained by one person. As a result, debugging and code improvement were limited to the time and resources available to Villani.

When the source code to DOS-C was made available to the public, the FreeDOS kernel was not able to support LBA, CD-ROM drivers, or network redirection. Most noticeable in the FreeDOS kernel was floppy access speed. While floppy drive support was functional, Villani originally wrote the code as part of a real-time OS that worked with DMA and was multitasking. Floppy access was always set for read-ahead, so that while DOS-C was reading or writing the current sector, the next one was filling the buffer. On the PC, DOS-C must fetch each sector and wait until it finishes. Each sector read waits for another revolution of the disk, and since this is tens of milliseconds, it is very slow.

However, when the source code to DOS-C was made available to the FreeDOS community, other developers were able to inspect the code. Improvements were suggested very rapidly. A developer named `ror4` provided a floppy driver that enabled buffering, considerably speeding up the floppy drive access for FreeDOS. Another developer named James Tabor provided CD-ROM support and added network redirection, which was later improved by FreeDOS and DOSEmu contributors Bart Oldeman and Tom Ehlert. Yet another developer named Brian Reifsnnyder provided LBA support. The list of patches goes on, with more names than can be listed here.

As a result of opening the source code to its users, the FreeDOS community was able to provide rapid code improvement and debugging. In contrast to the cathedral-building style of Villani's original development of XDOS and its successors, the evolution of the FreeDOS kernel was fluid and

very user-driven. Ideas and prototypes were often rewritten three or four times before reaching a stable final form in the kernel.

This concept of sharing source code for its cooperative development and improvement is often referred to as “mind share,” and is key for the development of any open source software project. Given a large enough co-developer base, even a complex system such as an operating system kernel can become simple to develop and to maintain.

C. Release early, release often

When many developers are involved in a single software project, many patches can be provided in a fairly small time window. It is important to maintain constant feedback to the users who are the co-developers of an open source software project. As projects continue to be supported by more than one person, we have begun to recognize the importance of frequent releases.

As new patches are provided to a project, a successful open source project must then release a new version of the software that includes those patches. This can sometimes be a daunting task. The importance is in keeping the co-developers and users constantly stimulated and rewarded by the sight of constant (even daily) improvement in the system they are using.

Yes, this can often result in an unstable release. But as your co-developers begin to see the immediate effect of patch becoming process, releases will result in a steady evolution of the source code into stable versions of the software. As was mentioned before, the bazaar model sometimes will resemble a babbling chaos of differing agendas and approaches out of which a coherent and stable system emerges only by a succession of miracles. Releases will be made, and the software will improve.

But the frequency with which you release your software will often depend on its size. A small library such as FreeDOS Cats (an implementation of the UNIX `catgets` function, to add internationalization support to FreeDOS programs) might be released quite often; sometimes, I have released more than one version of Cats in one day. A basic utility program such as `type` or `more` might be modified heavily only in spurts, for example when Cats support was added, but would otherwise remain static.

A collection of software, however, such as the FreeDOS beta distributions (the latest version is Beta6) might be released in increments, when enough of the packages have matured and stabilized. The FreeDOS distributions are released about once every six months. This is similar to commercial software, which is released according to some regular schedule. However, note that even a large collection such as the FreeDOS distribution still follows the “release early, release often” motto. While it may seem long to a FreeDOS developer, six months to wait between releases is still much less than the release cycle that large cathedral-building software companies typically use: Microsoft’s Windows operating system is released about once every other year.

D. Project coordinator/maintainer

Looking at the chaos of the bazaar, with a constant stream of patches and releases, we begin to wonder what holds it all together. How does this not devolve into self-destruction? The answer lies in the project maintainer.

An open source software project’s coordinator should have good people and communications skills. This person will be responsible for many things, including accepting and merging software patches, helping to contribute to the project’s documentation, listening to what the users and co-developers are looking for (but perhaps are not yet able to provide) and find ways to accommodate them.

But perhaps the skill that the project coordinator will find most useful is the ability to listen. The coordinator must recognize that no one person will have all of the correct answers all of the time. Knowledge and insight will come from different directions. It is the coordinator’s ultimate responsibility to understand that many developers working together on a project are better than one talented hacker.

When I founded the FreeDOS Project, I came into it with the naive view that most of my time could be spent writing code for FreeDOS, and only a little of my time dedicated to the various tasks of keeping the various efforts moving forward. My first contribution to the FreeDOS Project was writing basic file utilities. Later, I wrote the FreeDOS Install program, and became

the release coordinator for each of the FreeDOS beta releases.

In the early days, this was great. After all, I was still a student when FreeDOS was born, so how much time did I really have to dedicate to managing everyone's efforts? However, as the FreeDOS Project quickly grew, I began to realize that the opposite would be true: most of my time would focus on various coordination activities for the FreeDOS Project, such as responding to queries, writing documentation, and web site management, leaving little time for writing actual code.

Today, I spend approximately 90% of my time coordinating various efforts in the FreeDOS Project. In this seeming bazaar of conflicting agendas, differences of opinion are bound to appear. I try to take a hands-off approach in these clashes of egos, but I do act as a mediator to help both sides reach an agreement. Also, without a webmaster, I manage both the FreeDOS web site and the FreeDOS files archive. This leaves me precious little coding time available for the FreeDOS Project. I haven't contributed code for the FreeDOS kernel or FreeCOM (the FreeDOS command shell) in years. However, the dream of a free version of DOS, with source code available to all, is still alive, and that is what keeps me active in the FreeDOS Project. Each coordinator for an open source software project must similarly find his own motivation.

But other time commitments may eventually get in the way, and when that happens, motivation tends to suffer. When that happens, the only option left may be to hand the project over to someone else. This has happened to me once, in 1995 when I became gainfully employed in the private sector and getting acquainted with my new job ate up most of my free time. I passed the role of FreeDOS Project Coordinator on to Morgan "Hannibal" Toal. In 1997, Toal's involvement with the FreeDOS Project began to dwindle, while mine began to be re-invested, and in November of that year we passed the torch back to me.

Other open source software projects see this all the time. After several years of working with the FreeDOS Project, Villani found he was no longer able to dedicate sufficient time to the development of the FreeDOS Kernel. Jim Tabor had recently

contributed CD-ROM support to the kernel, and both he and Villani agreed that Tabor would become the new kernel maintainer. In 2001, Tabor similarly found that his free time was no longer enough to respond to kernel patches that users were submitting. Again, Bart Oldeman had been working quite closely with the kernel for some time, and both Tabor and Oldeman agreed that it would be best for the FreeDOS Kernel if Oldeman became the new kernel maintainer.

These examples highlight an important point of open source software development: the success of the project does not (and should not) depend on one person. When you lose interest in an open source project, your last act as the project's maintainer to it is to hand off the role to a competent successor.

E. Organization of the project

As discussed earlier, it is important to recognize that the users of an open source software project are also its co-developers. The success of the project depends solely on enough developers contributing toward that project. So it is extremely important to make the project interesting to developers.

When the FreeDOS Project was first founded in 1994, the Web had not yet become reality. It was not until around 1995 that the Mosaic web browser became readily available, and several years would pass before a majority of "Netizens" had access to the Web from their homes. So in our beginning, it was easy enough for the FreeDOS Project to simply provide an ftp site with all our software and source code. We conversed with one another via USENET newsgroups.

In today's world, a simple download site and discussion forum is not enough to garner new interest, and hence attract new blood to the development effort of the FreeDOS Project. Any modern open source software project that wants to thrive must provide a web site. And more importantly, that web site must be well organized.

The project web site must provide a continuous stream of news about where the project is going, new developments, major bugs found and fixed, and recognition of the project by outside groups. Developers are people too, and we like to see our name in print, even if it is on a transient medium

that is the Web. Spotlight those who make major contributions, but strike a careful balance. One sure way to kill interest in newcomers is to establish an environment of “us vs. them.” Remember: an open source software project is for everyone! Remain inclusive, not a members-only club.

The web site must make it easy to find everything about the project. Not everyone who decides to contribute to a project will be a developer capable of contributing code. Rather, some users will only be able to provide debugging information on strange new hardware. Others will have expertise in user interface design, while others will be highly skilled technical writers.

When the FreeDOS Project was formed, the need to create a free DOS was obvious: Microsoft was going to drop support for MS-DOS, but we still used and loved the DOS platform. If Microsoft wasn't going to continue with DOS, we needed to create and support our own DOS platform. My assumption at the time was that the only people who would use and contribute to the FreeDOS Project would be others who had used DOS for years and understood its technical underpinnings. Novice users, I assumed, would follow Microsoft down the path of Windows. I didn't know any better.

A lot of “general” users are using FreeDOS these days. Often, I hear about someone who has installed FreeDOS on an old PC they brought home from work, or on a PC that a friend gave to them. These systems are often too slow or don't have enough memory to run Windows or even Linux, but they will run FreeDOS just fine. A good number of these systems are used to access the Internet (using `dosppp` to dial the Internet provider) or as a system to write letters to friends and family with a DOS-based word processor. With the introduction of Seal as a graphical user interface to FreeDOS, more users who may never have used the DOS command line are now beginning to approach FreeDOS as a serious alternative operating system.

A good web site that provides the information these users are looking for will benefit the project as a whole. For example, on the FreeDOS web site we provide ways that graphic artists might contribute web banners for the FreeDOS site, or for software testers to help us identify new prob-

lems, or for technical writers to assist us in creating a body of documentation that helps others understand FreeDOS. This underscores the fact that the users of an open source software project such as FreeDOS are not all of one ilk. As more and more users come to the project, the variety of those contributing to it will increase.

F. Documentation

Documentation is often the first introduction a user will have to an open source software system. It provides both a way for new users to become acquainted with the software (in the form of user manuals) and for more experienced users to learn how to perform complicated tasks more easily with the software (in the form of recipe guides, also referred to as a “Howto.”) So it is not surprising that a large effort is typically put into making the documentation for an open source software project a first-rate experience.

This is such an important step for many projects, and often the documentation effort itself is split off into a separate, related project. Linux enthusiast Matt Welsh co-founded an effort to provide documentation for fellow Linux users, by writing recipe guides. In doing so, Matt demonstrated the success that a documentation effort can achieve. Today, the Linux Documentation Project's HOWTO guides are well-respected, frequently reprinted as books, and often referred to as one of the reasons for the success of Linux.

Recognizing the success the Linux Documentation Project (LDP) had in raising understanding about Linux, the FreeDOS Project has formed a similar sub-project. In the founding of the FreeDOS Documentation Project (FD-DOC), we followed the spirit of Matt's intent with the LDP. It would be well for any documentation effort to follow similar goals:

1. The documentation project is primarily a vehicle for enthusiasts and developers to share their knowledge about the system with other users and co-developers. People are motivated to contribute to the documentation effort because they know that by having their articles on the web site, many users are likely to read what they have written.
2. The documentation project should be the *de facto* standard place for people to go to find out about the open source software project that it sup-

ports.

3. As such, it is *vital* that it is as easy as possible for people to contribute to the documentation effort. Participation in complex standards processes, voting organizations, or high-traffic mailing lists should never be a requirement.

4. Likewise, the tools used to write documentation should be easy to use, widely-available, free, and well-supported. In the FreeDOS Documentation Project, we have adopted the `roff` program with the `-ms` macros to typeset our documents. We find that this is a widely-accepted documentation standard that is easy for new users to learn. Other documentation projects have chosen similar tools such as DocBook (an XML-based typesetting system.)

5. It is important to make it clear that the documentation project is open to contribution by anyone, and is not a closed, privately-run organization motivated by corporate profit concerns. Otherwise, the documentation project loses its identity as an open organization which exists to serve the user community as a whole. [10]

G. Bug tracking

Often a major sticking point for newcomers to open source software is the issue of tracking bugs. To empower your users as co-developers, you must provide them with the information needed to become efficient developers. An open source software project must equip its developers with a bug tracking mechanism. After all, a software project cannot improve itself if it cannot identify what is wrong in the software.

There are a number of bug tracking systems available for an open source software project. [11] Bugzilla, Jitterbug, and GNATS seem to be the most popular. These bug tracking tools all share a common set of features:

1. The bug tracking system must be easy to use. This applies both for entering new bugs and for tracking existing bugs. A system that is difficult to enter new bugs will not be used, and problems in the software will remain unaddressed. A system that makes it difficult to reassign bugs and to mark them as resolved will be similarly abandoned.
2. The bug tracking system must have a strong query capability. If users cannot find existing bugs, duplicates will be reported.

3. The system must be open to all eyes. This is perhaps the most difficult item for traditional software managers to accept. In the cathedral model of software development, bug lists are often hidden from customers. After all, there is no beta of the software released before its time, and software is released only when it is deemed “ready” by an internal quality assurance team, not by its end-users. However, in the bazaar model of software development, everyone who uses the software is potentially a developer for the system. In order to foster new development and patches to the system, developers must be able to see what bugs currently exist.

The FreeDOS Project uses a bug tracking system called simply `bugtrack` that merges the simplicity of Jitterbug with a user-friendly interface similar to Bugzilla. As of June 2001, over 750 active bugs had been entered into `bugtrack`. This might seem a large number, especially in a small project such as FreeDOS, but it highlights the fact that everyone who uses the system is also helping to debug it. More users find more bugs because adding more users adds more different ways of stressing the program. This effect is amplified when the users are co-developers. Each one approaches the task of bug characterization with a slightly different perceptual set and analytical toolkit, a different angle on the problem.

This level of openness is impossible without an effective bug tracking tool that is open to everyone.

H. Make source code available

Put simply, open source software is any software where users can see the source code. There are remarkably few rules that govern the conditions by which users may view the source code. However, the Internet community at large has come to recognize any software that provides source code as being “open source.”

Open source software requires that the source code be open to inspection. The users of the software must be able to view source code and to make improvements to it. The project’s maintainer can accept those improvements in the form of patches, and then a new release of the project made that benefits all its users.

Take, for example, the FreeDOS Project’s kernel

effort (DOS-C): when the source code to DOS-C was made available to the FreeDOS community, other developers were finally able to inspect the code. Improvements were suggested very rapidly, resulting in increased floppy access speed, CD-ROM support, network redirection, and LBA.

As a result of opening the source code to its users, the FreeDOS community was able to provide rapid code improvement and debugging. In contrast to the cathedral-building style of Villani's original development of XDOS and its successors, the evolution of the FreeDOS kernel was fluid and very user-driven. Ideas and prototypes were often rewritten three or four times before reaching a stable final form in the kernel.

Making the source code available to its users allows for the cooperative development and rapid code improvement that fosters "mind share."

I. Respond to submissions

Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging. Because source code is available, an open source software project can enjoy shortened debugging time. Given a bit of encouragement, your users will diagnose problems, suggest fixes, and help improve the code far more quickly than you could unaided.

However, the only way to *keep* your users working with you as co-developers is to respond to the patches they provide. Keeping the co-developers and users constantly stimulated and rewarded by the sight of constant (even daily) improvement in the system they are using.

In the FreeDOS Project, my most active development efforts are aimed at the installer and a library called Cats (to provide internationalization support to DOS programs.) A very competent hacker named Jeremy Davis found an interest in these programs, and began to regularly submit patches. Eager to have help in my coding efforts, I responded by immediately reviewing his patches and checking them into the source tree. Releases of the software became more frequent, and bugs were quickly identified and patched.

None of this would have been possible had I stopped responding to Davis. Even when I suffered a hard disk crash in the middle of our coding efforts, I let him know that I was still reviewing

the work and checking in what I could. The key to our collaboration was constant feedback.

The worst thing that can happen to an open source project is to become unresponsive to its users. After all, the users are also the developers. A sure way for a project to kill interest is for the maintainer to reply to a developer, "Thanks for the patch, but I was planning to add that myself anyway in the next release." The project coordinator must recognize that no one person can be the sole developer on an open source software project. Without users and co-developers, there is nothing to coordinate. It is the coordinator's ultimate responsibility to understand that many developers working together on a project are better than one talented hacker.

J. Recognize your limitations

By no means does the fact that a project is open source software imply success. No project, either open or proprietary, can be guaranteed that it will survive the test of time. Jamie Zawinski, well-known hacker from his years with Netscape and Mozilla, frames this statement for open source software projects:

Open source does work, but it is most definitely not a panacea. If there's a cautionary tale here, it is that you can't take a . . . project, sprinkle it with the magic pixie dust of "open source," and have everything magically work out. Software is hard. The issues aren't that simple. [8]

In other words: open source is not a magic bullet. A project must be interesting to other developers in order for them to contribute to it. Often, this can be a hard sell. The program's maintainer must demonstrate the value of the software project before interest can be garnered. True, the value sometimes reveals itself to be pure "hack value," where the project's only real purpose is as a vehicle for interesting experiments.

An example of "hack value" is the FreeDOS Internet Services Host (FISH). Written circa 1999 by Gregory R. Ball, FISH is a web server program that runs on a PC running FreeDOS. Really, the program was a demonstration that network applications could be written for FreeDOS using WATTCP. FISH was deployed on a 386 SX/25 with 4MB of memory, but the web site advertised that it will run on a far lesser system. No

one seriously believed that web servers would suddenly switch to a DOS-based system, and FISH was never taken as a real alternative to more established web servers such as Apache. However, FISH was released as open source software, and attracted quite a lot of attention for a long while.

The need to generate interest in order to attract developers to a project is summarized by open source software evangelist Eric S. Raymond in his 1996 essay, "The Cathedral and the Bazaar." Raymond's statement is simple:

When you start community-building, what you need to be able to present is a plausible promise. Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is (a) run, and (b) convince potential co-developers that it can be evolved into something really neat in the foreseeable future.

V. CONCLUSION

The concept of open source software has revolutionized the software industry. No longer is software developed behind closed doors, built like cathedrals, carefully crafted by individual wizards in splendid isolation. Rather, the bazaar model of software development, where users of the software are also its developers, has proven itself as a successful way to develop software.

An organization's Open Source strategy is becoming increasingly important in today's economy. Many users are beginning to expect and demand that much of the software they use be available to them in source form. Therefore, it is important to understand the qualities of effective open source projects.

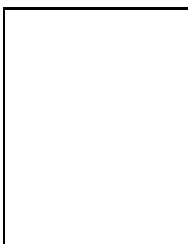
We have discussed ten basic qualities that an open source project must possess in order for it to succeed. By no means does this imply a recipe success. No project, either open or proprietary, can be guaranteed that it will survive the test of time. A project must be interesting to other developers in order for them to contribute to the software. However, with the software available to its users in source form, and a project coordinator who works well with co-developers, any project has the potential to become a successful open source software project.

ACKNOWLEDGMENTS

The author would like to acknowledge the contributions of many people who are a part of the FreeDOS Project.

REFERENCES

- [1] *The Open Source Initiative*, www.opensource.org
- [2] *The Open Source Definition*, Version 1.8, www.opensource.org/docs/definition.html
- [3] *The Free Software Foundation*, www.gnu.org or www.fsf.org
- [4] Raymond, Eric S. *The Cathedral and the Bazaar*, O'Reilly and Associates, 1999. Also www.tuxedo.org/esr/writings/cathedral-bazaar/index.html
- [5] Stallman, Richard. *Why "Free Software" is better than "Open Source"*, www.gnu.org/philosophy/free-software-for-freedom.html
- [6] Stallman, Richard. *The Free Software Definition*, www.gnu.org/philosophy/free-sw.html
- [7] Stallman, Richard. *The GNU General Public License*, Version 2, June 1991, www.gnu.org/copyleft/gpl.html
- [8] Zawinski, Jamie. *resignation and postmortem*, www.jwz.org/gruntle/nomo.html
- [9] Villani, Pasquale J. *The FreeDOS Kernel*, R&D Books, 1996.
- [10] Welsh, Matt. *Thoughts on the Linux Documentation Project*, www.slashdot.org/features/99/08/25/1351232.shtml
- [11] Vepstas, Linas. *Call Center, Bug Tracking and Project Management Tools for Linux: Open Source Bug Tracking/Trouble Ticketing Systems*, April 2001, www.linas.org/linux/pm.html



Jim Hall lives in St. Paul, Minnesota, with his wife (Sara) and three cats (Murphy, Vita, and Linus.) In 1994, he founded the FreeDOS Project, an open source project that aims to produce a free version of DOS that is compatible with MS-DOS. Jim also wrote GNU Robots, a free software program that allows students to learn the fundamentals of computer programming by creating a program in which a virtual robot explores a virtual world. Jim has also played an important role in the development of other open source and free software projects, and has contributed to GNU Emacs, DOS Freemacs, and several DOS/UNIX compatibility libraries. At work, Jim is Web Production Manager for the University of Minnesota.