Open Source Policies and Processes For In-Bound Software

Karen F. Copenhaver[a]
(a) Choate, Hall & Stewart LLP

DOI: 10.5033/ifosslr.v1i2.27

Abstract
Virtually all companies use some amount of software that has been made available to the company under an open source license. The re-use of these valuable software components is inevitable and should not be discouraged. In addition to reducing the time and cost of development, re-use of components that have withstood the tests of time and the many eyeballs of critical peer review provides assurance of quality and reliability. As companies begin to recognize the enormous value of this software, they come to the realization that they must implement policies and systematic processes that assure compliance with the terms of open source licenses. This includes both managing the use of open source software within the company and external obligations arising under relationships with open source communities.[1] This article focuses on policies relating to in-bound code.

Keywords
Law; Free and Open Source Software; Governance; Compliance

Info
This item is part of the **Platform** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement.

The most successful compliance programs are not replicated from a form or copied from another company. They are organically developed within an organization to fit within that company's existing internal control mechanisms. The exercise of developing the policy and processes is an important part of the organization's preparation for adoption and deployment. As a threshold matter, companies must understand the critical distinction between policy and process. The term "policy" refers to a set of company values that should not change over time. Such values are aspirational in nature and need to be supported by processes that implement the core values. Processes do, and should, change as frequently as necessary to reflect the development and growth of the organization's business.

Participation in the development of the policy and process assures personal and organizational buy-in and results in an efficient process that does not die of its own weight. It makes it possible to optimize the process before it is rolled out, rather than relying on iterative fixes following serial failures that may result in improvements but at the same time undermine confidence and

1 Such relationships typically include reciprocal contributions of software under an open source license, employee participation in open source projects, and commitments to open source communities.

commitment.

Open Source Policies

I divide compliance programs into two parts – a policy statement and a written business process to manage the company's compliance with the policy. This section discusses the design of company policies; their implementation through company processes is discussed in the following section. Think of your "policy" as the part of your company's written vision and value statement that will absolutely not change with time. Separating the changing from the unchanging gives more power to the permanent commitment and avoids the appearance that compliance is something akin to situational ethics. Here is an example of a policy statement for a software vendor that could be issued and remain unchanged over a very long period:

> XYZ respects the intellectual property of others and expects others to respect its intellectual property. All company personnel must operate within established procurement processes for the acquisition of intellectual property assets, including software [and trademarks], for use in XYZ's products or internal operations.
>
> XYZ's proprietary software and intellectual property are key assets that contribute unique value to the company. The impact of the introduction of any code licensed from external sources into XYZ's propriety software must be fully considered before its incorporation into the development process and, if incorporated, compliance with the terms of the applicable license must be achieved. It is required that XYZ maintain full accounting for all licensed materials that are included in products commercially distributed and sold by the company.
>
> This policy applies to all licensed materials, regardless of the method of procurement. Software that is available for download requires the same level of review and consideration as software acquired from a commercial company pursuant to a formal contracting process.
>
> Any questions or concerns regarding compliance should immediately be addressed to _____ [Title such as the VP of Engineering].

The rest – anything that is fluid – is all process. Why the distinction?

- Because companies should not violate their policies. Company policies are commonly intended to be immutable concepts that provide fundamental guidance in the form of "thou shalt nots."
- A company undermines the effect of all of its policies if any policy is not consistently applied.
- If you are in litigation, you do not want the other side to demonstrate that your actions were in violation of your own policy. You don't want to be "hoisted with your own petard."
- An overly dogmatic policy may also be poorly received by developers.
- Because a change in a policy implies that the policy was wrong to begin with and because most companies' use of open source will change over time, the policy should anticipate a changing set of internal and external norms and requirements.
- A "one-off" decision should not be a violation of company policy; it should be an anomaly that is specifically supported by the process as implemented.

A particularly illustrative example of a statement that should *not* be included in any policy is something like: "It is company policy not to use any code made available under the GPL." Here, the point is not whether the GPL is good or bad. Indeed, the company may, for whatever reason, prefer to choose other alternatives when all else is equal, but that does not amount to a flat-out prohibition against using GPL code. And even putting aside the thorny issue of using GPL code directly in the company's proprietary code base, there are too many invaluable tools and programs made available under the GPL that can be used to facilitate internal development appropriately and in full compliance with the applicable license obligations to impose a blanket prohibition without an exception process. Almost all companies that develop software use tools, such as GCC, the GNU Compiler Collection, that are made available under the GPL, and most companies use GPL licensed code such as the Linux operating system in their internal operations. Issuing a policy that prohibits all use of code licensed under the GPL is highly likely to be violated on the date that it is issued. Thus an issue has been created because the company has a policy violation even if the company is in full compliance with the license. Rather, the important point is that policies adopted without thoughtful consideration of the issue will fail almost immediately.

Essential Elements of Successful Open Source Processes

1. Assignment of Responsibility for Decision Making

Probably the most important indicator of whether a process carrying out a policy will be successfully implemented in a company is whether there is a clear statement of personal responsibility for every part of the process. While you are reading below the details of structures and decisions, I am sure you will be thinking: "Why would you spend so much time dividing the analysis into all of these different parts?" Your own analysis may differ, but, in my experience, the answer to why a process is not working is somewhere in these structural issues – even though the structural issues are often only reflected in personal conflicts.

Here is an example of a problem that, in reality, is caused by a lack of structural clarity but often manifests itself in the context of a personality clash. Let us say an engineer named John is asked to gather information about an open source project and its potential use and to analyze its pros and cons. John works with development management, and a careful decision is made in favor of using the project. But that decision gets rehashed and remade all the way up the chain by people who do not know or understand the technical analysis that went into the original decision or who incorrectly feel it is their responsibility to make an in-depth analysis. John feels angry and dismissed, and complains to everyone who will listen that he will never take the process seriously again.

If we assume that, in the end, it was the right decision not to use the project, how would you fix the problem? I suggest that the best solution is to make it clear that everyone in the chain is not actually rehashing the same decision, even if that appears to be the case. Rather, what is happening is that everyone is making *different* decisions based on their own knowledge and expertise. When the responsibility for making different parts of the decision is not clearly divided and assigned, it is likely that everyone in the chain will re-visit all of the different parts of the analysis and will be unnecessarily risk-adverse. Frequently, everyone in the chain is authorized to say "no" and no one is comfortable saying "yes." Therefore, identifying everyone's specific role in the process can be essential to empowering everyone involved in that process to make an affirmative decision. Furthermore, doing so makes it clear that everyone's contribution to the process is respected. For example, the lawyer should not be re-making technical decisions about component selection if the development manager has been clearly assigned responsibility for that task. Likewise, the development manager should not be making a decision based on his

assessment of the likelihood of a patent claim if that role is reserved to individuals tasked with corporate-wide risk management relating to patent matters.

There are many potentially successful structures, but decisions are commonly made either through a vertical or a horizontal process. A vertical process starts with a decision and recommendation made at the operational department level and specifies an orderly sequence through a series of chairs to confirm or overturn that decision. Each step in the sequence involves a single individual; there are no groups or boards that meet to consider the question. The last person in the sequence verifies that all the others have signed off and gives the final approval. The original request by the department is either confirmed or denied. No one in the chain exercises their own judgment regarding an alternative path. If the answer is no, the department goes back to work to develop another proposal and request. For example, the following is a common vertical process in a small organization:

- A non-management developer initiates a request to use an open source component. He or she has the responsibility to gather information about the open source project, the applicable license, etc. He or she provides that information to the development manager.
- The development manager assesses the usefulness and quality of the project, the time savings that would be achieved using that code, etc., and decides whether the choice is appropriate from a development perspective.
- The development manager consults with in-house or outside legal counsel who reviews the license, the proposed use and the business objectives, and determines that the license is appropriate for the use. The legal counsel's review is limited to matters within his or her expertise as a lawyer and confirmation that the company's internal processes have been followed.
- The development manager gets business unit management approval – probably without a meeting or presentation - based on confirmation that company policies and processes have been followed.

A vertical process is possible when issues are well-defined and when the impact of a decision is limited to a single business unit. The decision making and the process is largely in the control of the department making the request.

In contrast, a horizontal process provides for decisions to be made based on simultaneous input from multiple stakeholders. In a horizontal process the operational department requesting permission is not the decision maker. The impact of the decision goes beyond that department. Others are empowered to impose an alternative solution that is preferable for the organization. The answer to the original request may not be yes or no, it may be an entirely different plan of action. For example:

- The development manager gathers information and makes a business case to an Open Source Review Board on the pros and cons of using an open source component.
- All of the stakeholders across departments in the decision are represented on the Open Source Review Board, and the decision is made through the back-and-forth of committee deliberations. The committee reaches a consensus around the committee's recommended plan of action – which may be entirely different from the plan proposed by the department.

While both structures can work, processes that are not clearly defined are neither or both and the result is increased frustration at the requesting department level. The individuals in the requesting department have made a decision and know the facts regarding the request better than anyone else

in the organization. They want that decision confirmed unless there is a specific identified corporate policy that requires denial. They will assume the process is vertical unless expressly told otherwise. But an undefined process is likely to be conducted as though each rung in a vertical process is akin to an informal committee meeting resulting in a full review by participants acting outside of their expertise. A clearly defined horizontal process avoids this frustration by putting the department on notice that they are advocates for their request but not the decision makers. It also establishes a formal committee process with clearly defined areas of responsibility for everyone included in the meeting.

2. Who Initiates a Request and Who Gathers What Information?

Designers of effective processes need to clearly delineate the discrete tasks of gathering information and acting on that information. As discussed above, frequently the individuals tasked with collecting and presenting information believe—incorrectly—that they are responsible for making decisions based on their findings. If that is indeed the intended process, the process description should state so explicitly. Otherwise, those whose task is to gather and present information will provide only their conclusion and the information that supports that conclusion.

3. Who Makes What Decisions?

Generally, there are at least four components of any decision to use licensed code in your product.

- First, is it appropriate to have a dependency on code that is not owned by the company for the purposes of this product? The first decision is whether the company wants, or needs, to control the functionality to be provided by the code. Relevant considerations in that decision include: security, creation of a dependency on a format or standard, possibility of acquisition of the code owner by a competitor, difficulty or expense of removing the code after development on top of it has begun; and impact on product roadmaps or design decisions.

The tolerance for using licensed code has grown considerably over the years. In fact, an assumption has evolved that re-using assets is a wise business decision—and that assumption will only become stronger over time. But whether using open source code is a thoughtful decision or just a practical assumption, the first decision that the company is making is that using code that is not completely controlled by the company is appropriate in this instance.

This is the first step in the open source risk analysis. It is based on the company's lack of control or the absence of any confidentiality. It is not based on events that affect the project in general or any other users of a project. It is a determination of whether developing a dependency on this code creates a vulnerability for this particular company and product plan.

- Second, assuming that using code you do not own is appropriate, is this the right code? Does it provide the right functionality for today and is it extendable to provide the required functionality for tomorrow? Is it good code? Is it documented? Is support available, and, if so, from whom? Is there a compelling commercial reason for the support to be maintained? For example, the risk of using an open source project that is hosted by a commercial software company may be different from the risk of using code developed by an unincorporated project that is supported by many companies. The sole supporter may lose enthusiasm and there may be no other enthusiasts to pick up where the company left off. And if it is likely that support will be unavailable in the future, does it make business sense for the company to maintain the code in-house?

- Third, are the terms of the license for the code aligned with the company's business objectives? Can the company achieve its business objectives while complying with the license obligations? This is the lawyer's domain, where counsel can help the organization understand and prepare for the legal ramifications of business decisions. Here, most in-house lawyers have to conduct two lines of analysis: a legal interpretation of the license, and an analysis based on community consensus, or lack thereof, on the applicable license obligations.

Lawyers also have to determine (I) whether there is a process in place that will enable the company to stay in compliance in the future; (ii) the likelihood of inadvertent failure to comply with the license terms; and (iii) the impact of a compliance failure on achieving the company's objectives. If compliance is dependent on a specific set of facts, will a flag be raised if those facts change? For example, if compliance is dependent upon use of the code without modification, will a review be triggered if the code is modified? If achievement of business objectives, while remaining in compliance, is dependent upon limiting usage to internal application only, will a review be triggered if the code is distributed? This is the second step in the open source risk analysis, and involves internal risk that can be managed, rather than risk that arises from matters outside of the company's control. Companies with robust processes for managing compliance have more options for dealing with this exposure than companies that cannot be sure that implemented compliance policies will be maintained.

- Fourth, is the use of this code consistent with the company's tolerance for risk, its risk management practices, or both? This combined legal and business analysis is the third step in the risk analysis, based on matters outside of the company's control. This assessment is very similar for all code regardless of its origins. While the risk tolerance for open source in general has grown as open source has become more mainstream, risk analysis should always be project-specific. For example, the risk of using a project that is also used by many other industry leaders is very different from the risk of using a project that was long ago abandoned. Obviously, an established project is much more likely to have widespread support and resources if continued availability of the code and support is jeopardized.

This type of risk is not based on the applicable license, which means that companies that make decisions based entirely on the license will miss this part of the decision process. Who, if anyone, will assist in the defense of this code in the event of a patent or copyright infringement claim? What is the governance structure for the project? Has anyone reviewed the code for internal license conflicts? For example, since open source code does not come with any indemnification for intellectual property claims, is there some additional review of the code that should occur to make sure that the same diligence is applied to the open source code as is applied to internally developed or commercially licensed code provided without a viable indemnity?

4. Assignment of Responsibility for Information Gathering

This step is perhaps the most important element of identifying places where a process stalls or breaks down completely. It is very easy to create an appearance of progress by bouncing a request back and forth, but it is much harder to gather all the information necessary for the decision making described above. For instance, it is common for an original requester to complain about delays in making a decision, while the process is on hold awaiting information necessary to perform the analysis from a person tasked with obtaining the data. This problem is often compounded by a lack of feedback as to the effectiveness of information gathering—itself a sign of poor process.

Of course, the amount of information required for a decision varies significantly from organization to organization. Some companies get no more than the basic facts: project name; license; and some description about support options. Other companies want something approaching the Business Readiness Review ("BRR") process originally proposed and developed by Carnegie Mellon West, O'Reilly CodeZoo, SpikeSource, and the Intel Corporation.[2] The BRR looks at the following characteristics in assessing a project's maturity:

- Functionality: does the software meet user requirements?
- Usability: is the software intuitive, easy to install, easy to configure, and easy to maintain?
- Quality: is the software well designed, implemented, and tested?
- Security: how secure is the software?
- Performance: how does the software perform against standard benchmarks?
- Scalability: can the software cope with high-volume use?
- Architecture: is the software modular, portable, flexible, extensible, and open? Can it be integrated with other components?
- Support: how many sources of support are available?
- Documentation: is there good quality documentation?
- Adoption: has the software been adopted by the community, the market, and the industry?
- Community: is the community for the software active and lively?
- Professionalism: what level of professionalism does the development process and project organization exhibit?

Some companies do a full review of the source code before using it. Even though the project indicates that it is made available under a certain license, the file headers within the project may indicate other licenses that may or may not be compatible with the declared license for the project. Code scans can find other code that may have been copied from other projects that are made available under a license that is not indicated. Furthermore, some companies have very different review processes for code to be shipped in a product and code to be only used internally.

5. Assigning Responsibility for Follow-Through

Follow-through refers to an explicit mandate for ensuring ongoing compliance with the review process once initial approval has been granted. This is the most difficult step to implement in the entire process, as illustrated by the example of a company that had developed a very specific plan to ensure compliance with certain license obligations. The plan had been documented, and development had commenced. However, the details of the plan had not been communicated to the right people on the product management team and was ignored.

Best practices for implementing follow-through programs that ensure ongoing compliance with the process include tightly coupling review processes with product development cycles, usually in the form of automatic triggers when potential issues arise. It is important to avoid the all-too-common pitfall of follow-through plans: merely keeping a file in the lawyer's office with information about a particular component decision without ensuring that this information is actually reviewed as the code travels throughout the development process.

6. Education

To be successful, a process should place a heavy emphasis on ongoing education.

- Education at the time the policy is rolled out. The tone of the communication is

2 *See* Home – Open BRR, http://www.openbrr.org/ (last visited Jan. 22, 2010).

important. If it sounds as though the lawyers are "crying wolf," the process will do more harm than good. Generally, the best communication will portray the controls being imposed as positive steps enabling more efficient use of open source, where consistent with the company's goals. It is equally important to select the right people to deliver the communication. If a lawyer is sent to a department meeting to talk about the risks associated with using open source code, it is unlikely that anyone will long remember what was said. An enthusiastic and supportive department manager explaining the policy and process will find a much more receptive audience. And a dose of reality may help: the best education session I have attended was conducted by a manager who was new to the company and had just survived a remediation effort at his prior employer, which he described as the most tedious and frustrating experience of his working life. This personal account was a helpful illustration of what could happen if the introduction of code into the product was not actively managed.

- Education of all new employees. Some employees, especially those fresh from school, can sometimes develop bad habits that are very hard to break. For example, one company hired a brilliant programmer only to find out six months later that he had contributed massive amounts of code to the company's code base—amounts that were not humanly possible to write in his short tenure at the company. The employee had dutifully gone through all of the new employee training and had signed a statement that he understood the policies. Then how could he have thought that cutting and pasting massive amounts of code from Internet sites was appropriate? Simply, it was the way he had learned to code. The idea that he should start every project with a fresh sheet of paper seemed so preposterous to him that he had assumed that the company did not mean it. What is more, new employees may bring with them incorrect assumptions about open source code from their previous employers, or even from what they read on the Internet.
- Ongoing training regarding process improvements. Hearing and responding to criticism and implementing suggestions is key to a successful program. It is important to let the employees know that someone is listening and responsible for the success of the process.

7. Open Source Review Boards

To capture the benefits of horizontal processes described above, many organizations establish a cross-discipline group of individuals, often called the Open Source Review Board, who meet and decide as a group on all open source usage. Usually, this group assumes responsibility for all aspects of the decision because all of the stakeholders in the decision-making process are represented on the board.

Many companies rotate executives, who are in the best position to share knowledge across the entire organization, on the open source review board to spread their support of and confidence in the process across the company. At the same time, many organizations prefer to keep one or more individuals on the Open Source Review Board for several years to ensure continuity of learning. It is often these board members who are best able to change established precedent, because they remember the basis for the original decision and know when it is no longer applicable or appropriate to the facts of the situation.

Depending on organizational preferences, the role of an Open Source Review Board can be limited to confirmation of decisions already vetted through a vertical process. Alternatively, the company can adopt a largely horizontal process, with the board acting as a court of first impression. Either choice can be efficient, depending on the commitment of the members of the board to engage in the process and to make attendance at the meetings a high priority. If the board members do not make the board a high priority, then the role of the review board should be limited to confirming the results of a prior vertical process.

In my experience, Open Source Review Boards work well in larger organizations but not as well in small companies. In smaller companies the people on the board wear too many hats and expect others to attend the meeting when they are stretched thin. The in-house counsel frequently relies on the committee meeting for verification that the issues have been fully vetted and the counsel's work is delayed when the board meeting has to be rescheduled because the required decision makers are not in attendance. And precisely because a process already exists – to convene the committee – there is no alternative process to support thoughtful analysis within the management chain and outside of the review board.

8. Open Source Compliance Officers

Because the establishment of an Open Source Software Compliance Officer ("OSSCO") has been a component of many well-publicized settlements of litigation based on allegations of non-compliance with open source licenses,[3] many companies are proactively considering the establishment of a position with a similar title.

Beyond the title, there is a lot of variation in what the job description or mission statement for an OSSCO looks like. In a large organization, the job is probably more akin to an open source ombudsman who maintains some degree of a separation between the day-to-day business processes for open source approval, and is available to discuss concerns from individual employees concerned about the company's fulfillment of commitments to the communities from which the company benefits. A single compliance officer could not personally be involved in every decision, so the focus is at the process level and on specific issues that arise out of the ordinary course of business.

In smaller organizations, the description of the OSSCO's duties is closer to that of a one-person Open Source Review Board. The officer is involved in every decision that the company makes regarding development, use or distribution of open source software. To the extent the officer serves as a champion of open source software within the company, his job description should ensure that his recommendations are attuned to legitimate business needs of the company.
Here is a proposed job description that was provided by Karen Sandler of the Software Freedom Law Center:

> The OSSCO should be available and responsive regarding issues relating to free software license compliance. The OSSCO should undertake best efforts to resolve all such issues as quickly as possible. In cases where violations have been identified, the OSSCO should on a periodic basis provide to the copyright holders a written report of the scope and manner in which the company is redistributing the software and complying with the applicable licenses. The OSSCO should also be responsible for reviewing all of the company's products before they are offered to the public to ensure that they are in compliance with all applicable free software licenses.[4]

9. Timing and Tools

Most processes that are perceived by developers as successful have some element of guaranteed turn-around time. This does not mean that there is a promise that all issues will be resolved in a given time period, as that only guarantees a negative answer if the assessment has not been

3 *See, e.g.*, BusyBox Developers Agree To End GPL Lawsuit Against Verizon, Mar. 17, 2008, http://www.softwarefreedom.org/news/2008/mar/17/busybox-verizon/ (discussing the terms of the settlement between Verizon Communications Inc. and the Software Freedom Law Center acting on behalf of two BusyBox developers).

4 Private correspondence between Karen Sandler and Karen F. Copenhaver. For another excellent discussion of OSSCO, see also Posting of Stormy Peters to https://fossbazaar.org/?q=content/job-description-open-source-compliance-officer (Mar. 12, 2008) (offering thoughts on an OSSCO job description).

completed by the deadline. But a promise that there will be a response within a specified period provides confidence that open source issues will not hold up development. With a dedicated group sitting on an Open Source Review Board, institutional learning grows, and issues quickly begin to sort themselves into those that are relatively easy to answer and those that will take more time and work to bring to a conclusion. Timing commitments also tend to encourage automation of the process and the creation of forms and guidance documents that make for more efficient operation. Below are several thoughts on recommended best practices.

- Request forms and templates that make sure that all required information is provided with the initial submission avoid an inefficient back-and-forth information gathering process.
- Guidance documents that provide insight into a typical Open Source Review Board's analysis filter out requests that are unlikely to be approved.
- Email approval processes that make sure that the requests are circulated to the right people save time and frustration. Consider using dedicated mail accounts and distribution lists.
- Process management tools that provide status reports upon request and automatically remind decision-makers of approaching deadlines make the process more efficient.

To belabor an obvious point, a process for managing the use of open source software will benefit greatly from automation. Automated tools can:

- provide evidence of unintended open source usage (which often is the very first step in convincing management of the need for a formal process);
- deliver timely reminders or decision triggers by identifying open source components as they are added to a source tree;
- gather and organize the information necessary for decision making;
- provide a record of analysis and decision making;
- maintain a bill of materials for any code base that travels with the code or is available as part of the product checkpoint process; and
- assist in identifying all possible sources and available licenses of discovered open source code.

What Is the Right Policy and Process for a Company?

Of course, the right policy and process for a particular company will depend on many factors: the reason for implementing the process (e.g., what is the immediate issue?), the level of sophistication of the company's employees about open source software and communities, and the existence of a compliance issue that has already involved third parties in the process.

The main reason for implementing an open source review process is compliance. But the real answer is more nuanced, and requires understanding of events that triggered the introduction of a policy. Some examples follow.

- Customer demand. The number one reason for implementing a compliance program is a request from customers. For example, a software company has very little experience with open source but responds to customer demands for a version of their product that works on open source platforms, or to demands for a list of all open source code used in the product. The company intends to hire developers who are familiar with those platforms, and wants to educate both their existing and new developers on a consistent approach to controlling the introduction of open source code into their development environment.

- Reverse procurement policy. A company that had a prohibition on using open source software has decided to change that policy. To address the issue of gaining control of the process in its early stages, the company's policy addressed the fact that the company's employees had little or no experience with open source.
- Push the work off the lawyer's plate. The company's lawyer is getting requests for approval of licenses, and the requests arrive without any of the information required to make a decision. Frustration on both sides necessitates the creation of a process for using the attorney's time wisely and obtaining a commitment from the attorney for a target turn-around-time. Because this situation demands difficult compromises from both sides, it is important to employ well-respected internal champions who can move the process forward.
- The painful moment. A company suffers a rude awakening when it discovers a compliance failure, perhaps in the form of notices from third parties or even undesirable media attention.[5] The resulting remediation efforts are disruptive, and the company executives want to make sure that they are never in this position again. While the nature of the policies that arise in these circumstances is usually relatively ponderous, nonetheless the policies tend to work well *as designed* because they have executive backing at the highest level and are implemented quickly.
- Anticipating a merger or acquisition. A company anticipates an acquisition as its exit strategy and it wants to be prepared for due diligence inquiries from the acquiror. Here, the anticipated acquiror's counsel serves as the "bad guy" to whom all internal frustration can be transferred. Under these circumstances, developing an open source policy and process can be complemented by reviewing the company's code base for existing open source usage.
- Closing condition for round of financing. After a round of due diligence in relation to a financing, developers are instructed by a company's board to "get the code clean and to keep it clean." This incentive coming from the highest levels is likely to assure success of the process *regardless* of the nature of the policy adopted.
- Consistency across groups. A company has good procedures for most of its business operations but little or no procedures for certain part of its business (e.g., a recently-acquired small company whose core product contains significant open source code). While developing a global approach that works for all groups can be difficult, the existing disparity between divisions creates a feeling that the failure to adopt similar policies in the non-compliant segment of the business is intentional or willful. Moreover, this failure in the non-compliant portion destroys the value of the significant investment that has been made to bring the rest of the company's operations to compliance.
- Open sourcing own code. A company decides that it wants to contribute a code base to an open source project and to try to form a community around that code base. This investment in the release of the code and in the development of the community around it will be severely undermined if the company fails to comply with its open source obligations to other communities.

Conclusion

Companies should establish their core values with respect to open source usage, and formalize their analysis in a nuanced policy that responds to current issues and anticipates future challenges. Once a policy is in place, it should be operationalized by a well designed and articulated formal process. Regardless of the motivation for implementing the compliance program, an efficient

5 *See, e.g.*, GPL Violations homepage – The gpl-violations.org project, http://gpl-violations.org/ (last visited Jan. 22, 2010) (listing past and present infringers of the GPL).

process with clearly identified responsibilities is important to gain the necessary support of all of the stakeholders within the organization.

About the author

Karen F. Copenhaver *is listed in The International Who's Who of Internet & e-Commerce Lawyers, Chambers USA, Best Lawyers in America and as a Massachusetts Super Lawyer. She was also named in The Legal 500 for technology transactions. Ms. Copenhaver is only the 5th lawyer ever to receive Mass High Tech's prestigious "Mass High Tech All-Stars Award," which honors the thought leaders and innovators throughout the New England technology sector. She has been chosen by Intellectual Asset Management magazine as one of the world's top IP strategists in their feature "IAM 250 — A Guide to the World's Leading IP Strategists." Ms. Copenhaver is also director of intellectual property strategy for the Linux Foundation.*