The risks of writing proprietary software

Concrete economical reasons for avoiding proprietary software development

Matt Barton

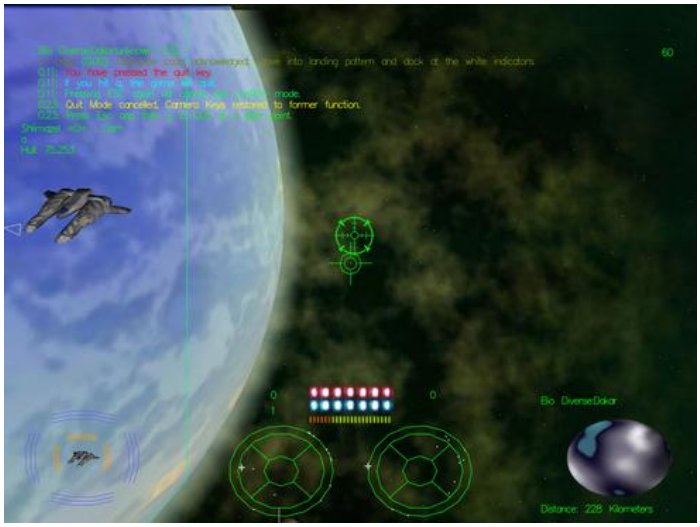
Every software developer faces a choice when deciding how to release a new software product. That choice is whether the program will be free or non-free. Unfortunately, many otherwise knowledgeable programmers aren't sure just what this choice means, and may complain that programmers with families really don't have a choice at all — if they want to earn a living, they must charge for their work. However, free software is not about giving software away without cost. Rather, free software is simply an ethical choice that guarantees the freedom of users — and, perhaps more importantly, the freedom of the developer. This last point is often lost, even in discussions among free software developers about the benefits of publicly licensed software.

Nevertheless, a developer who chooses a free software license is not necessarily acting under purely selfless motives - there are concrete economical reasons for doing so. I will first discuss the immediate benefits of releasing free software to developers, then discuss broader advantages to the industry and society.

The private benefits of public licensing

I recently had the opportunity to interview two game developers: Daniel Horn and Mike Boeh. Horn chose to release his popular *Privateer* remake, *VegaStrike*, under the General Public License, whereas Boeh has achieved the "holy grail" of independent game development - making enough money to adequately support his family - by releasing a series of high-quality proprietary action games. Both developers are highly skilled and successful at marketing their work, and though their programs offer the player substantially different experiences, their games are highly polished and professional - and a good deal more fun to play than many big-budget "A-List" titles.

Fig. 1: VegaStrike is one of the most sophisticated GNU games available



Comparing the development strategies of these programmers is quite revealing. Horn had a stunning revelation when he learned about GNU/Linux and the unique way it had been built. Horn, who had intended *VegaStrike* to be a standard proprietary product, decided then to embrace the free software model. "I realized that this was how it had to

be. This is what would differentiate me from my competition," says Horn. This decision proved to be the right one. Soon after he announced his game, another programmer who had been working on another *Privateer* remake heard of it and promptly decided to abandon his own project and dedicate his time and energy to *VegaStrike*. Graphic artists, musicians, and writers soon followed, and now *VegaStrike's* development is largely the responsibility of contributors volunteering their work. This is possible because users have access to *VegaStrike's* source code, or the "human-readable" scripts that tell computers what to do. If Horn had only released the binary code, or the "machine-readable" scripts, the users would have been unable to reliably discern how the program worked or contribute to it. Thus, by *sharing* his source code, Horn has received help from hundreds of users who gladly shared their resources and talents with him.

Mike Boeh's approach to software development differs sharply from Horn's. Most significantly, Boeh's games are released only in binary form. Furthermore, though Boeh offers "playable demos" of his titles, users must purchase the "full versions" and are not allowed to share them. While Boeh prides himself on the originality and versatility of his games, he admitted to me that he does share an engine, or the "core code" of his games, with another proprietary developer. However, Boeh does not choose to share this engine or his code with other developers. Instead, Boeh contracts out for artistic and music talent and either pays them up front or promises them a share of his royalties.

A developer who chooses a free software license is not necessarily acting under purely selfless motives - there are concrete economical reasons for doing so

According to Boeh, programming is hard, tedious work that very few people would choose to do for free. "The barrier of getting a game done is the size of the task. You have to write such a lot of code. If you have a wife and a child, it's hard to stay focused," says Boeh. While it's relatively easy to get a simple prototype up and running in a few weeks, months of tough and often frustrating work follow. The game must be polished; bugs must be found and eliminated; sloppy routines must be detected and smoothed

Fig. 2: Retro64's Cosmobots — proprietary shareware

out. These tasks are difficult and laborious - the sort of boring and repetitive tasks that most sensible people expect to be paid for performing. Boeh believes the value he adds to his software is his meticulous coding practices and polishing, and a few moments spent browsing the software library at *Retro64* is enough to demonstrate his superior craftsmanship and attention to detail.

Boeh feels he has good reasons for releasing his games under a proprietary, non-free license. Boeh's biggest fear is that unscrupulous competitors would use his code to quickly produce competing "knock-offs," or games that differ only superficially (if at all!) from his. Someone cloned his game *Z-Ball* and tried to fool consumers into thinking it was Boeh's game. A few developers even tried to copy Boeh's website, going so far as to clone his slogan "Where the fun is never old." Since Boeh is striving to earn a living doing what he loves - programming great games - he feels he has a good reason for keeping his code secret and doing his best to protect himself from competitors.

Boeh and Horn have made trade-offs. Horn has traded secrecy and a certain level of security in the hopes that he will be able to take advantage of the public's goodwill. Hundreds of developers have chipped in to help Horn find bugs and improve and extend his code. Boeh, on the other hand, has sacrificed this help for the sake of keeping his code secret and thus hopefully reducing the threat of competition. Which trade-off is more advantageous?

Although it is foolish to make a generalization based purely

on the experiences of two developers, in this case, it's clear that Boeh's method is earning him more revenue (I would *guess* he earns roughly $35,000 or more). Though Boeh didn't give me exact figures, he is proud to admit he earns a respectable living purely by producing and selling his games on the net. Horn, on the other hand, has made slightly over $200 selling CD versions of his game. For a developer faced with the choice of earning $35,000 vs. $200, the choice of whether to write free or proprietary software seems clear.

However, such figures are *highly* misleading. Horn is quick to point out that while $200 seems a paltry sum, the exposure the game has brought him amounts to much more. "It's helped me get a lot of jobs," says Horn. "I have worked a lot of places during the summer. I worked for Sony last summer, NVIDIA a few summers ago - I was working on OpenGL drive development." Producers looking for talent are impressed with Horn's work - not only because they like what they see in *VegaStrike*, but more importantly, they can get a good look at his coding practices. Source code is far better than resumes or recommendation letters for showing a potential employer that you have what it takes to contribute to important projects. After all, if you were a restaurant owner seeking a chef, wouldn't you want to watch that chef in action as well as taste her Chicken Roulade?

Another problem is that while Boeh has demonstrated his ability to produce quality games, he has not demonstrated his ability to work with large teams of other people - a critical skill in today's software development industry. Eric Raymond, author of *The Cathedral and the Bazaar,* puts it this way:

> "The developer who uses only his or her own brain in a closed project is going to fall behind the developer who knows how to create an open, evolutionary context in which feedback exploring the design space, code contributions, bug-spotting, and other improvements from hundreds (perhaps thousands) of people." (51)

Thus, releasing the source code to a piece of software not only allows potential employers to see a programmer's technical ability, but also her potential to manage a large project involving hundreds of other coders, artists, and musicians - a skill that sometimes seems more important today than programming.

Raymond also points out another vital characteristic of modern software development - very few programmers earn a living working for proprietary developers. The great majority of programmers work for "in-house" projects, creating and maintaining software for business and industries. Raymond advises his readers to check the want-ads of their local newspaper for evidence of this fact.

In short, a programmer striving to "learn the ropes" and get a leg up on the competition - the thousands of other aspiring programmers emerging from universities, colleges, and institutes - could do well for herself by developing and releasing a useful and influential free software program or contributing to an existing one. Raymond writes, "Prestige is a good way to attract attention and cooperation from others" which may very well earn the programmer much higher-paying jobs than she could otherwise expect (84).

The public benefits of free software

Though Bill Gates may sometimes contend that free software development is harmful to our way of life - even going so far as to refer to it casually as "communistic" in a recent CNET interview, the public benefits of public licensing are clear, and have been described quite compellingly by Lawrence Lessig, Richard Stallman, and Eric Raymond to name but a few.

Lawrence Lessig, author of several books that explore the great societal benefits of *commons*, makes one of the best cases for free software in his book *The Future of Ideas*. The idea is that a large pool of freely usable code forms a highly valuable and useful "commons" from which all programmers can take freely when building new programs. Developers who take advantage of this commons are freed from concerns about copyrights and patents, and from constantly having to re-invent the wheel. Lessig doesn't ever make the claim that *everything* should be shared in common. His point is rather that people should become aware of the great benefits to *all* by sharing certain types of resources. In Lessig's view, modern copyright and patent laws have become too powerful, giving powerful business and industry leaders an unfair advantage over the public. Perhaps a more compelling point though, is that these tyrannical practices are often injurious not only to the public but also to the industries themselves - it's hard to make progress or introduce innovation under the current regime.

Richard Stallman is more concerned about the *spiritual* effects that proprietary development has on programmers. In the *GNU Manifesto*, Stallman explains how the proprietary model allows programmers "to make more money, but… requires them to feel in conflict with other programmers in general, rather than feel as comrades." The secrecy and inability to share useful programs with their friends outside the company breeds a certain cynicism and pessimism that ultimately proves corrosive to a society in the "information age".

> Source code is far better than resumes or recommendation letters for showing a potential employer that you have what it takes to contribute to important projects

While Eric Raymond seems to prefer economic benefits of free software over philosophical concerns, he nevertheless shares Stallman's belief that free software allows for a more positive and fulfilling working environment than proprietary: "We're proving not only that we can do better software, but that joy is an asset" (60). One need only glance at some of the unpleasant news coming from *Electronic Arts* these days to be rest assured that free software is a blessed alternative to sacrificing one's principles for the sake of another man's profit.

I will end this section with an observation that may seem questionable at first: programmers have a unique responsibility to society and should think about their job in moral terms as well as economic. Programmers shouldn't be mere technicians doing unimportant, thankless drudgework. Rather, they should realize their critical importance and responsibility in a world gone digital. They are in many ways similar to the priests and monks of Europe's Dark Ages; they are the only ones with the training and insight to read and interpret the "scripture" of this age. Though powerful business leaders have tried hard to devalue their creativity and force them into working in demeaning conditions, programmers ought to recognize that *they* have the power to change this situation. No modern business or industry could survive without the diligent assistance of programmers. It's time that programmers became cognizant of this fact and used this leverage to make the world a better place - they have a responsibility not only to themselves, but to a future society in which their children will thrive or suffer. Freedom, openness, and sharing are not merely desirable; they are *essential* for the future of democracy and ensuring that our descendents emerge as *citizens*, not servants.

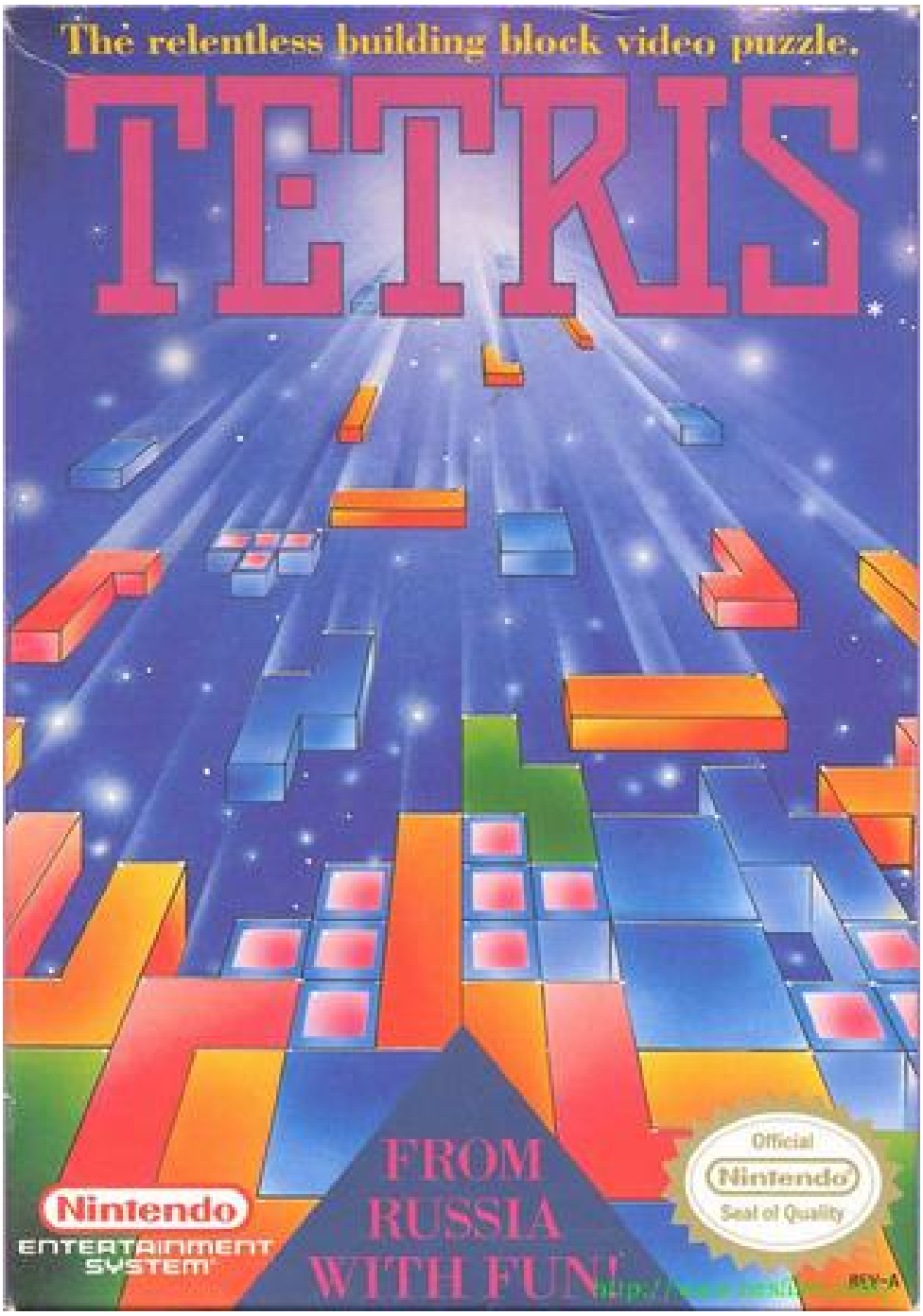
Platforms and permanence

I will finish this piece with a consideration that ought to appeal to any programmer's ego: *Permanence*. While most of the coding performed by programmers may be routine or mundane, other projects have a much different feel to them. Sometimes a programmer will be seized by an idea so exciting that it is difficult to keep her fingers still enough to enter the code. These are the landmark projects; the paradigm shifts; the software that we can only compare to true works of art. These works deserve our best efforts at preservation and are simply too valuable and precious to be controlled by any single entity or corporation. They are glorious gifts that should be rightly bestowed upon an eager and appreciative public, who will long remember the contribution and catapult the programmer's name to fame and history.

Let us take a handy example of such an event: Alexey Pajitnov's *Tetris* game, first released in 1985 in the Soviet Union. The game was so original and compelling that it clearly represented a breakthrough. Unfortunately, Pajitnov was living in a communist country that *supposedly* valued the public sharing of resources, *Tetris* represented too large a cash cow to be "sacrificed" to communist principles. The Soviet government claimed control of Pajitnov's game and made money by licensing it to publishers in other countries. The game sold extraordinarily well and made millions for the corporations that licensed and published it. The game made countless fortunes, but not for Pajitnov. Nevertheless, his name will likely live forever in history as one of the game industry's most influential innovators.

Let us assume that Pajitnov had lived in the United States in 1985 and had submitted his game to a commercial software publisher, such as the Nintendo Corporation. Would *Tetris* have had the impact it had if this had been the case? I very seriously doubt it, because, Nintendo would have undoubtedly been better able to leverage its "intellectual property rights" to generate more profit for itself at the expense of having the game reaching less players. It is likely that they would have taken a fist-of-iron approach to the hundreds of clone makers. Furthermore, the game would *only* be avail-

Fig. 3: Tetris Box – No mention of Pejitnov here!



able for Nintendo's own platforms (or licensed at exorbitant prices for computer software makers). Finally, if this had been the case, Pajitnov would not now have the rights to his game; those would have been assigned to Nintendo forever at the outset.

It's really anybody's guess whether *Tetris* would be as popular as it is today if Nintendo had been granted monopoly rights to its distribution. Several other NES originals, such as *Super Mario Bros.* and *Legend of Zelda*, remain popular today, and Nintendo has made them available for its newer platforms. Still, it's undeniable that even these games would be more accessible if Nintendo had released them into the public domain or under a public license. Of course, doing so would cost Nintendo some valuable "intellectual prop-

erty", but, then again, is that really a concern for the teams that created these games? I doubt most people in the street would be able to name a single person who assisted in their development - and what happens if Nintendo goes bankrupt (and its assets get tied up in a legal morass for decades) or decides not to release these titles on future hardware? Developers with heroic aspirations have to keep these possibilities in mind.

The situation is even more grim for developers for computer applications. Sure, Microsoft's *Windows* enjoys greater market share than GNU/Linux or other competitors. Nevertheless, even Bill Gates seems surprised at times that his corporation has achieved such great success and has held it for so long. Meanwhile, the United States government and plenty of foreign governments have taken Microsoft to court for monopolistic practices, and while Microsoft has endured, these attacks are unlikely to cease or grow less threatening. A developer who chooses to work strictly with Microsoft's own development software and proprietary tools must consider whether her projects - especially those "paradigm shifting" mentioned earlier - are really worth risking on a closed platform. Indeed, at this stage of the game, a true "killer app" for *GNU/Linux* would seem more likely to vault a programmer into the annals of history than a comparable application for *Windows*, where it would likely get lost in the sea of competing commercial applications. It is *certainly* true that an application, which threatened to significantly alter the way we use computers would seem a dangerous threat to an established corporation *whose future depends on maintaining the status quo.* Consider briefly how the proprietary software industry has responded to developments like peer-to-peer networking. Where did the majority of "killer apps" for the internet come from? Today, we know Tim Berners-Lee as the "inventor of the World Wide Web", but we scratch our heads when someone asks us who developed Apple's *HyperCard*, a stunningly original application that in many important ways was a progenitor of hypertext.

Bill Atkins must have anticipated this sad fate for his groundbreaking program when he insisted that Apple would release his program for free on *all* Macs. Apple chose to ignore this agreement when it released the next version of his program. What if Atkins had released *HyperCard* under a general public license? Or perhaps placed it into the public domain? Tim Berners-Lee was knighted by Queen Elizabeth in 2004. Who's Bill Atkins, again?

Concluding thoughts

The risks of writing proprietary software are many, and the sole benefit - quick cash - seems to pale in comparison to the many, longer-lasting benefits of writing free software. A truly wonderful program released under a free software license is much more likely to earn a developer prestige, reputation, influence, and fame than a comparable proprietary program. Besides these personal benefits, there are also societal benefits that are impossible to ignore by men and women of integrity. A developer intent on really making a difference ought to consider whether history shows that tyranny is superior to freedom; if feudalism is better than democracy. Surely, the history of the United States offers evidence that it is only when people are allowed *to be free* that they are also allowed to truly prosper. The same is visibly true of software. Freedom sells, and the future is buying.

Bibliography

[1] Lessig, Larry. *The Future of Ideas: The Fate of the Commons in a Connected World.* New York: Vintage Books, 2002.

[2] Raymond, Eric S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Cambridge: O'Reilly, 2001.

[3] Stallman, Richard. "The GNU Manifesto." *Free Software, Free Society: Selected Essays of Richard M. Stallman.* Ed. Joshua Gay. Boston: GNU Press, 2002. 31-39.

Copyright information

About the author

Matt Barton is an educator and writer, who is currently living in Tampa, Florida. He is an advocate of free software and the Creative Commons. He hopes to receive his Ph.D. in May 2005.