6-2009

Free, Source-Code-Available, or Proprietary: An Ethically Charged, Context-Sensitive Choice

Marty J. Wolf
Bemidji State University

Keith W. Miller
University of Illinois at Springfield

Frances Grodzinsky
Sacred Heart University, grodzinskyf@sacredheart.edu

Free, Source-Code-Available, Or Proprietary: An Ethically Charged, Context-Sensitive Choice

Marty J. Wolf
Bemidji State University
mjwolf@cs.bemidjistate.edu

Keith W. Miller
University of Illinois, Springfield
kmill2@uis.edu

Frances S. Grodzinsky
Sacred Heart University
grodzinskyf@yahoo.com

Abstract
We demonstrate that different categories of software raise different ethical concerns with respect to whether software ought to be Free Software or Proprietary Software. We outline the ethical tension between Free Software and Proprietary Software that stems from the two kinds of licenses. For some categories of software we develop support for normative statements regarding the software development landscape. We claim that as society's use of software changes, the ethical analysis for that category of software must necessarily be repeated. Finally, we make a utilitarian argument that the software development environment should encourage both Free Software and Proprietary Software to flourish.

Introduction

Much of the ethical analysis of Free, Libre, and Open Source Software, either separately or collectively, focuses on its differences from proprietary software. Chopra and Dexter give a particularly comprehensive analysis in their book [Chopra and Dexter, 2008]. Typically, in these analyses, all contexts of software use have been treated as ethically equivalent. It is not uncommon to find a specific example given in support of a particular claim with an implicit suggestion that the argument can be extended to all kinds of software. In this paper we explore the notion that some categories of software elicit different ethical concerns than software in general. We use the term *software category* to refer to different software that accomplishes roughly the same purpose. Some categories will admit subcategories. For example, the category of "Productivity Software" includes the subcategory of "Word Processing Software." When we talk about *software type*, we are referring to whether it is Free Software (FS), Proprietary Software (PS) or Source Code Available Software (SCAS). By analyzing the ethics of FS, SCAS, and PS in the context of a particular category of software, we develop a more nuanced approach that allows a more focused ethical analysis of each category.

For the purposes of our paper, we define Proprietary Software as software that is developed by either an individual or company and is made available to the public and other businesses only in binary form. Furthermore, it is licensed in such a way as to restrict its further distribution and to prevent reverse engineering.

An earlier version of the paper was published in Proceedings of Ethicomp 2008, University of Pavia, Mantua Italy, ed: Bynum, Calzarossa, De Lotto, Rogerson, 2008.

In order for a piece of software to be considered Free Software (FS) it must be licensed under an approved Free Software License [FSF License, 2007]. The Free Software Foundation promotes the four software freedoms to provide clarity about what those licenses are designed to accomplish [FSF Definition, 2007]. Briefly, software that is "free" affords four freedoms:

- Freedom 0: The freedom to run the program, for any purpose.
- Freedom 1: The freedom to study how the program works, and adapt it to your needs.
- Freedom 2: The freedom to redistribute copies so you can help your neighbor.
- Freedom 3: The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

In addition, FS usually includes the notion of "copyleft," which uses copyright law to ensure that all derivative works of FS are also FS. This is accomplished by licensing software with a clause that requires all derivative software be licensed with the same license [FSF Copyleft, 2009]. We include this notion in our sense of FS.

We will call all software that has its source code available in a standard electronic format, Source Code Available Software (SCAS). Freedoms 1 and 3 require access to the source code, so FS is also SCAS. However, there is SCAS that is not FS, including some Open Source Software (OSS), which also has a formal definition that requires more than source code availability [Coar, 2007]. We focus on FS, PS, and SCAS, and include non-free OSS in SCAS.

In the remainder of the paper we make the claim that the *category* of software is a relevant factor when attempting an ethical analysis of the software. First, we explore the relevance of the software's *type* by examining the ethical tension between FS and PS that stems from the two different kinds of licenses. Next we consider a variety of *categories* of software and, using the lenses of FS, SCAS, and PS, develop support for normative statements regarding the software development landscape. As we consider some *categories* of software, we accumulate evidence to support the claim that as society's use of software changes and the role of software in society changes, the ethical analysis for that category of software should be repeated. We conclude by noting that an overarching ethical concern surrounding software development is ensuring that the software development environment has the agility to allow both FS and PS to flourish.

Critiques

In some sense, FS is itself a critique of PS. The FS definition, its GPL licenses and many supporting documents argue that FS preserves the autonomy of individuals and the cohesiveness of communities that use the software. There is a strong emphasis on users' rights over developers' rights in the FS definition. It is precisely this emphasis that leads to some notable critiques of FS.

In an earlier paper [Grodzinsky and Wolf, 2007], we examined some ethical critiques of FS. They largely fall into two groups. Both groups of critiques involve how under FS licenses FS software authors have to relinquish some of their rights. In the first group, the critiques note that any software author who has dedicated time, talent, and energy to develop software, at least deserves control over his/her creation, if not some financial reward as well. Therefore, the argument goes, an author who writes truly original

software should not be required to give up both control and future financial rewards by making their software into FS. This argument is not unique to software as Himma and others make a similar argument for the more general case of intellectual property [Himma, 2006]. A closely related argument is that authors who make "value-added" changes to existing FS should not be coerced by the FS license into making the changed software FS as well. Watson, among others, argues that there should be no requirement of quid pro quo. Just because a developer is in a position to take advantage of the software written by others, does not mean the same developer must return his/her contributions to the free software development community. Watson objects to copyleft as a coercive system [Watson, 1999].

The second group of critiques centers on a concern that some FS borrows heavily from existing PS, with much of the user interface and many of the ideas for features co-opted from PS. There is a sense that these ideas have been taken unfairly from the original creator. This tension over the co-opting of features is manifested particularly when, by default, a category of software by the very nature of its use requires openly exposing features of the software. Software with this property includes application software such as a word processor or a spreadsheet, and software that uses any sort of network communication protocol.

Much of the ethical tension between FS and PS stems from the tension between the rights of the users of software and the rights of the developers of software. We explore this tension by looking at different categories of software.

Software Categories

The two groups of critiques of FS offer starting points for analyzing the question, "When should software be Free?" In this analysis, the user and the developer of a piece of software each bring different ethical considerations to the table, and the power structure between the user and the developer also impacts the ethical analysis.

We have not found a compelling ethical argument that *all* software should be FS. Similarly, we do not think there is a compelling ethical argument that all software should be PS. Instead we find that our ethical analyses vary depending on the category of software under analysis. As we examine different software categories to understand better the ethical issues pertaining to the categories, we examine these kinds of questions: Should software in this category be FS? Should it be PS? Is SCAS sufficient to satisfy ethical concerns? Have the ethical arguments changed due to the changing role of the software in society?

3.1 In-House Software for Internal Use

As a straightforward case we consider software that is developed in-house for internal use by a company. Because the software users and the software developers are (at one level of abstraction) the same entity--the company, although different people in the company can separately fulfill the roles of developer and user--the ethical tension is greatly diminished. Even the most ardent supporters of FS concede that in-house software need not be free to people outside the company. (FS advocates would certainly be *pleased* if such in-house software would be made into FS, but they do not see it as a moral imperative.) A key observation is that this software is not distributed outside the company, and those outside of the company have no special rights to that software. The

software is developed by the company, runs exclusively on the company's computers and does not interact with outside users or outside data.

3.2 In-House Software Used as a Service by Remote Users

A slightly different category of software is FS software that is modified in-house and used internally by a company without releasing the modifications back to the FS community. Typically, the FS community has accepted this type of use since the FS license is focused on how software is distributed and the copyleft provisions do not "kick in" until distribution takes place. However, there are web-based services that allow the public to run this "internal" software on the company's computers. A client submits data via the web to the company, the software runs on the company's computers and the results are sent back to the client via the web. (Web-based email, calendaring and task management software are commonplace examples.) It seems ethically uncontroversial that a company could use either PS or FS on its internal machines to service the remote users. However, an interesting issue arises when the company obtains a piece of FS, modifies it to service remote users, but does not release the modified version as FS.

Software used this way does not meet the legal definition of being "distributed," so the company is using the software in a way consistent with the legal requirements of FS. Even so, there are those within the FS community who find FS modified and then used as software-as-a-service objectionable. This issue was an important consideration during the development of version 3 of the GNU General Public License (GPLv3). It seems that even within the FS community there is not universal, and perhaps not even widespread, support for preventing the offering of modified FS as a service. However, a second form of the GPL, the Affero GPL, was developed to address these concerns directly. (See [Miller, Wolf, and Grodzinsky, 2008] for more information on the development of GPLv3 and AGPL.) Why is there uneasiness within the FS community about deploying modified FS as a service?

Allowing people to use FS at little or no cost has the potential to elicit feelings of good will both in the FS user and the FS developer. On the other hand, when people or organizations make money using modified FS without making contributions back to the FS community (declining, for example, to release their modifications as FS) there is a sense that those people or organizations are exploiting the original developers of that FS, acting as "free riders." The original FS developers shared, but these modifiers do not share. Again there is tension between the rights of the user and the rights of the original developers. In this case of server-side software offered as software-as-a-service, the users of the original FS who become the developers of the modified software, although adhering to the letter of the FS license, are viewed as violating the spirit of the FS community.

We find it interesting that the Free Software community that resents this type of "non-sharing use" of FS is making a complaint similar to the second group of critiques of FS by PS developers. Recall that PS developers object to FS that is similar in function and interface to a piece of PS because the FS does not compensate the PS developer for what the PS developer deems to be work that the FS conceptually relies upon. In the case of server-side software, assume that company X adapts a piece of FS, call it S1, producing a modified version, S2, and then uses S2 as server-side software. In this case, S2 is naturally quite similar to S1, in fact includes significant parts of S1. But because S2 is

not shared as FS, the Free Software community is not being "compensated" (X does not share S2) for the value of S1 used by X.

The situation in which the company X modifies S1 into S2, uses S2, but does not share S2, has similarities to PS sold to a consumer. In both cases, the developer has great power over the users. In the case of PS, the roles of developer and users are clear. However, in the case of FS, some care is required because the organization X is both a user (of the original FS S1) and a developer (of the modified software S2). With regard to S2, X dictates all of the terms of use of the software, while a remote user of S2 has no influence on the functionality of the software. As a user, X was free to use S1 in ways that are not available to remote "users" of the server-side software S2.

This asymmetry of X's use of S1 as FS and the development of S2 as non-FS is apparently legal under most interpretations of FS licenses. But ethically, the asymmetry is troubling. One way in which X can address this asymmetry is to release some improvements to S1 as a new FS version (S1*) as FS while still holding S2 in-house. S2, which holds information that X wishes to be a trade secret, is protected, but S1* is still a contribution to the Free Software community. Some organizations using modified FS in-house do participate in just this way (Shankland, 2008).

Returning to our original questions, our judgment is that in-house software used as a service for remote users can be either PS or FS at the option of the company on whose machines the software runs. If the software is modified FS, there is an ethical obligation for the company to contribute to the FS community in some meaningful way, an obligation that does not apply if the software used is PS or unmodified FS.

3.3 Client-Side Web Software

Web software can be divided into two broad subcategories: server-side software and client-side software. The software-as-service discussed in section 3.2 is an example of server-side software. Server-side and client-side software are distinct cases when analyzing questions about whether FS, PS, or SCAS is most appropriate. The location of where the software executes is crucial to our analysis. In the case of the server-side software, the software runs on computers owned by the company running the website. As such, the software does not have access to remote user's computers and data, except the data a remote user has sent to the server. Without further analysis of the purpose and activities carried out by the website, this case is identical to the analysis in section 3.2.

On the other hand, client-side software immediately has privacy concerns not prevalent in server-side software, and so requires additional consideration. Even a careful web surfer who does not supply personal information to any website is potentially subject to surveillance by the web browser as well as any software embedded in web pages. It is possible to imagine a government requiring that browser producers include surveillance software in browsers. The availability of FS web browsers seems to preclude such a requirement. Even if there were such a requirement, the nature of FS would allow individuals to remove the surveillance components prior to installing the browser. While the casual user may not choose to do so, the type of person who fears being watched would be highly likely to do so, making the surveillance software requirement relatively useless. Thus, the mere availability of FS in the browser market reduces the chances of a government from becoming overly intrusive through web browsers. Note that this

argument is not strong enough to require all browsers be FS; it only rises to the level of not preventing the development of FS web browsers. (A less clear consideration is whether the ethical argument is strong enough to require the development of at least one FS web browser.) Furthermore, it may be the case that there are other mechanisms to reduce the chances of a government becoming overly intrusive in this context. If so, more general arguments, such as developer autonomy, are needed for allowing the development of FS web browsers. However, any positive claim for a right or obligation to actively prevent the development of FS web browsers would, it seems to us, require a strong case, a case we do not see.

It is also worth considering whether all of the requirements of FS are really needed to discourage intrusive browsers; is SCAS sufficient to mitigate the concern? While a web browser that is SCAS would allow for the same improvements mentioned above, it is not clear that having the source is sufficient to ensure development of a piece of high quality software that is likely to be adopted by a large community of users. Given the complexity of web browsing software, the large user base and the potential for a large developer base, it seems reasonable to conclude that a FS browser will stay Free as it is improved (at least if people follow the license), whereas we have no such expectation of a SCAS browser.

Note that our normative claim about FS web browsers has an implication that extends beyond software. Since the browser must communicate with servers, there must be both protocols by which the communication takes place and a universal language so that all web browsers can reliably translate information into a properly displayed page. The ethical requirement, to allow the development of FS web browsers, demands that the protocols and languages used for web communication be open and available as well. If they were not available, there were onerous restrictions on their use, or the protocols were intentionally misleading and allowed back doors for surveillance, then the protocols would lead to privacy concerns for web browser users.

To this point we have been assuming that the web user has not been knowingly supplying any personal data to any of the websites being visited. What are the ethical implications when users are consciously supplying such data? It is reasonable for users to expect that such data be kept private and be used only for its intended purpose [Kobsa, 2007]. It is certainly the case that web server software that is SCAS would allow users to understand how their sensitive data is handled. While this transparency is preferable, this argument does not rise to the level of an ethical requirement for SCAS because even with SCAS, the data could be mishandled after it was delivered to the web site owner. In this case, there are other factors in play that typically encourage the holders of private data to treat it responsibly and privacy policies that articulate how the data is used. Kobsa discusses how the on-line business environment, in connection with privacy laws in various jurisdictions, has gone a long way to protect personalized information that organizations collect [Kobsa, 2007].

3.4 Malware

Malicious software can lurk in any kind of software. At least theoretically, FS allows users to examine, compile, and link their own executables, which could lead to greater security. However, some people claim that opening the source code to hackers encourages more sophisticated attacks on FS than on closed source PS. We will not

attempt an authoritative answer to the complex and ultimately empirical question "is FS or PS more vulnerable to malware?" Instead, we will assume that until that question is settled, either FS or PS can be used based on its vulnerability to malware or other attacks. However, if for a particular category it was demonstrated that either FS or PS was less vulnerable to hacker attacks, then that type of software would gain a strong practical and ethical (in a consequential sense) advantage. In the case of such a situation for a particular category, we can then envision a strong argument that the type of software that is more hacker-resistant *should* be used in that category of application. However, absent that demonstration, the threat of malware does not affect our questions.

3.5 E-government Software

Chopra and Dexter argue that a "system of e-government built on closed software is itself closed" [Chopra and Dexter, 2008, 166]. They argue that the closed software embodies laws and policies that are unknowable by the public and destroy the very nature of a participatory democracy. They draw particular attention to voting software. In the U.S. and elsewhere, there is a movement to eliminate PS from the voting process [Open Voting Consortium, 2006 and Zetter, 2003]. Chopra and Dexter contend that there is a strong ethical claim in favor of using FS for voting in any democratic system. FS is by design and definition more open and transparent than closed source PS. PS in this context, even if it is *not* being manipulated, cannot ever be as trusted by the general public as a functionally equivalent FS solution. This is not to say that FS, in itself, guarantees free and fair elections. But FS seems far more likely to enable fairness than PS simply due to the transparent nature of FS. However, FS is not necessary to achieve this sort of transparency. The transparency comes from the ability to inspect the source code. In fact, we defend the non-intuitive claim that some of the freedoms of FS are ethically undesirable in voting software.

There is little value to the electoral process in allowing the modification and distribution of voting source code. Voting software only has value on voting machines, which should be strictly under control of the government. The license to individually modify and distribute new versions of the software is contrary to democratic systems where laws and policies are drawn up by constitutionally defined processes. We contend that the ideal situation for voting software is that it be SCAS and not FS so that the software can be examined as part of a free and open democratic process, and that derivative software not be used without this open, public process. This, then, is our first example of a category of software we contend *should* be SCAS but should not be FS.

While voting software should be SCAS and not FS, we suggest another category of software that should be FS: government software that is used to analyze and implement policy and in its day-to-day operations. To be clear, we are not advocating access to sensitive data, rather to the code that handles that data. The point is that policy is embedded in code and transparency of software adds to the transparency of the government. To maintain openness and fairness, it is important that such software be FS. Anyone wishing to be critical of government's operations and analysis would have access to the same software tools as the government and could therefore explore alternative analyses and operations. Indeed, citizens could participate by suggesting software modifications that could increase efficiency and improve analyses by government

personnel. Berry and Moss contend that making e-government tools FS has the potential to increase participation in government [Berry and Moss, 2006].

Finally, we note that a government that makes the claim of openness and fairness would need to communicate with the people via open communication channels. Government websites that require citizens to use PS as part of the communication process seem to fail on this point. Closed software at any point in the communication channel remains an impediment to open governance.

3.6 Productivity Software

Productivity software typically refers to software that is generally used in an office setting to make individual workers more productive in their jobs. While there may be some argument as to what belongs in this category of software, it seems clear that word processing, spreadsheet and presentation software are all included. We use these three subcategories to discuss this category.

To begin, consider how productivity software was used prior to widespread use of email and the advent of the World Wide Web. An employee might develop documents and spreadsheets as mechanisms for sharing information internally. If there was a need to share the information, it was printed and sent to the intended recipient. In this situation there is clear ownership of the information as well as the document. There is little in this situation to lead us to consider ethical concerns external to the company. Productivity software was used for little other than increasing the productivity of employees, so the decision to purchase such software was mostly a business decision based on a cost/benefit analysis for the company. Such a decision seemingly had little impact beyond the company. In this case, PS, FS, and SCAS could be used ethically, and the existence of all three types of software is useful because it encourages innovation and competition.

Today, with the widespread use of email and the ubiquity of the World Wide Web, productivity tools are no longer used solely for the internal dissemination of information. In essence, the output from productivity software has become a communication mechanism. Individuals send each other word processing documents. Spreadsheets are exchanged between small businesses. These documents have become part of society's communication fabric. Note that there are two important concepts in the output of productivity software—both the document and the information contained within the document.

Ethical concerns in this environment are raised on three levels. Since the documents are communicated over some medium, there are concerns over the security of the medium. The second level involves the ownership of the information in the document. Privacy concerns and the appropriateness of communicating the information are at this level. Concerns at these two levels are beyond the scope of this paper. Assuming that the network is relatively secure and the communication is ethically appropriate, our ethical consideration, the third level, is whether the software for interpreting these documents should be free or proprietary.

Since this software is part of the communication infrastructure, some consideration must be given to the ability of people to communicate freely with one another. If one company was to control all communication via this method, in addition to the usual concerns about

monopolies, we would have to be concerned about the impact on free communication and the company exerting unwarranted control of the information contained within documents. Seen from this perspective, our previous argument about web browsers is relevant. One solution to this concern could be the use of open (and non-proprietary) communications standards that would allow all software to interact, including FS as well as PS software from multiple vendors. It would be beneficial to all users to have a seamless transition from Open Office to MS Office and vice versa. (Such cooperation is highly desirable for users of both FS and PS, although it may not be in the best economic interest of PS vendors.) Having multiple competing products, proprietary or otherwise, is not sufficient in this situation. Since this is a communication mechanism, both the receiver and the sender must be in a position to encode and decode messages in order for there to be effective communication. Thus, as is the case with the World Wide Web, productivity software best serves people when there is a well-defined and open standard for storing information that is to be shared in this way. Furthermore, an open standard produces an environment that decreases the threat of monopoly and the potential abuses of that monopoly.

The reader might object that such standards are unlikely to occur and would be impractical if they did. However, the success of the open HTML and XML standards demonstrates the viability (though not the inevitability) of such standards. The existence of standards such as JPEG is another example of an existing open standard; however, there are can be difficulties in converting from a proprietary encoding to an open standard like JPEG. Aspects of the native object may be lost, or expensive new space may be required for the transformed file. When these costs proliferate, users may give up on the open standards and surrender to a more convenient proprietary standard, whose convenience increases in proportion to its market share. Standards decisions that tend to reduce the possibility of fair competition between both PS and FS alternatives for productivity software are, we contend, unethical.

In summary, PS, FS, and SCAS are all useful and ethically appropriate choices for productivity software. However, to make sure all are viable and can compete on a reasonably level playing field, open standards for documents, messages, and other protocols are required.

3.7 Gaming Software

Gaming software is an increasingly lucrative field. Is there an ethical imperative that gaming software be FS? We think not. It may be that some users will have a strong preference for FS for gaming (because of price, the opportunity to fine-tune the software, or for community responses to requests for changes), and other users may prefer PS for gaming (paid support, the lure of hidden secrets in the code, and commercial add-ons). Although some users heavily invest their time and money into gaming software, our position is that these users can decide for themselves whether FS or PS fits their gaming needs. We do not see a compelling ethical case for requiring FS, SCAS or PS for gaming.

3.8 Operating Systems Software

An operating system is software that can affect users significantly. This dependency adds weight to questions about whether operating system software should, ethically, be exclusively FS or PS. We contend that the issue of trust is decisive here. Convenience,

features, and price can be competitive in FS and PS in an operating system, and we see no ethical reason to prefer one or the other based on these characteristics. In addition to convenience, features and price, an operating system user might prefer an operating system that is reliable (especially in the long run after the user comes to rely on its familiarity), holds no hidden malware, and is maintained sufficiently both now and in the foreseeable future. Again we are faced with an empirical question that must be answered prior to answering a question about whether an operating system ought to be either FS or PS: does FS or PS have a consistent advantage over the other in being reliable, secure, and maintainable? Unfortunately, each of these points is controversial, with FS advocates asserting the advantages of free operating systems, and PS advocates asserting the advantages of proprietary operating systems. Absent an authoritative answer to these competing claims, we cannot make a consequential argument based on reliability, security, or maintenance.

There is, however, at least one other way to examine the ethical nature of operating systems: through the prism of "trust." Insofar as operating systems are relied upon for frequent and fundamental electronic services, there is a need to *trust* the software, at least in some abstract sense. In a more concrete sense, operating system users depend upon the operating system developers. The operating system will fulfill the goals above only insofar as the developers are diligent in delivering reliable, secure, and well maintained software. Are FS developers more trustworthy than PS developers, or vice versa? Advocates can argue that FS and SCAS operating systems are far more worthy of trust, since their developers are willing to share the source code with users, and PS developers are far more secretive. Furthermore, FS advocates can point to the altruistic spirit of software developers willing to share their expertise in a voluntary community. However, PS advocates counter that it is difficult to trust an amorphous group that changes rapidly. PS is written, the argument goes, by professionals whose reputation and livelihood depends on delivering value for money. PS developers are not software hobbyists.

There is a FS counter to this PS argument: FS developers are an open community where peer reputation is paramount. PS is developed by companies and perhaps the reputation of the company more than the developer is at stake. Granted, a professional may lose a job based on poor performance, but given the way PS is developed, the modules you write may contribute to the whole, but it is not generally an individual who is getting the credit (or blame) for software. In FS, you more often know (or can discover) who has done the work.

The trust perspective, then, transfers the question of "PS or FS" to a question of "whom do you think is more likely to be trustworthy: developers cooperating in a volunteer community, developers who allow you to inspect their code, or developers paid by proceeds from PS?" As with several categories above, there are plausible arguments for each of these positions; there are instances of FS and PS that have demonstrated long-term trustworthy developer behaviors, and there are instances of FS and PS that have disappointed users, often by dropping support for software depended upon by users. Therefore, we do not foresee a compelling argument that FS, SCAS or PS is clearly preferred ethically for operating system software.

Conclusion

Some advocates of both FS and PS have made general declarations that categorically extol the virtues of their type of software or declare the vices of the other type of software. We think that any such blanket categorical declaration can not accurately reflect the strengths, weaknesses, successes and limitations of FS, SCAS and PS. We have presented ethical arguments that some applications should be FS, that some applications should be PS, some should be SCAS, and that some applications ethically can be any of the three types. For other categories of software, the existence of a FS alternative can be ethically useful, although we don't see a compelling case to mandate that all such software must be FS (for example, browser software). Furthermore, the ethical tension that exists among FS, SCAS and PS is useful because it serves to keep ethical considerations closer to the forefront for software developers. Public arguments about the quality of FS and SCAS versus the quality of PS in a particular category encourage all developers to be mindful of software quality.

When no strong ethical argument mandates FS, SCAS or PS in a particular category, there may be compelling reasons for an individual to prefer to FS, SCAS or PS. Someone convinced that a particular PS application, currently unchallenged by a viable FS alternative, was using its monopoly unfairly, could make a case that developers *should* provide a FS alternative. A developer with pressing financial obligations (perhaps to family members) might decide it more ethical to pursue the more obvious profitability of PS instead of working on FS where financial rewards are unlikely to match those possible with PS.

But in many other cases, it seems that a developer has a choice to participate in FS or not, and this participation need not be exclusive. An individual developer could (and many do) develop PS, SCAS and FS. Companies as well can simultaneously be involved in FS, SCAS and PS. And in many cases, these "dual-citizenships" are ethically appropriate, economically fruitful and technically advantageous.

The context sensitivity of software suggests that discussions about the ethics of software need to be nuanced in order to deal with the complexities discussed above. Furthermore, the complexities for certain software categories change over time, demanding that ethical analyses be re-evaluated when society uses software differently than when it was first introduced. Advocates for both FS and PS should be carefully about their claims when the claims include ethical obligations, especially for others. Considering in detail the way the software will be used is essential for making careful ethical analysis about software.

References

Berry, D.M., and Moss, G. (2006), Free and open-source software: Opening and democratizing e-government's black box. *Information Policy.* 11, 21-34.

Chopra, S. and Dexter, S. (2008), Decoding Liberation. Routledge: New York.

Coar, K. (2007), The Open Source Definition. (July. 2007),

http://opensource.org/docs/osd, accessed 10.01.2008.

FSF Copyleft. (2009), http://www.gnu.org/copyleft/copyleft.html, accessed 16.03.2009.

FSF Definition. (2007), http://www.gnu.org/philosophy/free-sw.html, accessed 9.01.2008.

FSF Licenses. (2007), http://www.fsf.org/licensing/licenses/, accessed 9.01.2008.

Grodzinsky, F. and Wolf, M.J. (2007), "Ethical interest in free and open source software," in The Handbook of Information and Computer Ethics, K.E. Himma and H.T. Tavani, eds. Wiley, Hoboken, NJ.

Himma, K.E. (2006), "Justifying Intellectual Property Protection: Why the Interests of Content-Creators Usually Wins Over Everyone Else's" in Information Technology and Social Justice, E. Rooksby and J. Weckert, eds., Idea Group.

Kobsa, A. (2007), Privacy-enhanced personalization. *Commun. ACM* 50, 8 (Aug. 2007), 24-33.

Miller, K., Wolf, M.J. and Grodzinsky, F. (2008), GPLv3, Freedom 0, and the Free Software community, manuscript.

Open Voting Consortium. (2006), Open Voting Solution. http://www.openvotingconsortium.org/our_solution, accessed 21.01.2008.

Shankland, S. (2008), Q&A: Google's open-source balancing act. CNet News (May 28, 2008), http://news.cnet.com/8301-13580_3-9952719-39.html, accessed 17.06.2008.

Watson, B. (1999), "Philosophies of Free Software and Intellectual Property," http://www.ram.org/ramblings/philosophy/fmp/free-software-philosophy.html, accessed 21.01.2008.

Zetter, K. (2003), Aussies do it right: E-voting. *Wired* (Nov. 3, 2003). http://www.wired.com/techbiz/media/news/2003/11/61045, accessed 21.01.2008.