Copyright in Open Source Software – Understanding the Boundaries

Omar Johnny,[a] Marc Miller,[b] Mark Webbink[c]

(a) Student, New York Law School; (b) Student, New York Law School; (c) Visiting Professor of Law, New York Law School.

DOI: 10.5033/ifosslr.v2i1.30

Abstract
Copyright ownership tends not to be an issue in closed-source, software development. In that model an individual or business owns — or in-licenses — the copyright in all of the code used in the software application, licenses it to end-users under a binary-only license, and relies on a combination of copyright and trade secret law to enforce contractual rights in the code. By contrast, when software is developed in an open source model, copyright issues abound, and many of these copyright issues are not well understood by software developers. This lack of understanding can undermine the intent of the developers and can potentially lead to unattractive outcomes. As early as the launch of conceptual design in open source software, issues can arise as to ownership of the work and its progeny. When a wide range of hands can touch the open source code, ownership and rights in the code can become blurred. Moreover, not all code contributions to an open source project will be protected by copyright. This paper seeks to explore the application of U.S. copyright law to software, and particularly software that is developed and licensed under an open source model. We address the boundaries of copyright protection and ownership, the importance of intent, timing and creative expression in determining these boundaries, and provide guidance to those looking to launch open source projects.

Keywords
Law; copyright; software

Info
This item is part of the **Articles** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement. This article has been independently peer-reviewed.

Software Development and Copyright

Intent, timing, and creative expression are the three themes that are central to the analysis that will follow. Each is fundamental to the application of U.S. copyright law to the development and ownership of software. Intent refers to the intention of multiple parties as they choose to collaborate and share code. Is it their intent to create a single work or multiple works? Is it their intent to permit reuse of code or place restrictions on such use? Timing addresses the time at which they manifest their intention. Is it before the work commences? Is it after one party has already written some code? Is it after all of the code has been developed? Finally, creative expression goes to the question of whether the code is actually eligible for copyright expression.

Before delving into the issues of intent, timing and creative expression, it will be helpful to understand and appreciate a wide range of development scenarios that can arise in open source development. Every day new open source projects arise, and every day the developer or developers in such projects must make decisions as to who will own the copyright in the code, who will decide how it is to be licensed, what licensing scheme will be used, and, rarely addressed, the limits of those copyright claims and license rights. Following are some of the more typical development scenarios that can be observed in open source software development. In these scenarios, a reference to a sole developer can mean either an individual or a single business entity, for example, a corporation.

Software Development Scenarios

The Sole Developer

This is probably the easiest scenario to understand since it consists of a sole developer who independently works on developing software. Assuming the developer writes all of his/her own code, this individual is the owner of the original work and is still capable of producing or authorizing a derivative work of that original. As we will see, even where the sole developer utilizes pre-existing code the developer will hold rights in that original expression contributed by the sole developer to modify the pre-existing work and to the ultimate product consisting of the combination of pre-existing works and her separate contributions. The rights she elects to share with others in her copyright in this code arise at the sole discretion of this individual developer. Intent and timing play no role as there is only one party involved. Only creative expression remains a factor in determining to what the sole developer has an interest. Where the sole developer is an individual this scenario is most frequently seen at the module level rather than in full blown applications. Examples where the developers were business entities can possibly be seen in the original development work of entities like MySQL[1] and JBoss[2] where employees of the entity did all of the original development work.

1 "MySQL was originally founded and developed in Sweden by two Swedes and a Finn: David Axmark, Allan Larsson and Michael "Monty" Widenius, who had worked together since the 1980's." http://www.mysql.com/about/ Last visited March 29, 2010.

2 "Marc Fleury started the JBoss project in 1999 in order to advance his middleware research interests." http://en.wikipedia.org/wiki/JBoss_%28company%29 Last visited March 29, 2010.

Sole Initial Developer with Others Contributing Code After Initial Development

In this scenario an original developer develops the original code and, as in the first scenario, holds all rights in the code, including the right to determine the licensing of the code. However, after the code is released to the public, the original developer invites others to make contributions to the code. In this instance the intent of the original developer in opening the development process is frequently manifested in the open source license that original developer has used to make the code available. But what if the original developer wants to retain a unified copyright in the entire code base, including the code developed by others? This is the situation faced later by companies such as MySQL and JBoss. It can also be seen in the Netscape/Mozilla development where Netscape developed its browser as a traditional, proprietary software application but later opened the development process to others and permitted them to contribute code – Mozilla.[3]

One or More Developers Agreeing to Develop Jointly in Advance of Development Work

This scenario is most descriptive of the jointly planned open source project. Here the independent developers see a need and come together to develop a strategy around the need. Perhaps they have developed a standard and need to develop a reference implementation of that standard. In any case, prior to the commencement of the work the parties agree that they intend to create a single work to which they will all contribute, and it is their intent that all contributing parties will have a right to exploit the work. It is also possible that they will form a new entity to be the home for the work. In either of these approaches, the intent and timing of the various parties at the time the work commences becomes important to determining the interests in the code, with creative expression remaining a limiting factor in determining rights in the code.

Joint Developers Who Invite Others to Join After Initial Development Work

In this scenario multiple independent developers agree to jointly develop the code as in the scenario above. However, in this instance they invite a new developer to the project after initial development work has been complete. Both the initial developer(s) and the subsequent joining developer intend to create a single, unified work. While the intent is the same as in the previous scenario, the timing of that intent differs. Creative expression remains a limiting factor in determining rights in the code.

One or More Developers Agreeing to Develop Jointly After Initial Development by Each

Multiple independent developers work independently on distinct software modules. After the initial development work is done by each developer, they see the benefit of combining their individual works into a single software application. All parties agree that they intend to create a single, unified work, but that intent is not expressed at the outset of the individual works, only after the works have been created. Thus, timing of the intent becomes a factor along with the

3 January 1998 was also the month that Netscape started the open source Mozilla project. Netscape publicly released the source code of Netscape Communicator 4.0 in the hopes that it would become a popular open source project. It placed this code under the Netscape Public License, which was similar to the GNU General Public License but allowed Netscape to continue to publish proprietary work containing the publicly released code. http://en.wikipedia.org/wiki/Netscape Last visited March 29, 2010.

creative expression in determining rights in the code.

Other Development Activities

While the scenarios above are intended to describe a variety of discrete projects, activities can occur within those development scenarios that further impact issues of intent, timing and creative expression. Among those are:

- Sequential versus simultaneous – or parallel – development
- Borrowing code from other projects
- Partial rewrites – bug fixes
- Complete rewrites incorporating concepts from earlier code

The scenarios and activities described above may not describe every possible permutation of open source software development activity, but they should be sufficient to impart the importance of intent, timing and creative expression in developing open source software. Before turning to why the factors of intent, timing and creative expression are so important in this context we first need to have a clear understanding of ownership interests in copyright.

Ownership of Copyrights in Software

Section 102 of the U.S. Copyright Act – the "Act" – provides copyright protection for "original works of authorship fixed in any tangible medium of expression."[4] Nothing more is required. Thus, whatever ownership interests exist in the copyright commences with the reduction of the expression to some tangible medium, whether paper or an electronic file.

Sole Developer

Section 201(a) of the Act provides that copyright protection in a work "vests initially in the author or authors of the work."[5] In the case of a sole developer, copyright in the software code will vest with her as the sole owner upon fixation of the code in a tangible medium of expression. That is, once she types and saves the code, fixing it in the computer's memory, she is now the proud owner of a brand new copyright in that code that will last the duration of her life, plus an additional 70 years after her death.[6] Nothing more is required of the developer to own the copyright.

Works for Hire

In the case of a developer writing software code within the scope of her employment, the resulting work is known as a "work made for hire." With a work made for hire, "the employer or other person for whom the work was prepared is considered the author for purposes of [the Act]," and

4 17 U.S.C. § 102(a).
5 17 U.S.C. § 201(a).
6 17 U.S.C. § 302(a).

the owner of the copyright in the work.[7] The developer and her employer may agree otherwise, where ownership remains with the developer, through an express agreement in a written instrument signed by both parties.[8] The intention of the developer and her employer, expressed before the code is written, will dictate who owns the copyright in the resulting work.

The result changes where the software developer is not an employee. Software is not a form of copyrightable material that statutorily falls under the list of special order or commissioned works within the scope of works made for hire unless it constitutes a commissioned contribution to a collective work.[9] Thus, the transfer of ownership of software produced outside of the scope of employment must be supported by express contractual language in writing.[10]

Joint Ownership

Section 201(a) of the Act further provides that "[t]he authors of a joint work are co-owners of copyright in the work."[11] Joint authorship in a work arises when "a work is prepared by two or more authors *with the intention that their contribution be merged into inseparable and interdependent parts of a unitary whole [emphasis added]*."[12] The hallmark of joint authorship is the authors' "joint laboring in furtherance of a preconcerted common design."[13] That is, each contributor must intend for their contributions to be merged, however, the contributors are not necessarily required to "work in physical propinquity, or in concert, nor that the respective contributions made by each joint author must be equal in quantity or quality."[14] Furthermore, it is not necessary that the contributors expressly agree, in writing, to create a joint work.[15]

The touchstone of joint authorship is the intention of the joint authors that their contributions be merged at, or before, the moment in time when the contribution of each joint author is created.[16] That is, two developers, who intend that the code they contribute to a project be merged into inseparable and interdependent parts of a unitary whole must express that intent before development commences, and each will own an equal and "undivided ownership in the entire work."[17]

So long as the intent to create a joint work exists prior to the commencement of work, it is even possible for joint authorship to occur "even though the joint authors do not work together in their common design, do not make their respective contributions during the same period, and indeed even if they are complete strangers to each other."[18] This situation is common among developers working independently in developing software modules that are to be included in a unified open-source project. The intent and timing of these developers will dictate who owns the copyright in

7 17 U.S.C. § 201(b).
8 *Id.*
9 17 U.S.C. § 101.
10 17 U.S.C. § 204.
11 17 U.S.C. § 201(a).
12 *See,* 17 U.S.C. § 101.
13 1-6 Melville B. Nimmer, Nimmer on Copyright § 6.03.
14 *Id.*
15 *See, Id.*
16 1-6 Nimmer §§ 6.02, 6.03.
17 *Id.*
18 1-6 Nimmer § 6.03.

the resulting work.

The joint authors are the co-owners of a single copyright in the joint work.[19] However, joint authorship is not the only means by which joint ownership of a work may arise.

> *A joint work will result under any one of the following circumstances: (1) if the work is a product of joint authorship; (2) if the author or copyright proprietor transfers such copyright to more than one person; (3) if the author or copyright proprietor transfers an undivided interest in such copyright to one or more persons, reserving to himself an undivided interest; (4) if upon the death of the author or copyright proprietor, such copyright passes by will or intestacy to more than one person; (5) if the renewal rights under the Copyright Act or the terminated rights under the termination of transfers provisions, vest in a class consisting of more than one person; (6) if the work is subject to state community property laws.*[20]

Each co-owner of a joint work "obtains an undivided ownership in the whole of the joint work, including any portion thereof."[21] In other words, each co-owner may use or license the work, without the consent of other co-owners, in any way she may wish.[22] Co-owners of a copyright do, however, owe to each other a duty to account for any income derived from their use or license of the work.[23]

The Derivative Work

The Act also protects derivative works and compilations.[24] A derivative work is a work based in whole, or in substantial part, upon a pre-existing work, and recasts, transforms, or adapts the underlying work in some way.[25] For copyright protection to extend to a derivative work, "the additional matter injected in a prior work, or the manner of rearranging or otherwise transforming a prior work, must constitute more than a minimal contribution."[26] Since most new works are influenced, in some way, by a pre-existing work, there exists a fine line between a derivative work and an entirely new work.[27] If a developer uses very little of an pre-existing work, taking only code not protected by copyright – like a basic function, or if she uses the pre-existing code in such a way that the resulting program is substantially different from the original, the new creation is

19 17 U.S.C. § 201(a).
20 1-6 NIMMER § 6.01.
21 1-6 NIMMER § 6.06[A].
22 *Id.*
23 *Id.*
24 17 U.S.C. § 103(a).
25 17 U.S.C. § 101
26 1-3 NIMMER § 3.03[A] (citing Feist Publications, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 348, 111 S. Ct. 1282, 113 L. Ed. 2d 358 (1991); Siegel v. Time Warner Inc., 496 F. Supp. 2d 1111, 1151 (C.D. Cal. 2007); Sherry Mfg. Co. v. Towel King of Fla., Inc., 753 F.2d 1565 (11th Cir. 1985); Montgomery v. Noga, 168 F.3d 1282, 1290 n.12 (11th Cir. 1999); Moore Pub., Inc. v. Big Sky Mktg., Inc., 756 F. Supp. 1371, 1374, 1378 (D. Idaho 1990)).
27 Lothar Determan, *Dangerous Liasons – Software Combinations as Derivative Works? Distribution, Installation, and Execution of Linked Programs Under Copyright Law, Commercial Licenses, and the GPL*, 21 BERKELEY TECH. L. J. 1421, 1430 (2006).

simply a new, original work of authorship and not a derivative work.[28]

Ownership of a separate copyright in a derivative work not only requires more than a minimal contribution to the prior work, but also permission from the owner or owners of the copyright in the prior work. Even with such permission, the creator of the derivative work will own the copyright in only that portion of the derivative work he contributed and not in any portion of the pre-existing work upon which the derivative work is based.[29]

Compilations and Collective Works

Finally, the Act protects interests in compilations, including collective works.[30] A compilation is "a work formed by the collection and assembling of preexisting materials or of data that are selected, coordinated, or arranged in such a way that the resulting work as a whole constitutes an original work of authorship."[31] The preexisting materials or data incorporated in a compilation may or may not, on their own, be capable of being protected by copyright.[32] For example, a program created by stringing together a set of basic functions – which in themselves are not protected by copyright – may receive copyright protection in the selection, coordination, and arrangement of such basic functions.

Those compilations that do, however, incorporate preexisting material or data capable of receiving copyright protection are known as collective works.[33] The Act defines a collective work as "a work, such as a periodical issue, anthology, or encyclopedia, in which a number of contributions, constituting separate and independent works in themselves, are assembled into a collective whole."[34] For example, a program that includes preexisting modules – which in themselves protected by copyright – may receive copyright protection in the selection, coordination, and arrangement of the modules.

As with derivative works, for copyright protection to extend to a compilation or collective work, "the additional matter injected in a prior work, or the manner of rearranging or otherwise transforming a prior work, must constitute more than a minimal contribution."[35] In the case of a collective work, the copyright in the prior work and copyright in the collective work as a whole are separate and distinct; the author of the prior work retains copyright ownership in her work, while ownership of the collective work, including contributions made by the author of the collective work, vests in the author of the collective work.[36]

Joint Works vs. Derivative Works vs. Compilations

28 *Id.*
29 1-6 Nimmer, § 6.05.
30 17 U.S.C. § 103(a).
31 *Id.*
32 1-3 Nimmer § 3.02.
33 *Id.*
34 17 U.S.C. § 101.
35 1-3 Nimmer § 3.03[A] (citing Feist Publications, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 348, 111 S. Ct. 1282, 113 L. Ed. 2d 358 (1991); Siegel v. Time Warner Inc., 496 F. Supp. 2d 1111, 1151 (C.D. Cal. 2007); Sherry Mfg. Co. v. Towel King of Fla., Inc., 753 F.2d 1565 (11th Cir. 1985); Montgomery v. Noga, 168 F.3d 1282, 1290 n.12 (11th Cir. 1999); Moore Pub., Inc. v. Big Sky Mktg., Inc., 756 F. Supp. 1371, 1374, 1378 (D. Idaho 1990)).
36 17 U.S.C. § 201(c).

Joint works of authorship share similarities with derivative works and compilations and, but for the intention of the authors, could be seen as one and the same.[37] Depending on the intent at the time of creation, one author's recasting, transforming, or adapting of another author's preexisting work may create either a derivative work or a joint work consisting of inseparable parts.[38] Similarly, depending on the intent at the time of creation the assembling of the works of several different authors into a collective whole may create either a compilation or a joint work consisting of interdependent parts.[39] Which case applies in each instance "lies in the intent of each contributing author at the time his contribution is written."[40] If, at the time of creation, the author intends his contribution and those contributions of others "be merged into inseparable or interdependent parts of a unitary whole" then such a merger creates a joint work.[41] On the other hand, if the intention to merge occurs only after creation of the work, then such a merger results in a derivative work or a compilation.[42]

Software Development Scenarios – The Ownership Issues

The Sole Developer

In the case of a sole developer, copyright ownership in the her work will vest with her as the sole owner. Creative expression here is vital; if the sole developer works independently, writing all of her own code, the resulting program is an original work of authorship all her own. However, if the sole developer bases her program on pre-existing code, as most software developers will inevitably do, then she toes a fine line between original works of authorship and derivative works. If the sole developer uses the pre-existing code protected by copyright[43] in such a way that the program she develops is substantially different from the original program, she will own the copyright in the original work of authorship. However, copyright ownership may vest in the sole developer as the owner of a derivative work if her work is based on a prior protected work in whole, or in substantial part, and she recasts, transforms, or adapts the underlying work in some way that exhibits more than a minimal contribution. It is important to remember that in the case of a derivative work the sole developer must have received permission from the upstream owner of the prior protected work and that ownership of a copyright in the resulting derivative work has no effect upon the copyright ownership of the prior protected work.

Similarly, copyright ownership may vest with the sole developer using a software development kit, with pre-packaged modules, to create her program. If she merely selects, coordinates, organizes, and arranges the pre-existing modules in a particular fashion, the resulting work will receive "thin" copyright protection as a compilation; conversely, if she adapts, transforms, or recasts the pre-existing modules in some way that exhibits more than a minimal contribution, the resulting work will receive copyright protection as a derivative work.

37 1-6 NIMMER § 6.05.
38 *Id.*
39 *Id.*
40 *Id.*
41 *Id.*
42 *Id.*
43 We refer here to works protected by copyright or "protected works" to emphasise the fact that not all software code will be protected by copyright. Our discussion of code not protected by copyright follows later in the paper.

Sole Initial Developer with Others Contributing Code After Initial Development

In this scenario, where the sole developer releases her program to the public and invites others to make contributions to the code, the sole developer retains ownership in the original code. The sole developer has the exclusive right to license the use of her work to other contributors. "Under the conventional doctrine of derivative and collective works," the sole developer retains her copyright in the original code, while the downstream contributors may claim exclusive ownership over the copyright in the derivative work, subject to the license to use the underlying work.[44] In this case, the sole developer, "does not obtain any property right in the derivative (or collective) work, and likewise the downstream contributor does not obtain any property right in the underlying work."[45]

The sole developer may modify the conventional doctrine of derivative and collective works and retain a unified copyright in the entire code base, including the code developed by others, by requiring modifications and enhancements to be assigned back to her as the owner of the copyright in the main code base. This is the approach taken by the GNU projects run by the Free Software Foundation.[46] This will ensure that the entire open source project may be protected as a whole by a single copyright owner, instead of having several copyrights in different modules, owned by different developers.[47]

One or More Developers Agreeing to Develop Jointly in Advance of Development Work

Where one or more developers agree to jointly develop an open source project, the copyright issues may be fairly straightforward. When independent developers see a need and come together to develop a strategy around that that need, their actions give rise to a joint work of authorship. This scenario falls squarely within the Act's definition of a joint work – a work "prepared by two or more authors with the intention that their contributions be merged into inseparable or interdependent parts of a unitary whole."[48] Independent developers jointly labouring in furtherance of a pre-concerted common design is the hallmark of joint authorship.[49] The developers are not required to work in physical proximity to each other – in fact, they may be complete strangers to one another – nor are the developers required to contribute equal portions in quantity or quality, nor are the developers required to develop their contributions simultaneously. All that is required for joint authorship is that the developers intend that their contribution be merged before, or at the moment in time when the code is developed.

Furthermore, the developers need not memorialise their intention to merge their contributions in an

44 1-6 NIMMER § 6.06[B]. Remember, if the downstream contributor is an unauthorised user of the underlying code (meaning the terms of the license were broken), that downstream contributor receives no copyright protection for the resulting work and is liable to the sole developer for copyright infringement.

45 *Id.*

46 Eben Moglen, *Why FSF Gets Copyright Assignment From Contributors*," http://www.gnu.org/licenses/why-assign.html Last visited March 30, 2010.

47 Sun Microsystems requires, through the open source license, that developers contributing to the OpenOffice.org project assign their copyright back to Sun and that all contributions to the source code are required to be made automatically available under the same open source license. *See*, OPENOFFICE.ORG, *FAQs – Licensing*, http://www.openoffice.org/FAQs/faq-licensing.html#sca1 Last visited March 30, 2010.

48 17 U.S.C. § 101.

49 1-6 NIMMER § 6.03.

express agreement.[50] However, it would be wise of the developers to draw up an agreement, perhaps in the form of a license, and clearly lay out the terms of the development plan for all to see. Remember that each developer, as a joint owner, has an undivided interest in the entire project and has the right use or exploit the work without consent of the other joint owners, subject only to the duty to account.[51] When memorialising their agreement in writing, the joint developers may chose to form a new entity for the purposes of holding the unified copyright, or they may impose certain restrictions and limitations on the use of the work by future downstream contributors through one of the several open source licenses.

Joint Developers Who Invite Others to Join After Initial Development Work

This scenario is much like the previous scenario, where independent developers agree to jointly develop an open source project. However, the sole differentiating fact is that the initial group of developers subsequently invites new developers to work on the project after the initial development work has been completed. Regardless of that factual distinction, the copyright ramifications are no different – so long as both the initial developer(s) and the subsequent joining developers intend "that their contributions be merged into inseparable or interdependent parts of a unitary whole," they will be considered co-owners of a joint work of authorship.[52]

Despite the fact that the timing of the developers' manifestation of their intent differs, their intent to have their contributions merged into a single unitary whole carries the day. Courts have held, "that the design of collaboration between joint authors need be pre-concerted only in the sense that at the time each author makes his contribution he intends that it shall be an integrated part of a greater work with supplementary contributions to be made by one or more other authors."[53] In other words, the initial developers' intent to jointly develop an open source project that would include future contributions by other developers is sufficient to create a joint work of authorship so long as the subsequent developers also manifest their intent to contribute to a unified. work. "The fact that the identity of such other authors has not been determined at the time of the original creation does not, according to these cases, derogate from their status as joint authors.'[54] Because of the potential for ambiguity in a subsequent contribution constituting a part of a joint work or a derivative work, capturing the express intent of the subsequent author at the time of authorship and contribution can be important.

One or More Developers Agreeing to Develop Jointly After Initial Development by Each

In this scenario, one or more independent developers work independently on distinct software modules. Each software module is an original work of authorship and copyright ownership is vested with each individual developer as the sole owner.[55] After the initial development by each developer, they collectively see the benefit of combining their individual works into a single software application. Because the developers' intent and the timing of that intent are different than

50 *Id.*
51 *Id.*
52 *Id.*
53 *Id.* (citing Words & Data, Inc. v. GTE Communications Servs., Inc., 765 F. Supp. 570, 575 (W.D. Mo. 1991).
54 *Id.*
55 17 U.S.C. § 201(a).

in the previous two scenarios, the copyright ramifications, as well, are distinctly different.

Had the independent developers initially, or primarily, intended for their contributions to be "merged into inseparable or interdependent parts of a unitary whole," they would be joint authors of the resulting software application, just as in the above scenarios. However, it is important to remember that each developer solely owns the copyright in her module as an original work of authorship. They are neither necessarily inseparable nor independent. And because the developers manifested their intent to merge the individual modules into a single application well after the modules were developed, the resulting software application is a considered a collective work for copyright purposes.[56] The intent of each contributing developer at the time her contribution is written distinguishes a collective work from a joint work based upon interdependent parts.[57] "If [her] work is written 'with the intention that [her] contribution ... be merged into inseparable or interdependent parts of a unitary whole' then the merger of [her] contribution with that of others creates a joint work. If such intention occurs only after the work has been written, then the merger results in a … collective work."[58]

While the individual developers retain sole ownership in the copyright of their respective individual modules, as simultaneous contributors to the collective work with the intent of creating a single unified collection work, they may each own an undivided interest in the copyright on the software application as a collective work.

Other Development Activities

The above-mentioned scenarios are intended to illustrate the issues over copyright ownership that arise in typical development scenarios. Activities can occur within those development scenarios may further impact issues of intent, timing, and creative expression. Some examples of these activities are:

- Sequential versus Simultaneous – or parallel – development.
- A developer borrowing code from other projects to incorporate into her project, assuming the borrowed code is more than a *de minimis* amount of copyright protected code, would place that developer in the position of creating a derivative work. To receive copyright protection for her work in that case, the developer would need permission to use the underlying work and would need to recast, transform, or adapt the borrowed code in some way.
- A developer working on a partial rewrite of existing code, such as a bug fix, is not likely to receive copyright protection for her contribution. If the modifications are slight in nature, the developer's contributions will fail to meet the minimum threshold of creative expression for copyright protection as an original work of authorship. Similarly, the developer's contributions will fail to meet the threshold for a derivative work since the rewrite does not recast, transform, or adapt the underlying work in a way that constitutes more than a minimal contribution or trivial

56 1-6 NIMMER § 6.05.
57 *Id.*
58 *Id.*

variation.

- Conversely, a developer working on a complete rewrite that incorporates concepts from the original work will likely receive copyright protection as either an original work of authorship or a derivative work. If the developer uses the original work in such a way that the rewrite she develops is substantially different from the original program, she may own the copyright in the original work of authorship. However, copyright ownership of the rewrite may vest in the developer as the owner of a derivative work if her rewrite is based on a prior work in whole, or in substantial part, and she recasts, transforms, or adapts the underlying work in some way that exhibits more than a minimal contribution.

Software Code Not Subject To Copyright

Open source software developers may assume every line of code they write is protected by copyright such that when they apply an open source license to that code, they are under the impression that the license will govern all use of the code. However, under U.S. copyright law, not every line of code receives copyright protection. It is important to understand why that is the case, and how it impacts enforcement rights. Assuming *arguendo* that a violation of an open source license constitutes copyright infringement, one need understand how U.S. courts will approach the issue of copyright infringement in software code.

Abstraction, Filtration, Comparison Test

When presented with copyright infringement in software code, different U.S. Federal Circuit Courts apply different tests. The most broadly adopted of these tests is the abstraction, filtration and comparison test – the "AFC test" – as first adopted by the Second Circuit. A few Circuits have adopted narrower versions of the AFC test, a handful have expressly rejected the AFC test in favor of a still narrower standard, and some have yet to adopt any definition of derivative works in software.[59]

Given the dominance of the AFC test we will focus our attention to its application. AFC is a test for substantial similarity of a computer program structure. Under this approach a court first breaks down the allegedly infringed program into its structural parts.[60] This process of breaking down the alleged infringing program into various levels or layers is the abstraction part of the test. It is a means of separating the program by its various levels – *abstractions* – to determine where there could be possible infringement. The abstractions test progresses in order of "increasing generality" from object code, to source code, to parameter lists, to services required, to general outline.[61]

"[Next, the court examines] each of these parts for such things as incorporated ideas, expression that is necessarily incidental to those ideas, and elements that are taken from the public domain;"[62]

59 Dan Ravicher, *Software Derivative Work: A Circuit Dependent Determination,* 1, (Nov. 2002).
60 *See, Computer Assoc.*, 982 F.2d at 706.
61 *See*, *Computer Associates Intern., Inc. v. Altai, Inc.*, 775 F. Supp. 544, 560 (E.D.N.Y. 1991).
62 *Id.*

in other words, the court is seeking to identify those components within a particular level that are not protected by copyright. “[The] court would then be able to sift out all non-protectable material.”[63] This process is the filtration part of the test. It is a process for identifying and removing code that is not protected under the Act from that material which is subject to copyright protection. Examples of elements or code not subject to copyright protection are functional elements, merger of expression and idea, scènes à faire in software, facts in software, and code in the public domain.

“Left with a kernel, or possibly kernels, of creative expression after following this process of elimination, the court's last step is to compare the protectable material with the corresponding sections of an allegedly infringing program.”[64] This is known as the comparison part of the AFC test. “The result of this comparison will determine whether the protectable elements of the programs at issue are substantially similar so as to warrant a finding of infringement.”[65]

Consider the following example of the AFC test for infringement. The holders of the copyright in Busybox claim the program Tools[66] has copied some portion of the Busybox code and is in violation of the license governing Busybox - the GNU General Public License version 2. The first level of abstraction, or layer of review, would be the entire source code for Busybox.[67] A secondary layer would be the various files that are contained at the next level of that Busybox source, e.g., applets, arch, archival AUTHORS, Config.in, e2fsprogs, README, etc. A tertiary layer would be the actual source files, like executables, header files, or text files that contain program instructions or information. The final layer would be the actual lines of code or text. Where the court starts its analysis will largely be determined by the extent of alleged infringement, and each layer of abstraction is reviewed for both literal and non-literal copying

In our hypothetical the court determines that the sole basis for the infringement claim lies within a file in Busybox named e2fsprogs. Saying that there is an infringement of this file within Busybox does not necessarily mean that the alleged infringing party infringed the entire file. If the entire file is alleged to infringe, the court will assess the infringement at two levels, the structure and sequence of the entire e2fsprogs file, and then any sublayers. To consider the sublayers the court abstracts the e2fsprogs file into its various sub layers. In e2fsprogs the court will consider each of the component files, such as the e2fs_lib.h file, a header file. At this level there are no longer any subfiles to abstract. Within e2s_lib.h – at the file level – the court would examine whether the actual lines of code or text within the header file constitute material protected by copyright. This is the filtration test. We next consider the various considerations that may be applied by a court in determining protectable versus non-protectable material.

Literal vs. Nonliteral Copying

With respect to such things as musical, dramatic, and motion picture works, and works of “literature,” as contrasted with "literary" works in the broader statutory sense, to the extent that

63 *Id.*
64 *Id.*
65 *Id.*
66 “Tools” is a fictitious program and should not be construed to be any real program.
67 The source for Busybox may be found at http://git.busybox.net/busybox/snapshot/busybox-1_15_3.tar.bz2 Last viewed on March 1, 2010.

such a work contains original, literal manifestations, the work is protected by copyright."[68] However, a work in one form for may infringe the same work expressed in a different form even if it does copy word for word. For example, a motion picture may infringe a book by using "the story's unique setting, characters, plot, and sequence of events."[69] This is nonliteral copying. "This type of copying of nonliteral expression, if sufficiently extensive, has never been upheld as permissible copying; rather, it has always been viewed as copying of expressive elements of creative originality."[70]

In *Lotus* the court recognized the amorphous nature of "nonliteral" elements of computer programs. [71] "Unlike the written code of a program or a flowchart that can be printed on paper, nonliteral elements – including such elements as the overall organization of a program, the structure of a program's command system, and the presentation of information on the screen – may be less tangibly represented."[72] "In the context of computer programs, nonliteral elements have often been referred to as the "look and feel" of a program."[73]

The *Lotus* court's conclusion is consistent with the treatment of the user interface and some other nonliteral aspects of computer programs, which are not merely articles "having an intrinsic utilitarian function."[74] When computer programs include both literal and nonliteral elements, which can be identified separately from and are capable of existing independently of the utilitarian aspects of the program, they are potentially copyrightable.[75]

Because the court must determine the scope of copyright protection that extends to a computer program's nonliteral structure,[76] the *Computer Associates* court held that comparison, the third and final step of the abstraction, filtration, comparison test for substantial similarity, is appropriate for nonliteral program components.[77]

Nimmer warns of the pitfalls in use of a "look and feel" type of test.

> *More broadly, the touchstone of "total concept and feel" threatens to subvert the very essence of copyright, namely the protection of original expression. "Concepts" are statutorily ineligible for copyright protection; for courts to advert to a work's "total concept" as the essence of its protectible character seems ill-advised in the extreme. Further, the addition of "feel" to the judicial inquiry, being a wholly amorphous referent, merely invites an abdication of analysis. In addition, "total concept and feel" should not be viewed as a sine qua non for infringement--similarity that is otherwise actionable cannot be rendered defensible simply because of a different*

68 *See*, *Lotus Dev. Corp. v. Paperback Software Int'l*, 740 F. Supp. 37, 51 (D. Mass. 1990).
69 *Id.* at 52, quoting Stewart v. Abend, 495 U.S. 207, 109 L. Ed. 2d 184, 110 S. Ct. 1750, 1759, 14 U.S.P.Q.2D (BNA) 1614 (1990).
70 *Id.* at 52.
71 *Id.* at 46.
72 *Id.*
73 *Id* at 62.
74 17 U.S.C. § 101 (defining "useful article").
75 *Lotus*, 740 F. Supp. at 54.
76 *See*, *Computer Assocs. Int'l v. Altai*, 982 F.2d 693, 703 (2d Cir. N.Y. 1992).
77 See, *Id* at 710.

> *"concept and feel." In sum, therefore, the frequent invocations of this standard do little to bring order to the inquiry into what constitutes substantial similarity, and would be better abandoned.*[78]

However, the Ninth Circuit ultimately defended the standard against Nimmer's critique:

> *Some commentators have worried that the "total concept and feel" standard may "invite an abdication of analysis," because "feel" can seem a "wholly amorphous referent." . . . But the [Ninth Circuit's] caselaw is not so incautious. Where [the court] has described possible infringement in terms of whether two designs have or do not have a substantially similar "total concept and feel," the court generally has taken care to identify precisely the particular aesthetic decisions--original to the plaintiff and copied by the defendant--that might be thought to make the designs similar in the aggregate.*[79]

Functionality Exception to Copyright Protection

When developing computer programs it is inevitable that some of the code will be functional in nature. As stated earlier, the Act awards copyright protection to creative expression. "Functional elements and elements taken from the public domain do not qualify for copyright protection."[80] Therefore, there is no striking similarity even between two identical works so as to warrant an inference of copying to the extent that, albeit copyrightable, functional considerations can account for the identity.[81]

What makes an element "functional?" Elements are functional if they are necessary to the program and do not exhibit any creativity. Aspects of a program's structure which are dictated by the nature of other programs with which they were designed to interact are functional in nature and, thus, not protected by copyright.[82]

Functional elements may also be dictated by the nature of the program being developed. In *Computer Associates*, "the district court found that the overlap exhibited between the list of services required for both ADAPTER and OSCAR 3.5 was determined by the demands of the operating system and of the applications program to which it was to be linked through ADAPTER or OSCAR."[83] These aspects of the program's structure are therefore functional in nature and not copyrightable.

For example, graphical user interfaces [GUI's] generated by computer programs are partly artistic and partly functional. They are a tool to facilitate communication between the user and the computer. GUIs do graphically what a character-based interface, which requires a user to type in

78 4-13 Nimmer § 13.03[A][1].

79 *Tufenkian Import/Export Ventures, Inc. v. Einstein Moomjy, Inc.*, 338 F.3d 127, 134 (2nd Cir. 2003).

80 *Computer Assoc.*, 982 F.2d at 714.

81 *See,* 4-13 Nimmer §13.02[B].

82 *See, Computer Assoc*., 982 F.2d at 715.

83 *Id.*

alphanumeric commands, does manually.[84]

In *Lotus* the court held that the Lotus menu command hierarchy is an uncopyrightable method of operation. [85]

> *The Lotus menu command hierarchy provides the means by which users control and operate Lotus 1-2-3. If users wish to copy material, for example, they use the Copy command. If users wish to print material, they use the Print command. Users must use the command terms to tell the computer what to do. Without the menu command hierarchy, users would not be able to access and control, or indeed make use of, Lotus 1-2-3's functional capabilities.*[86]

The menu command hierarchy in Lotus 1-2-3 is functional by nature of the program and therefore not copyrightable.[87]

Other areas to consider when determining whether an element is purely or primarily functional include:

- hardware standards;
- software standards;
- computer manufacturer design standards;
- target industry practices; and
- computer industry programming practices.[88]

Idea/Expression Merger Exception to Copyright Protection

Under the Act, in no case does copyright protection extend to any idea regardless of the form in which it is described, explained, illustrated, or embodied in such work.[89] "It is a fundamental precept of copyright that only the expression of ideas, and not the ideas themselves, are copyrightable."[90] "Merely stating the rule, however, does not make any easier the task of drawing the line between where idea ends and expression begins."[91]

> *The line between idea and expression may be drawn with reference to the end sought to be achieved by the work in question. In other words, the purpose or function of a utilitarian work would be the work's idea, and everything that is not necessary to that purpose or function would be part of the expression of the idea…Where there are various means of achieving the*

84 *Apple Computer, Inc v. Microsoft Corp.,* 35 F.3d 1435, 1445 (9th Cir. 1994).
85 *See, Lotus Dev. Corp. v. Borland Int'l*,, 49 F.3d 807, 819 (1st Cir. 1995).
86 *Id.* at 815.
87 *See, Id.* at 815.
88 4-13 Nimmer §13.03[F].
89 17 U.S.C. 102(b).
90 1-2 Nimmer §2.02.
91 4-13 Nimmer §13.03[B][2][a].

> *desired purpose, then the particular means chosen is not necessary to the purpose; hence, there is expression, not idea.*[92]

The characteristics of computer software, a utilitarian work, make the determination of idea and expression more complicated. Competitive forces that exist in the software marketplace lead to the problem that multiple programmers may design identical or highly similar works.[93]

> *Efficiency is an industry-wide goal. Since, as we have already noted, there may be only a limited number of efficient implementations for any given program task, it is quite possible that multiple programmers, working independently, will design the identical method employed in the allegedly infringed work. Of course, if this is the case, there is no copyright infringement.*[94]

The merger doctrine is as an exception to the idea-expression dichotomy which holds that, when there are so few ways of expressing an idea, not even the expression is protected by copyright.[95] When idea and expression merge such that a given idea is inseparably tied to a particular expression, rigorously protecting the expression would confer a monopoly over the idea itself, in contravention of the statutory command. To prevent such an occurrence, courts have invoked the merger doctrine.[96]

In the realm of computer programs, merger issues may arise in unusual ways. Although, there may be many ways to implement a particular idea, efficiency concerns can make one or two choices so compelling, as to virtually eliminate any other form of expression.[97]

> *Computer searching and sorting algorithms provide good examples of this phenomenon. Any computer system that deals with significant quantities of data will spend much of its operating time engaged in sorting and searching through that data. Because the amount of time spent on sorting and searching operations can significantly influence a program's operating speed, efficient methods of sorting are highly desirable. A great deal of computer science research has been devoted to developing methods of sorting or searching through data, and to analyzing the relative efficiency of various methods. As a result of such research, it is now recognized that some methods of sorting or searching are significantly more efficient than others in handling particular types of data, even though any of numerous methods will work. In such cases, the merger doctrine should be applied to deny protection to those elements of a program dictated purely by efficiency concerns.*[98]

92 *Whelan Assoc., Inc. v. Jaslow Dental Lab, Inc.*, 797 F.2d 1222, 1236 (3rd Cir. 1986).
93 *Computer Assoc.,* 982 F.2d at 708.
94 *Id.*
95 *See, BUC Int'l Corp. v. Int'l Yacht Council Ltd.*, 489 F.3d 1129, 1143 (11th Cir. 2007).
96 See, 4-13 NIMMER § 13.03[B][3].
97 See, 4-13 NIMMER § 13.03[F][2].
98 *Id.*

While the merger doctrine and the functionality exception to copyright protection are similar, there is a slight difference which distinguishes the two. "Under the merger doctrine, when an idea can be expressed in only one fashion, that expression is not protected by copyright."[99] Here the focus is on the limitations of the expression of an idea which results in the merger of that idea and its expression. In contrast, elements are functional if they are necessary to the program and do not exhibit any creativity.[100] In reference to the functionality exception, the focus is not on the limitations on expression of an idea resulting in merger of the two, but on aspects of a program's structure which are dictated by the nature of other programs with which they were designed to interact.[101]

Scènes à Faire in Software Exception to Copyright Protection

The Act does not directly define the scènes à faire doctrine. Scènes à faire refers to aspects of a work that are indispensable or standard parts of the material to be copyrighted.[102] "The [scènes à faire] doctrine is often invoked to immunize from liability similarity of incidents or plot that necessarily follows from a common theme or setting."[103] "Judge Leon Yankwich has called such incidents scènes à faire, *i.e.*, scenes which must be done."[104]

> *As was remarked above concerning merger, this doctrine does not limit the subject matter of copyright; instead, it defines the contours of infringing conduct. Labeling certain stock elements as "scènes à faire" does not imply that they are uncopyrightable; it merely states that similarities between plaintiff's and defendant's works that are limited to hackneyed elements cannot furnish the basis for finding substantial similarity.*[105]

In *Durang,* the court found that alleged similarities that follow obviously from the unprotected idea are therefore unprotected scènes à faire.[106] The *Durang* court held that the lower court properly applied the scènes à faire doctrine to hold unprotectable, forms of expression that were either stock scenes or scenes that flowed necessarily from common unprotectable ideas.[107] The *Durang* court went on to explain that common in that context means common to the works at issue, not necessarily referring commonly found in other artistic works.[108]

Further, under the doctrine of scènes à faire, elements of an original work are not protected if the "common idea is only capable of expression in more or less stereotyped form."[109] "Beyond mere plot incidents applicable to works of fiction, the scènes à faire doctrine can be invoked throughout other copyright contexts as well; from guidebooks to infomercials to Frequently Asked Questions

99 4-13 NIMMER § 13.03[F][2]
100 *See, Computer Assoc.*, 982 F.2d at 715.
101 *See, Computer Assoc.*, 982 F.2d at 715.
102 See, *Id.* at 710.
103 4-13 Nimmer §13.03[B][4].
104 *Id.*
105 *Id.*
106 *See, John William See v. Christopher Durang and LA. Stage Co.*, 711 F.2d 141, 143 (9th Cir. 1983).
107 *Id.*
108 *Id.*
109 *Mist-On Sys. v. Gilley's European Tan Spa*, 303 F. Supp. 2d 974, 978 (W.D. Wis. 2002).

web pages and beyond."[110]

In *Gilley's European Tan Spa*, "[the] plaintiff contended that defendants infringed plaintiff's exclusive rights under the Copyright Act by preparing and displaying on their web page an unauthorized Frequently Asked Questions page that mirrors the Frequently Asked Questions page found on plaintiff's web page."[111]

> *The Gilley's court held a business cannot copyright a Frequently Asked Questions page as such or copyright words or phrases commonly used to assemble any given Frequently Asked Questions page. The format of a Frequently Asked Questions page is a common idea in our society; the elements of a Frequently Asked Questions page (a list of questions beginning with common words) are stereotypical. Some additional similarity beyond generic formatting is necessary to establish infringement."*[112]

Applied to computer programs, the merger and scènes à faire doctrines suggest that if a limited number of options exist to achieve a given function efficiently, interoperate with another application, or run in a given environment, copyright will not permit exclusive control over those program elements.[113] Scènes à faire is distinguishable from the merger doctrine because, the merger doctrine holds that when there are so few ways of expressing an idea, not even the expression is protected by copyright.[114] The idea and expression are in essence, fused. In contrast, scènes à faire relates to alleged similarities that follow obviously from the unprotected idea.[115] The focus in scènes à faire is not on the merged idea and expression or the limited number of ways to express the idea, but on the similarities between expression in question which are a natural result of the idea being expressed.

Moreover, scènes à faire is also distinguishable from the functionality exception to copyright protection. While scènes à faire is expression that relates to stock scenes or elements which are necessary to the idea such as frequently asked questions or "readme" files, functionality relates to aspects of a program's structure which are dictated by the nature of other programs with which they were designed to interact,[116] such as hardware or software standards. As software development languages become more and more sophisticated in the ready-made tools they provide developers and as more and more developers, especially open source developers, reuse standard or stock bits of code to carry out standard functions, we will see the scènes à faire doctrine applied with greater regularity in software to deny copyright protection.

Public Domain Exception to Copyright Protection

Works eligible for copyright protection may nonetheless enter the public domain, i.e., no longer enjoy that copyright protection. For example, a work whose copyright term has expired is

110 4-13 NIMMER § 13.03[B][4].
111 *Mist-On Sys.*, 303 F. Supp. 2d at 976 (W.D. Wis. 2002).
112 *Id.* at 978.
113 *Computer Assocs.*, 982 F.2d at 709-10.
114 *BUC Int'l Corp. v. Int'l Yacht Council Ltd.*, 489 F.3d 1129, 1143 (11th Cir. 2007).
115 *See v. Durang*, 711 F.2d at 143.
116 *Computer Assoc.*, 982 F.2d at 715.

obviously not protected. Similarly, a work may have entered the public domain by reason of the failure to satisfy certain statutory formalities of the Act as it existed prior to 1978. In addition, an author may choose to lift the protections of copyright and voluntary place the work into the public domain.[117] "Moreover, copyright protection under the Act is not available for any work of the United States Government, but the United States Government is not precluded from receiving and holding copyrights transferred to it by assignment, bequest, or otherwise."[118]

What is the public domain? "A work of authorship is in the public domain if it is no longer under copyright protection, it failed to meet the requirements for copyright protection, or the holder of the copyright disclaimed copyright in the work."[119] Works in the public domain are free for anyone to use without permission from the former owners(s) of the copyright.[120] Material found in the public domain is free for the taking and cannot be appropriated by a single author even though it is included in a copyrighted work.[121]

> *An enormous amount of public domain software exists in the computer industry, perhaps to a much greater extent than is true of other fields. Nationwide computer "bulletin boards" permit users to share and distribute programs. In addition, computer programming texts may contain examples of actual code that programmers are encouraged to copy. Programmers often will build existing public domain software into their works. The courts thus must be careful to limit protection only to those elements of the program that represent the author's original work.*[122]

Copyright protection is automatic and vested in the author the moment it is created and fixed in a tangible form.[123] Voluntarily placing a copyrighted work in the public domain requires some manifest expression of the author's intent.[124] Consequently, open source developers should be cautious about assuming code to be in the public domain without some express statement from the copyright holder declaring the code to be in the public domain. An invitation to use with nothing more may be sufficient, but combined with a requirement of attribution suggests the author is merely granting permission to use while retaining the copyright and its various protections. A more definite state, such as "as the author of this work I disclaim the copyright work and declare the work to be in the public domain" would leave little doubt as to the copyright holder's intent. The Creative Commons Copyright-Only Dedication statement gives some indication of the complexity of committing a work to the public domain.[125]

Facts in Software Exception to Copyright Protection

> *Facts, whether alone or as part of a compilation, are not original and*

117 *See*, 1-2 Nimmer § 2.03[G].
118 17 U.S.C § 105
119 http://www.copyright.gov/help/faq/faq-definitions.html Last visited March 30, 2010.
120 *See, Id.*
121 *See, Computer Assocs.*, 982 F.2d at 710.
122 4-13 Nimmer § 13.03[F][4].
123 http://www.copyright.gov/help/faq/faq-general.html, last visited April 3, 2010.
124 4-13 Nimmer § 13.03[F][4].
125 http://creativecommons.org/licenses/publicdomain/ Last visited April 3, 2010.

> *therefore may not be copyrighted. A factual compilation is eligible for copyright if it features an original selection or arrangement of facts, but the copyright is limited to the particular selection or arrangement. In no event may copyright extend to the facts themselves.*[126]

"In no case does copyright protection for an original work of authorship extend to any … discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work."[127] Nimmer explains that the discoverer merely finds and records.

> *He may not claim that the facts are original with him, although there may be originality and hence, authorship in the manner of reporting, i.e., the expression, of the facts. As copyright may only be conferred upon authors, it follows that quite apart from their status as ideas, discoveries as facts per se may not be the subject of copyright.*[128]

> *[The Court in Fiest*[129] *noted] the tension between two well-established copyright propositions, … facts are not copyrightable, whereas compilations of facts generally are. As the tool for untangling those disparate strands, the Court relied on the bedrock principle of copyright subsistence--that only original works of authorship qualify for protection. Given that facts, by themselves, are never copyrightable, the Court reasoned that the element of originality that renders a factual compilation protectible must lie in selection, coordination, or arrangement of facts, with the scope of protection concomitantly limited to that original selection, coordination, or arrangement. That formulation, it should be noted, corresponds to the scope of copyright generally for derivative or collective works.*[130]

How does this relate to computer software? In *WIREdata* an owner of a copyright attempted to hide data in its copyrighted program. [131] Specifically, the copyright owner attempted to use copyright law to "block access to data that not only are neither copyrightable nor copyrighted, but were not created or obtained by the copyright owner."[132]

> *The information at issue in [WIREdata] was collected and then was slotted into plaintiff's database. Defendant did not want that database's organized structure; it only wanted the raw data. That last consideration proved decisive in defeating plaintiff's copyright infringement claim: A work that merely copies uncopyrighted material such as facts is wholly unoriginal and the making of such a work is therefore not an infringement of copyright.*[133]

126 *Feist Publ'ns, Inc. v. Rural Tel. Serv. Co*., 499 U.S. 340, 350 - 351 (1991).
127 17 U.S.C. §102(b).
128 1-2 Nimmer § 2.03[E].
129 *Feist*, 499 U.S. at 350 (1991).
130 1-3 Nimmer § 3.04[2][a].
131 *Assessment Techs. of WI, LLC v. WIREdata, Inc*., 350 F.3d 640 (7th Cir. 2003).
132 1-3 Nimmer § 3.04[B][3][a].
133 *Id.*

Within the framework of computer software development it will not be unusual to find lines of code that merely make a factual statement. A reference in a line of code to another place in the program, a table showing equivalences, a target name may all be merely factual statements within the context of the software and, thus, not eligible for copyright protection.

Avoiding Infringement

Fair use

As we have seen, the owner of copyright in software code has the exclusive right to reproduce or to authorize others to reproduce her work. However, that right is subject to certain limitations, one of which is the doctrine of fair use.[134] Originally developed by the courts through case-law, certain uses or reproductions of a work protected by copyright are considered to be fair, and thus, not an infringement of the owner's exclusive rights granted by copyright law. In other words, fair use is a defence to copyright infringement.

Section 107 of the Act contains a list of the various purposes for which the reproduction of a particular work may be considered fair, such as criticism, comment, news reporting, teaching, scholarship, and research.[135] In addition, the Act sets out four factors to be considered by a court determining whether or not a particular use is fair:

1. the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes;
2. the nature of the copyrighted work;
3. the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and
4. the effect of the use upon the potential market for or value of the copyrighted work.[136]

From a practical perspective, it is important to recognize that the fair use doctrine is malleable – the court has wide discretion in its application of the four factors to the particular facts of the case before it. There are no hard and fast rules in fair use and the difference between an infringing use and a fair use may be murky and not easy to delineate. Using a work protected by copyright without permission poses a substantial amount of risk. But for fair use, the unauthorized use of a work protected by copyright is an infringement. Unless the use falls within one of permissible statutory uses, there is no way to conclusively know whether the use is fair without costly and expensive litigation.

Should a developer choose to roll the dice and rely on fair use as a defence to copyright infringement, she can only stand to benefit from paying close attention to the language of the statute. A safe play is to use the copyrighted work for one of the permissible purposes expressed

134 17 U.S.C. § 107.
135 *Id*.
136 *Id*.

by the statute. In the relatively straightforward case of students using copyrighted code in the classroom setting, the use is fair. However, once the nature of that use changes, so does the copyright analysis. If those same students were to use the copyrighted code outside of the classroom and, for example, develop a module and post it on the Internet, such use is likely to be an infringement of the original copyright.

On the other hand, if the use of the copyrighted work falls outside of the express permissible purposes, the developer can tailor her use to navigate the fair use factors in her benefit. First, the purpose and character of the developer's use – the developer should steer clear of commercial uses (court's are not likely to allow a developer to profit from the unauthorized use of another's work) and instead try to transform the original work by adding new expression and adding value to the original work by creating something new. Second, the nature of the copyrighted work – the developer will have a strong case of fair use if the original work contains code that is not subject to copyright such as, functional or utilitarian code, scenes a faire code, or public domain code. Third, the amount and substantiality of the portion taken - the less the developer takes, the stronger her case for fair use will be. However, even if she takes only a small portion of a work, her use may not be fair, if the portion taken is the heart of the work, i.e., the few lines of code that really make the program the program. Fourth, the effect upon the potential market for the original – if the developer's use deprives the original owner of income or undermines a new or potential market for the original code, such use will severely weaken her case for fair use.

Copying Code Not Protected By Copyright

A prima facie cause of action for copyright infringement requires that the plaintiff prove that protected elements of its work have been copied.[137] In other words, if the code copied is not protected by copyright, there is no copyright infringement. Accordingly, a developer may use the unprotectable elements of a computer program, without permission from the upstream developer, and the subsequent developer will not infringe the copyright protecting that particular program. The salient question then becomes, which elements of a computer program are protected by copyright and which are not?

In the previous section, we discussed several examples of parts of programs that may not be protectable. Recall that the purpose or function – the idea – of a utilitarian work – like computer software – is not protected by copyright. Similarly, factual data is not protectible, despite the fact that the software program, as a whole, may be protected by copyright. Code in the public domain and code falling within the scènes à faire doctrine are also not protected by copyright.

A developer is free to use any elements of an existing program that are dictated by external factors such as efficiency, compatibility and interoperability requirements, computer manufacturer design standards, hardware and software specifications, widely accepted target industry practices, and widely accepted programming industry practices. A developer is also free to use any elements of an existing program that have entered the public domain. Finally, a developer is also free to use elements of an existing program that are standard, stock or common to a particular subject matter under the doctrine of scènes à faire.

137 52 Am. Jur. Proof of Facts 3d 107.

Copying De Minimis Lines of Code

De minimis non curat lex. Roughly translated to English, this legal maxim means the law does not concern itself with trifles. Applied to copyright law, this maxim means that copying which has occurred to such a trivial extent does not constitute an actionable claim of infringement. In other words, a developer who uses a small amount of code, without permission from the upstream owner, does not infringe the copyright protecting the upstream program. For example, one court held that copying thirty characters from approximately fifty pages of source code was *de minimis*, and not an infringement.[138]

However, a developer should be aware that courts do not approach the *de minimis* inquiry in a vacuum; but rather, will consider the context in which the copying took place. As the great Judge Learned Hand said, "no plagiarist can excuse the wrong by showing how much of his work he did not pirate."[139] The court will measure the quantity of the portion used, but it will also measure the quality of portion used. The analysis will occur at the module level and even if only a small amount of code has been used, a court is likely to find it an infringing use if the portion used constitutes the heart of the original work. This last point should ring familiar. The *de minimis* inquiry is part and pacel of the third prong of the fair use analysis – the amount and substantiality of the portion used in relation to the copyrighted work as a whole.

Testing For Derivation

The term *derivative work* is paramount within the open source software community. Derivative works are part and parcel of open source software projects – by its very definition, open source software participants are encouraged to modify, recast, transform, and adapt the source code and redistribute it back to the community for further modification, recasting, transformation, and adaptation.

Remember that the Act defines a derivative work as is a work based in whole, or in substantial part, upon a pre-existing work, and recasts, transforms, or adapts the underlying work in some way.[140] For copyright protection to extend to a derivative work, “the additional matter injected in a prior work, or the manner of rearranging or otherwise transforming a prior work, must constitute more than a minimal contribution.”[141] In addition requiring more than a minimal contribution, a derivative works requires permission from the owner or owners of the copyright in the underlying work. In the open source software community, this permission typically comes in the form of an open source software license. The question becomes, then, when does a developer create a derivative work? The answer, as one might imagine, is not entirely clear.

At one end of the spectrum, a new program will be held to be a derivative work when the source code of the original program was used, modified, translated or otherwise changed in any way to create the new program. A developer can avoid copyright infringement in this instance and her work can be deemed to be an authorized derivative work with one more requirement. She must

138 *Vault Corp. v. Quaid Software Ltd.*, 847 F.2d 255 (5th Cir. 1988).
139 *Sheldon v. Metro–Goldwyn Pictures Corporation*, 81 F.2d 49 (2d Cir. 1936).
140 17 U.S.C. § 101.
141 NIMMER, *supra* note 20.

have the permission from the upstream copyright owner to create that derivative work, and that means remaining within the bounds of the original program's license. Conversely, a work will not be held to be a derivative work where the developer merely applies minor, trivial variations to the source code of the original program without adding anything original of her own. The resulting work will constitute a non-literal copy of the original program, thus infringing the copyright of the original program.

At the other end of the spectrum, a work will not be held to be a derivative work where the developer uses library functions and other off-the-shelf routines contained in an original program, without ever touching the original program's source code. As we have seen above, these components of the original code are not copyrightable themselves, given their highly functional nature, and as such, downstream developers are free to use them in their subsequent works. This work is an independently created new work, not a derivative work.

Utilizing Free and Open Source Code Licensed By Others

An open source software license is "merely a mechanism by which the copyright owners place limitations on the downstream end user's ability to utilize the software code [under the Copyright Act]."[142] In closing, this article is not intended to inform developers about black letter compliance with the open source software license under which they are operating. Rather, it is intended to suggest what a developer is and is not permitted to do in the grey areas outside of the open source software license under which they are operating.

By adhering to the principles illustrated by this article, a developer may utilize free and open source software code and steer clear of the thickets of copyright infringement. The four factors of the Fair Use doctrine stands ready to provide a developer with a safe harbour for her use preexisting open source software code or if the purpose of her use is "criticism, comment, news reporting, teaching, scholarship, or research."[143] A developer can utilize the unprotected elements of preexisting open source software code in her program, without creating a derivative work of the original program. Likewise, a developer is free to utilize those elements of preexisting open source software code that are in the public domain. Although the code she develops is a derivative work, she will not infringe the copyright protecting the preexisting program because her work is one that is derived from the public domain elements. A developer can also utilize a small, *de minimis*, amount of open source software code, so long as the code she uses does not constitute the "heart" of the preexisting program. Finally, a developer can simply create a new, original program, and not a derivative, by utilizing any combination of the above unprotected elements of a preexisting program, or by changing the preexisting program so much that the new program differs substantially from the original.

About the authors

Omar Johnny *is a legal extern at Nielsen Company in the Corporate Legal Department working*

142 Brad Frazer, *Open Source is Not Public Domain: Evolving Licensing Philosophies*, 45 IDAHO L. REV. 349, 365 (2009).
143 17 U.S.C. § 107.

on patent law. Johnny is a second year law student at New York Law School. Johnny holds a B.S. in Computer Science from Hobart College.

***Marc Miller** is a third-year student at New York Law School where he is a Student Research Fellow at the Institute for Information Law & Policy. Miller is also affiliated with the Institute as a John Marshall Harlan Scholar and is a Notes & Comments Editor of the New York Law School Law Review. Miller holds a Bachelor of Science in Management Information Systems from the University of Vermont.*

***Mark Webbink** is a Visiting Professor of Law and Executive Director of the Center for Patent Innovations at New York Law School. Webbink is also a Senior Lecturing Fellow at Duke Law School and has served as an Adjunct Professor at NCCU Law School. From 2000 to 2007 Webbink served in various capacities with Red Hat, Inc., including General Counsel, Deputy General Counsel for Intellectual Property, Senior Vice President and Secretary. Webbink presently serves on the board of directors of the Software Freedom Law Center and on the advisory board of the Axial Exchange. Webbink has written and spoken extensively on the subjects of open source software, software patents, and patent reform. Webbink received his B.A. Degree from Purdue University in 1972, his Masters in Pubic Administration from the University of North Carolina – Chapel Hill in 1974, and his J.D., magna cum laude, from North Carolina Central University School of Law in 1994. Webbink maintains a website on open source and intellectual property law at www.walkingwithelephants.com.*