Strategies for Litigating Software Copyright Infringement Cases

BY STEVEN R. HANSEN, HANSEN IP LAW PLLC, AND DR. RICHARD TURLEY, COLORADO STATE UNIVERSITY

Steve Hansen is the owner of Hansen IP Law PLLC, a law firm specializing in intellectual property matters. He can be reached at 248 504 4849 and srh@hanseniplaw.com.

Dr. Turley is a Professor of Computer Information Systems at Colorado State University. He can be reached at rick.turley@business.colostate.edu.

INTRODUCTION

The courts have recognized for some time that copyright protection is available for software. However, copyright is intended to protect works of *expression*, not methods, functions, or algorithms. Except in those cases where there is verbatim copying of all or a large portion of source code, it can be difficult to determine whether similarities between programs constitute copyright infringement.

The courts have developed analytical techniques for isolating the expressive, and therefore copyright-protectable, aspects of software. However, modern computer program source code can total hundreds of thousands of lines of code, or even more. The judicial techniques for identifying what is protectable can be difficult to implement in practice. In general, they seek to identify what is ***not*** protectable and "filter" it out of the comparison between the copyrighted and accused code, which can be a difficult and expensive task.

It can also be difficult get a plaintiff copyright owner to identify the protectable aspects of registered code which were allegedly copied. Without a mechanism for defining the scope of the copyright owner's claim, seeking summary judgment can be very difficult. In this article, we look at those aspects of computer source code that may be "expressive" by examining various design choices programmers make which may serve as a type of "fingerprint" for their expressive style. By identifying these choices, we hope to provide litigants with some key expressive features that they can use to frame and litigate software copyright infringement cases.

COMPUTER PROGRAMMING BASICS

The Copyright Statute defines a computer program as "a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result."[1] Programs may comprise "object code," which is a non-textual code that is only understandable by computers or "source code," which is human-readable code. Only object code can be executed by a computer. Source code is converted to object code by compiling it. Computer programs are written to perform functions, such as receiving and storing data, retrieving data, performing mathematical calculations, providing graphical user interfaces, and displaying data. The specific sequences of steps used to perform those functions are often referred to as "algorithms."

Source code computer languages can be broken down into two general categories: "procedural programming languages," and "object oriented programming languages," or "OOP." Procedural programming is built around sequences of steps and algorithms performed on the program inputs to generate program outputs. Procedural programs tend to execute sequentially, going line by line through the program. Structurally, there is generally a main program and a number of functions or subroutines, each of which performs its own discrete functions. These functions are generally invoked by the main program in a set sequence.

In contrast, OOP builds on procedural programming and further defines "classes" which are characterized by particular attributes and which may perform *methods* (e.g., data manipulation or calculations) on the attribute data and/or to generate the attribute data. The objects created in the program are associated with "classes" which provide a template for the objects, defining their various attributes and methods. Structurally, OOP source code defines a variety of classes and "objects" that are specific instances of a corresponding class. Procedural programming languages include C, BASIC, Fortran, Cobol, and Pascal. OOP languages include Java, C++ and C#. OOP languages represent the current state of the art for lengthy and complex programming because changes can be more easily made to discrete classes or objects without affecting the remaining parts of the programming. Object oriented languages also allow for better software reuse, making programming more efficient.

THE SCOPE OF COMPUTER SOFTWARE PROTECTION

Among the statutory categories of copyright-protectable works, computer programs are considered to be "literary works."[2] However, "[i]n no case does copyright protection . . . extend to any idea, procedure, process, system, method of operation, concept, principle or discovery . . . ."[3] This prohibition is critical because software necessarily includes some number of functional features and processes that cannot be protected by copyright law.

Courts have developed certain techniques to separate the protectable from the unprotectable aspects of software. Probably the most widely used technique is the Abstraction-Filtration-Comparison test set forth by Second Circuit Court of Appeals in *Computer Associates Int'l, Inc. v. Altai*.[4] A similar test used by the Ninth Circuit Court of Appeals is the "analytic dissection" test set forth in *Apple Computer, Inc. v. Microsoft Corp., et al.*[5] While the tests are not identical, they both seek to ensure that determinations of copyright infringement are based only on those aspects of programs that constitute protectable expression. Thus, the tests seek to filter out or dissect the following:

1. Ideas (functions)
2. Features dictated by efficiency or which are necessarily incidental to the ideas underlying the program
3. Features dictated by the mechanical specifications of the computer on which the program is intended to run
4. Features necessary to provide compatibility with other programs
5. Features dictated by industry standards or common practices
6. Public domain elements (e.g., open source programs)[6]

DIFFICULTIES IN APPLYING *APPLE* AND *COMPUTER ASSOCIATES*

The *Apple* and *Computer Associates* analytical techniques are elegant and consistent with the goals of copyright law. However, they are difficult to apply in practice, especially as a defendant hoping to frame discovery or obtain summary judgment. How do you get a copyright owner to admit what is "non-protectable" or to define what is truly protectable and allegedly included in the accused work? If you cannot pin the copyright owner down, how do you move for summary judgment or frame the case for the jury? In a patent case, the parties have a set of patent claims which defines the scope of what is protected. However, in a software copyright case, the scope of the protected right is much more amorphous and difficult to pin down.

As with other cases, you can and should serve contention interrogatories. However, aggressive copyright owners may resist committing to the scope of their claim and may ask the Court to allow them to defer responding until the close of discovery. If the copyright holder responds in general, vague terms, you may have to meet and confer several times to force the issue. How do you develop a discovery plan, frame your expert reports, or otherwise develop the case when you do not even know what you're aiming at? In general, filing a summary judgment motion forces a plaintiff to commit to its positions, but how can you given the analytical framework provided by the courts?

One strategy is to set up a "straw man" by identifying the *potentially expressive* aspects of a copyrighted program and showing that they are not present in the accused program. To do this, it is instructive to look at the arbitrary or expressive design choices programmers make when writing software.

WHAT ARE THE EXPRESSIVE ASPECTS OF SOURCE CODE?

Generally speaking, the expressive elements relate to the choices made by a programmer which are not constrained by requirements of the design or the programming languages and development environments used to create the code. Here are some to look for:

Programming Languages & Development Environment

The programmer is often free to select the programming language for implementing a program. At a high level, the programmer chooses whether to use procedural programming or OOP. The programmer can then choose from among a variety of procedural and OOP languages, each having its own commands and syntax.

Once a language is selected, it inherently constrains the programmer and limits his expressive choices. Some languages place more constraints than others. To the extent that a certain language requires the use of specific key words or structures, these cannot be considered expressive elements.

The programmer may also select a "software development environment" (SDE), which is a program that provides an interface for writing and debugging code. A development environment such as Microsoft's Visual Studio® assists the programmer by automatically generating certain code, such as that used to create user interfaces. Once an SDE is selected, however, it will also constrain the expressive aspects of the programmer's coding. For example, certain SDE's automatically create certain lines of code such as declaration headers or user interface implementations (e.g., the shapes and colors of buttons, screens, etc.). Such elements are not original to the programmer and cannot be protected.

Program Structure

Programmers choose how to organize their code. For example, programmers may modularize their code to varying degrees. In the case of OOP, this means that one programmer may define a larger number of classes than another programmer, whereas a procedural programmer may make greater use of discrete subroutines. The amount of complexity and functionality provided by the individual modules may also differ. For example, "pure" OOP programmers tend to limit their classes to a single functional purpose, while others may "stuff" several functions into the same class. The former type of programmer will have more classes with fewer attributes and methods defined for each class than will the latter. The particular combinations of methods (e.g., calculation or data processing sequences) used in functionally analogous classes may also differ between different programmers. Method declarations-- including name, method return data type, and the number and type of parameters--are particularly expressive elements.

Coding Style

Programmers also choose their coding style. One common coding style issue is the use of conventions regarding the names used for variables, methods, and classes. There are several variable naming conventions, referred to as "Hungarian Notation," "Camel Case," and "Pascal Case." The conventions differ as to their use of prefixes, capitalization, the use of underscores, etc. Programmers not only choose which convention to use, but also how consistently to apply that convention. Some programmers may be rigidly consistent while others may change conventions at different points in their code. Programmers may also dispense with the standard naming conventions and use their own convention. Within a particular naming convention, the variable naming format is standardized, so the use of the standardized format is not protectable, but the choice of standardized format may be.

TOOLS FOR EXPRESSIVE ANALYSIS

Automated tools are useful for detecting verbatim copying, but are not as helpful where the alleged copying is not verbatim. There is no current industry standard set of tools for identifying the expressive elements of source code or filtering out the non-protectable elements. Some tools help automate the process, but the comparison still relies on a substantial amount of manual investigation of proposed similarities.

FRAMING DISCOVERY AND SUMMARY JUDGMENT MOTIONS

At a minimum, contention interrogatories should be used to force the copyright holder to identify what protectable expression was allegedly copied. However, another approach is to direct targeted discovery to expressive aspects of the accused and copyrighted programs. Deposition questions, interrogatories, and/or requests for admission may be directed to the following issues:

- The variable naming conventions used, and why they were used;
- The scheme used to structure classes (in the case of OOP languages) and why it was used;
- The programming language that was used and why;
- Differences between the structure and function(s) of key classes or the numbers of classes used to achieve a particular programming function;
- The method headers (method names, return types, and number and type of parameters) used to implement key features, and why each was used;
- The function and use of the SDE.

While courts rightfully seek to identify protectable expression in software, their techniques can be unwieldy for a defendant seeking to define and litigate a software copyright infringement case. We hope that the foregoing strategies will help especially where copyright holders resist efforts to define and focus their claims. IPT

ENDNOTES

1. 17 USC § 101
2. *Computer Assoc. Int'l, Inc. v. Altai, Inc.*, 982 F.2d 693, 702 (2d. Cir. 1992)
3. 17 USC § 102(b)
4. 982 F.2d 693 (2d. Cir. 1992)
5. 35 F.3d 1435 (9th Cir. 1994)
6. *Computer Assoc., Int'l*, 982 F.2d at 707-710; *Apple Computer, Inc.*, 35 F.3d at 1444-1445.

Cozen O'Connor Wins Another $15.8 Million, And More, in Patent Infringement Damages for Metso Minerals, Inc.

Cozen O'Connor, among the top 100 law firms in the United States, announced that it has successfully protected the patent rights of Metso Minerals, Inc., a global supplier of technology and services for the mining and construction industries, against Terex Corporation, a global manufacturer of heavy machinery, one of its subsidiaries and two of its dealers. The final judgment is expected to lead to a total of $50 million in damages.

In December 2010, a jury had awarded Metso Minerals, Inc. $15.8 million in damages for patent infringement nearly five years after the Metso Minerals, Inc. v. Powerscreen International Distribution Limited et al lawsuit began. The first named defendant is now known as Terex GB Limited, and the other defendants include its corporate parent, Terex Corporation, and two of their distributors, Emerald Equipment Systems, Inc. and Powerscreen New York, Inc.

On December 9, 2011, the Federal District Court for the Eastern District of New York affirmed the jury's verdict of one year earlier, which was reached after a seven-week trial. The jury verdict had held that the defendants willfully infringed Metso's U.S. Patent No. 5,577,618 directed to mobile screening and crushing machines. In the court's decision, each of the defendants' four motions to overturn the jury's verdict and/or for a new trial were dismissed. The court affirmed the jury's verdict that Metso's patent was infringed, that it was infringed willfully, that the patent was not obvious, that the patent was not unenforceable due to alleged inequitable conduct, and that Metso had not delayed commencing its lawsuit. The court also doubled the primary damages award (raising it to $31.6 million), ordered an accounting for supplemental damages for October 2007 to the present that were not included in the jury's damages award, and awarded pre- and post-judgment interest. In July 2011, the court issued an order permanently enjoining the defendants from marketing their 11 infringing mobile screeners. It is estimated that after all of the accounting is completed, the final amount of the judgment in this case could reach $50 million.

Metso is represented in this litigation by Cozen O'Connor members Michael C. Stuart and Lisa A. Ferrari, with Marilyn Neiman assisting during the trial.

"We are delighted that we were successful in protecting our client's patented design which had become the standard in the industry ten years ago and has been copied by a number of other competitors," said Mr. Stuart. "The enhanced damages award underscores the clear willful infringement of this valid, enforceable and important patent."

Established in 1970 and ranked among the 100 largest law firms in the United States, Cozen O'Connor has 575 attorneys who help clients manage risk and make better business decisions. The firm counsels clients on their most sophisticated legal matters in all areas of the law, including litigation, corporate, and regulatory law. Representing a broad array of leading global corporations and middle market companies, Cozen O'Connor serves its clients' needs through 22 offices across two continents.