Learned Hand Never Played Nintendo: A Better Way to Think about the Non-Literal, Non-Visual Software Copyright Cases

David W.T. Daniels†

The rapid development of computer science has left courts struggling to fit new technologies into the traditional framework of intellectual property law. One of the most difficult legal issues is defining the limits of copyright protection for computer software. For about a decade, courts have held that the literal code of a computer program enjoys copyright protection in the same way that a literary work does.[1] But this conclusion does not settle whether and how the *non*-literal structure of a computer program should be protected. Under long-standing case law, one can infringe on the copyright of a play or novel not only by copying the literal words, but also by duplicating the work's plot or structure.[2] Courts have likewise concluded that copying the non-literal structure of a computer program counts as a copyright infringement.[3]

---

† A.B. 1989, Princeton University; M.A. (A.B.D.) 1991, The University of California (Berkeley); J.D. Candidate 1994, The University of Chicago.

[1] *Apple Computer, Inc. v Franklin Computer Corp.*, 714 F2d 1240, 1249 (3d Cir 1983). See also *Williams Electronics, Inc. v Artic International, Inc.*, 685 F2d 870, 875 (3d Cir 1982).

[2] See *Nichols v Universal Pictures Corp.*, 45 F2d 119, 121 (2d Cir 1930).

[3] See, for example, *Whelan Associates, Inc. v Jaslow Dental Laboratory, Inc.*, 797 F2d 1222, 1245-46 (3d Cir 1986); *Computer Associates International, Inc. v Altai, Inc.*, 982 F2d 693, 701 (2d Cir 1992). These cases deal with the non-visual structure of the program. Related but distinct cases involve the visual interface of the computer software. In the visual interface cases, courts have considered the possibility of copyrighting the overall "look and feel" of a computer program and investigated the extent to which one can copy specific visual elements that serve as an interface with the user. Both sets of cases often refer to the same case law and read as though they were applying the same law to both situations. See, for example, *Apple Computer, Inc. v Microsoft Corp.*, 799 F Supp 1006, 1041-42 (N D Cal 1992); *Lotus Development Corp. v Paperback Software Int'l*, 740 F Supp 37, 62-68 (D Mass 1990); *Williams Electronics*, 685 F2d at 873-74. This Comment assumes these two categories pose distinct legal problems, each of which deserves independent treatment notwithstanding the overlapping case law. In particular, visual interfaces present distinct practical problems. For example, as more people use programs with a particular visual interface, they derive increased utility from software with a similar "look and feel." This is the phenomenon of "network externalities." See Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 Stan L Rev 1045, 1066-71 (1989).

Non-literal computer copyright cases have been a headache for the courts because the issues involved tend to push the boundary of copyright law into areas generally associated with patent law. Copyright law traditionally protects the literal word and, by extension, literal computer code. Patent law protects inventions and processes and, by extension, computer "ideas." Courts, however, have gradually expanded copyright protection to guard against those who would copy non-literal elements of a computer program.[4] This expansion has created a set of nominal "copyright" cases which straddle the conceptual border between patent and copyright rules. By protecting non-literal expression through copyright, courts in effect protect ideas—but this muddles the border where copyright ends and patent begins.

Like most intellectual property conundrums, this problem involves the need to preserve incentives for innovation without stifling subsequent improvement. Classically, copyright law has protected only the literal word. Under the theory that literary innovation can proceed apace even if past authors have strong property rights in their texts, the grant of copyright protection is readily given and long lasting. The law is stingier in granting monopolies through patent rights. Because technical innovation often builds on the earlier work of others, the federal government awards patent protection only if an invention clears certain hurdles, and even then only for a relatively short period of time.

When courts extend copyright protection to non-literal expression, they risk granting copyright protection in instances that only warrant whatever protection patent law may afford. A computer program is both a series of written instructions that a microprocessor executes *and* the embodiment of a process. Processes generally fall under patent rules, and as such, computer programs look to be unlikely candidates for even literal copyright protection. A computer program's non-literal features seem to be even more unlikely candidates for copyright protection. Even so, almost without exception, courts hold that both literal and non-literal aspects of computer programs deserve copyright protection.

This Comment argues that courts facing computer copyright issues have seen one difficulty where in fact there are two. Courts have traditionally evaluated the problem of non-literal copying, both for literary works and, recently, for computer programs, in terms of a distinction between idea and expression;

[4] See *Altai*, 982 F2d 693; *Whelan*, 797 F2d 1222.

only expressions receive copyright protection. But this distinction is not especially helpful in thinking about non-literal software copyrights. Instead, we must carefully distinguish two issues that courts often conflate. First, we need a framework for comparing the structures of two similar programs so that courts can decide what counts as "copying." Second, we need rules that articulate a clearer boundary between those non-literal elements of a computer program that should be subject to patent rules and those elements that merit the more lenient protection of copyright. This Comment sets out an approach designed to meet both of these needs. It argues that merger doctrine can help articulate a clearer boundary between the proper spheres of copyright and patent, while a selection test that is attentive to unfair competition through copying will help frame the comparison of similar programs.

I. COMPUTER SOFTWARE: THE ELUSIVENESS OF "STRUCTURE"

This section provides a brief tour through the world of software design and operation[5] in order to answer a central yet elusive question: what exactly is the non-literal, non-visual structure of a computer program? Given the great diversity of computer software, the courts have understandably adopted a case-by-case approach to defining "structure," assigning that label to a broad range of software elements. But even though courts have refrained from attempting a single, universally applicable definition of a program's "structure," one can make some generalizations about the nature of computer programming in order to understand the particular issues to which the case law responds.

Computer programs are sequences of instructions designed to manipulate a computer in a particular way. When a programmer sets out to write a program, he generally begins by identifying as clearly as possible the task he wants to automate. He then divides that overarching task into a hierarchy of simpler subtasks, and then those subtasks into smaller ones. The overall purpose is broken down into discrete, manageable pieces called modules or subroutines. Between these modules flow data, and again, the programmer maps out this flow in advance.[6]

---

[5] For a comprehensive account, see Anthony L. Clapes, et al, *Silicon Epics and Binary Bards: Determining the Proper Scope of Copyright Protection for Computer Programs*, 34 UCLA L Rev 1493 (1987).

[6] See *Altai*, 982 F2d at 697-98.

Isolating each of the individual steps in these modules can be very laborious and time consuming.[7] Depending on his programming style, one programmer might map out the relationship between steps with flow charts. Another might sketch the function of his work by using pseudocode—an English-like notation which is not itself a programming language, but that mimics the instructional nature of such languages.[8] Regardless of the method used, the programmer usually has considerable discretion in deciding how to articulate the smaller tasks that carry out a larger function. The sound exercise of this discretion forms the essence of programming skill.

Once the programmer sketches the program's outline, he can encode the result in the computer language of his choice. He has a wide range to choose from. So-called high-level languages, such as COBOL, FORTRAN, PASCAL, C, or C++, produce an end result known as source code. The source code is a sequence of instructions that are intelligible to a human, but not to the computer itself. In order for the computer to act on those instructions, a program must "compile" the source code into a series of 0's and 1's (binary form) known as the object code.[9] The resulting program, in its source code form, can be a list of instructions thousands or even millions of lines long. That list will have its own internal coherence and its own set of landmarks. In operation, it behaves and interacts with its user in its own peculiar way.

Courts have included both internal structure and outward behavior as elements of the program's "structure."[10] The Third

---

[7] See, for example, *Whelan*, 797 F2d at 1231 ("Ms. Whelan spent a tremendous amount of time studying Jaslow Labs, organizing the modules and subroutines for the Dentalab program, and working out the data arrangements, and a comparatively small amount of time actually coding the Dentalab program.").

[8] Jeff Duntemann, *Parts Isn't Parts*, Dr. Dobb's Journal 146-47 (Oct 1992) ("There's no standard definition for pseudocode . . . . s with flowcharts, pseudocode can be implemented in any structured language. It's a much shorter trip to real code than from flowcharts, since all the control-flow structures are there in the pseudocode in English-like form.").

[9] High-level programming languages are the commercial standard, but alternatives do exist. Some languages, notably BASIC, require no object code. The source code is fed directly into the computer via an interpreter. Still other languages require no interpreter and can be used by the computer as is (Assembly, for example). The original programming method required little or no code at all. The keepers of the very first computer, the now-ancient ENIAC, programmed that machine by connecting cables in different patterns. Newer technology allows a programmer to develop software without actually writing all the code himself. These code generators overcome some of the more onerous obstacles to programming, increasing the speed of writing software. See Andrew Johnson-Laird, *Reverse Engineering of Software: Separating Legal Mythology from Actual Technology*, 5 Software L J 331, 337 (1992).

[10] See Arthur Miller, *Copyright Protection for Computer Programs, Databases, and*

Circuit used the phrase "structure, sequence and organization" as a shorthand way of referring to this combination.[11] The breadth of that particular locution has been criticized by some,[12] though perhaps unfairly.[13] In any case, what the user sees is conceptually distinct from the arrangement of the program's guts. To keep the distinction between visual and non-visual structure clear, this Comment will use "non-literal, non-visual structure" to designate the organization of static code and "visual structure" to refer to the program's operation from the user's perspective.[14]

Non-literal structure comes in a variety of forms. Program design presents an easily understandable conception of what can count as non-literal, non-visual structure: the hierarchical division of tasks into subtasks, the interaction of one module with another, and the way that data passes from one part of the program to the other. But "structure" has other senses as well. One court found that the type, length, and names of data fields in a program were protected as software structure.[15] Another court extended protection to the sequence of signals that operated as an electronic key to unlock a particular piece of hardware.[16] In short, it is hard to specify in advance what elements of a computer program count as non-literal structure. The inquiry is inevitably fact-driven, but law does guide the inquiry in one way: courts are especially concerned with any copying of non-literal structure that makes the plagiarist's job easier.[17]

Computer Generated Works: Is Anything New Since CONTU?, 106 Harv L Rev 955, 977 (1993) (CONTU was a Congressional commission whose 1980 report influenced several changes to the copyright statute to accommodate computer software.).

[11] *Whelan*, 797 F2d at 1224 n 1, 1248.

[12] See Randall Davis, *The Nature of Software and its Consequences for Establishing and Evaluating Similarity*, 5 Software L J 299, 316 (1992); *Sega Enterprises, Ltd. v Accolade, Inc.*, 977 F2d 1510, 1524-25 (9th Cir 1993).

[13] See Miller, 106 Harv L Rev at 997-98 (suggesting that *Whelan* was true to the common law at the time, but that courts have since properly refined their approach).

[14] This latter category includes the whole range of visual contrivances generated by the software to allow the user to receive and enter information into his computer. Litigants have fought over the copyright of program menus, *Lotus Development Corp. v Borland International, Inc.*, 799 F Supp 203 (D Mass 1992), *Lotus Development Corp. v Paperback Software International*, 740 F Supp 37, 62-68 (D Mass 1990); over the use of graphical icons and windows as a means of representing information, *Apple Computer, Inc. v Microsoft Corp.*, 799 F Supp 1006 (N D Cal 1992); and over the sequence of screens and graphs, *Autoskill, Inc. v National Educational Support Systems*, 793 F Supp 1557, 1560 (D NM 1992).

[15] *CMAX/Cleveland, Inc. v UCR, Inc.*, 804 F Supp 337, 355 (M D Ga 1992).

[16] *Atari Games Corp. v Nintendo of America Inc.*, 975 F2d 832, 840 (Fed Cir 1992).

[17] See, for example, *Altai*, 982 F2d at 701, quoting *Nichols v Universal Pictures Corp.*, 45 F2d 119, 121 (2d Cir 1930) ("It is of course essential to any protection of literary prop-

This concern helps identify where a court should focus its attention, but raises in turn an attendant risk. Because software is inherently functional, aggressive protection of non-literal structure may blur the distinction between what properly belongs to the realm of patent and what belongs to copyright. The next Section shows why this is an especially acute problem for computer programs.

II. DOCTRINAL DEVELOPMENT

A. A Copyright Primer and the Muddle of Idea, Process, and Expression

Copyright law, naturally enough, protects against copying (though not against independent invention as would a patent). In comparison with patent law, the right granted is generous, generally lasting for the life of the author plus fifty years.[18] Patent law, meanwhile, offers protection only for seventeen years from the date of registration.[19] Moreover, the law grants a copyright liberally. While a patentable invention must be useful, nonobvious, and novel,[20] copyrighted material need only be original—a threshold so low that even a phone book can qualify.[21] Since copying is often difficult to prove, the law creates a rebuttable presumption of copying if the plaintiff proves that the copier had access to the material and the copy is substantially similar to the original.[22]

erty . . . that the right cannot be limited to the text, else a plagiarist would escape by immaterial variations.").

[18] 17 USC § 302(a) (1988).

[19] 35 USC § 154 (1988).

[20] 35 USC §§ 100-04 (1988).

[21] See *Feist Publications v Rural Telephone Service Co.*, 499 US 340, 361 (1991).

[22] Paul Goldstein, 2 *Copyright* § 7.2.1 at 8 (Little, Brown, 1989). There is disagreement over how to view the connection between substantial similarity and copyrightability. Some courts first ask whether the alleged copy is substantially similar to the original and then ask whether the similarity constitutes a copyright infringement. *Whelan*, 797 F2d at 1232. Others distinguish between "probative" similarity and substantial similarity. "Probative" similarity is a factual inquiry into whether copying took place; "substantial" similarity embodies the legal inquiry into whether the copying violated a right held by the plaintiff. See Melville B. Nimmer and David Nimmer, *Nimmer on Copyright* § 13.01[B] at 13-8 to 13-15 (Matthew Bender, 1993) ("*Nimmer*"). These distinctions have led to confusion in both case law and commentary. Some refer to substantial similarity as a legal question, see Comment, *A Square Peg in a Round Hole: The Proper Substantial Similarity Test for Nonliteral Aspects of Computer Programs*, 68 Wash L Rev 351 (1993), while others, such as the *Whelan* court, refer to substantial similarity as a factual inquiry. 797 F2d at 1245-46. This Comment follows the *Whelan* approach.

The protection of copyright law extends beyond the literal word to protect non-literal elements as well.[23] How far this protection should extend is a perennial problem for copyright law. The longstanding distinction has been that copyright, at common law and under modern statutes, only protects expression, not ideas or processes.[24] Processes and—in the sense of inventions—ideas are properly subjects of patent protection. For instance, copyright law protects a book about accounting (expression), but that protection does not extend to the accounting method itself (process).[25] (The latter would be, if anything, subject to patent rules.) Likewise, copyright protects the plot of *Romeo and Juliet* (expression), but not the basic story line of boy meets girl (idea).[26]

Distinguishing ideas from expression is often a somewhat slippery enterprise. In the context of literary works, Learned Hand put the problem most clearly and attempted to solve it with his celebrated abstraction test:

> [U]pon any work . . . a great number of patterns of increasing generality will fit equally well, as more and more of the incident is left out. The last may perhaps be no more than the most general statement of what the [work] is about, and at times might consist only of its title; but there is a point in this series of abstractions where they are no longer protected, since otherwise the [author] could prevent the use of his "ideas," to which, apart from their expression, his property is never extended.[27]

"Idea/expression distinction" is an unfortunate way of putting the real nature of the legal test. Whenever courts grant protection to a story line or even to a translation, they are already protecting, in an important sense, the idea of the work rather than the literal text. In effect, Hand's abstraction test is not a way of distinguishing an idea from an expression, but rather a way of distinguishing protectable ideas from unprotectable ideas based on a

---

[23] *Nichols*, 45 F2d 119 (plot of a play protected); *Peter Pan Fabrics, Inc. v Martin Weiner Corp.*, 274 F2d 487 (2d Cir 1960) (general design of a piece of fabric protected); *Roth Greeting Cards v United Card Co.*, 429 F2d 1106 (9th Cir 1970) ("total concept and feel" of a greeting card protected).

[24] See 17 USC § 102(b) (1988); *Baker v Selden*, 101 US 99, 102 (1879).

[25] See *Baker*, 101 US at 104-05.

[26] See 3 *Nimmer* § 13.03[A][1][b] at 13-34 to 13-39 (cited in note 22).

[27] *Nichols*, 45 F2d at 121.

criterion of generality.[28] As such, Hand's solution is more a method of analysis than a hard and fast test. Fixing the right level of abstraction must necessarily be ad hoc. Hand himself despaired of finding a better method.[29] Courts have naturally turned to Hand's abstraction test as the starting point for determining how far copyright protection extends to computer programs. Unfortunately, the nature of computer programs makes a straightforward application of the idea/expression doctrine—a doctrine developed for literary works—impossible.

The central insight of the computer revolution was the discovery that mathematical functions and other recursive processes generally can be expressed through a series of 0's and 1's. Unfortunately, that insight baffles copyright law. To give computer programs any protection at all, one must extend copyright rules to cover processes in at least one sense; the literal code of a program, after all, is an algorithm that embodies a process. If one copyrights the program, one in effect copyrights a process.

However, not all processes embodied in code look like proper candidates for copyright protection. For example, a programmer may desire to write a bit of code to add together a hundred numbers. The code that he writes embodies an algorithm in two ways. First, the program executes the mathematical algorithm of addition. In a second sense, the program is itself a sequence of instructions that allow the computer to carry out the mathematical process. Certainly, copyright should not extend so far as to protect a mathematical algorithm; yet generously expanding nonliteral protection of computer code risks protecting the algorithm that the programmer sets out to reduce to code. That outcome may stifle subsequent innovation. If the protection is drawn too narrowly, however, then one may fail to accord copyright protection to the creative and original expression that the code embodies.

This difficulty resembles the problem that Learned Hand grappled with in the context of literary works, but only by rough analogy. Again courts must attempt to fix some level of generality at which copyright protection ends. However, for literature,

[28] See generally 3 *Nimmer* § 13.03[A][1][a] at 13-31 (cited in note 22).

[29] "Obviously, no principle can be stated as to when an imitator has gone beyond the 'idea,' and borrowed its 'expression.'" *Peter Pan Fabrics*, 274 F2d at 489. Others have proposed rules, such as the Chaffee pattern test and the "total concept and feel" test, to tackle essentially the same problem. 3 *Nimmer* § 13.03[A][1][b]-[A][1][c] at 13-31 to 13-41. But these rules generally suffer from the same deficiencies as the Hand analysis: neither gives a judge a hard-edged rule to apply.

fixing the right level of generality does two things simultaneously. It allows a court to pick out the relevant abstract features—plot, characters, and so on—to prevent non-literal plagiarists, yet it also permits the court to pick a low enough level of generality such that the copyright need not discourage subsequent authors from writing a somewhat similar work. Computer software, as we shall see, is different in that it is harder to envision the tradeoff between protecting against copying and the costs of restricting such copying as lying on a single continuum. Instead of having a single rule, such as the abstraction test, to balance the need for protection and the desire to encourage further invention, this Comment argues that it would be better to have a different rule which attends to each concern separately. Courts have yet to discern this distinction clearly, and are thereby missing an opportunity to provide a more coherent and manageable system for software protection.

B. The Leading Computer Copyright Cases: *Whelan* and *Altai*

The Third Circuit staked out the strongest copyright protection position for structural similarity in *Whelan Associates, Inc. v Jaslow Dental Laboratory, Inc.*[30] Jaslow, a dental lab owner, decided that his business would be more efficient if computer automated. Finding that the programming requirements were beyond his expertise, Jaslow hired Whelan to write the appropriate software. Whelan duly wrote a program in a language known as EDL (Event Driven Language), which worked only on relatively expensive computers. Jaslow hoped to sell this software to other dental laboratories, but in order to do so, it had to be usable on less expensive machines. Toward this end, Jaslow rewrote the program in BASIC, rendering it useful on a wide range of much cheaper computers. After Jaslow terminated all of his agreements with Whelan, Whelan sued for infringement of the structure of the EDL program.[31]

Because the two programs were written in different computer languages, they shared no identical code. Moreover, Jaslow's copy was not a straightforward translation of EDL into BASIC. Otherwise, however, the two programs had very similar structures. The court found that the two programs shared the follow-

[30] 797 F2d 1222 (3d Cir 1986).
[31] Id at 1225-26.

ing elements: five subroutines which functioned identically, the file structure, and some screen outputs.[32]

The Third Circuit used general common law principles to decide whether such similarity amounted to copyright infringement. The court began with standard copyright law, noting that infringement is presumed if the plaintiff can show access and substantial similarity.[33] Access was not a question in *Whelan*, nor, the court found, was substantial similarity. However, the court still had to decide whether the offending program, by copying the structure of Whelan's code, infringed on protectable expression. Conceding the difficulty of this task, the court fixed on one standard:

> [T]he 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 desired purpose, then the particular means chosen is not necessary to the purpose; hence, there is expression, not idea.[34]

The court preferred to state the purpose at a general level: the program's idea was "the efficient management of a dental laboratory."[35] Since that idea could be implemented in several ways, the court found that all of the dental lab program's structure counted as part of the program's expression.[36]

At the other end of the spectrum of copyright protection stands the Second Circuit's decision in *Computer Associates International v Altai*.[37] Computer Associates marketed a scheduling program called CA-Scheduler, which scheduled and controlled various tasks performed on a particular family of IBM computers. CA-Scheduler had one component designated "Adapter" which allowed the program to run on either of three different operating systems. After this program was developed, one of Computer Associate's employees, Claude Arney, went to work for Altai, a

[32] Id at 1242-46.

[33] Id at 1231-32.

[34] Id at 1236 (emphasis omitted).

[35] Id at 1236 n 28.

[36] Id. The program's expression thus included "the manner in which the program operates, controls and regulates the computer in receiving, assembling, calculating, retaining, correlating, and producing useful information either on a screen, print-out or by audio communication." Id at 1239, quoting the district court's opinion in the same case, 609 F Supp 1307, 1320 (E D Pa 1985).

[37] 982 F2d 693 (2d Cir 1992).

competitor. Altai had its own computer scheduler called ZEKE, but that program could only run on one IBM operating system. Unbeknownst to either of his employers, Arney had secreted away some of the source code for the Adapter component. Using this code, Arney set out to build a new adapter component for Altai—so that ZEKE, like CA-Scheduler, could run on several operating system platforms. Altai's new, more versatile scheduling program was renamed OSCAR.[38]

Altai first learned about the copying when the summons for the lawsuit arrived. Arney was summarily fired. Altai immediately began to rework the copied portions of OSCAR, using engineers who knew nothing of the copied code. Computer Associates persisted in its suit, claiming that even the new copy bore a substantial similarity to the structure of its Adapter program. But while the trial court found some structural similarities between the two programs, it held that these similarities resulted from shared functions and operating systems and therefore refused to find a copyright violation.[39]

The Second Circuit affirmed. While holding that non-literal program structure could obtain copyright protection, the court rejected the *Whelan* approach for defining the extent of that protection.[40] Instead, the court suggested a three-part procedure by which to judge infringement: abstraction of the various layers of program structure, filtration of unprotectable elements, and comparison of the protected elements with the infringing product.[41]

The abstraction process takes its inspiration from Learned Hand. The *Altai* court instructed trial judges to analyze the program in question by dissecting its structure starting at the lowest level of abstraction, then working up to higher and higher levels of description. As the court put it, "a court should dissect the allegedly copied program's structure and isolate each level of abstraction contained within it. This process begins with the code and ends with an articulation of the program's ultimate function."[42]

The *Altai* court then introduced a second step: filtration. Once the program has been separated into conceptual layers, the judges should discard the unprotectable elements before judging

[38] *Computer Associates, Inc. v Altai, Inc.*, 775 F Supp 544, 549-54 (E D NY 1991).
[39] Id at 561-62.
[40] *Altai*, 982 F2d at 702, 705-06.
[41] Id at 706.
[42] Id at 707.

infringement. For instance, a judge should filter away any pieces of the program that are already in the public domain,[43] elements dictated by external factors,[44] and elements mandated by efficiency.[45]

Only after abstraction and filtration does *Altai* require a comparison of the two works. By that point, the finder of fact should have isolated the "golden nugget" of protectable expression—one which, if found in the copy, would amount to an infringement.[46] The court believed that its new method of analysis narrowed the scope of copyright protection, and that such steps were necessary to preserve the integrity of "certain fundamental tenets of copyright doctrine."[47]

III. The Square Peg and the Round Hole

Courts have felt ill at ease as they have sought to apply traditional copyright doctrine to the area of computer software. The *Altai* court, for example, labelled the exercise an effort to fit "the proverbial square peg in a round hole."[48] In part, this should be no surprise: it would be odd indeed if copyright doctrines designed for literary works could be applied wholesale to computer code. There is an important dissimilarity between liter-

[43] Id at 710.

[44] Id at 709. Here, the court elaborated on the *scènes à faire* doctrine, which holds that where "it is virtually impossible to write about a particular historical era or fictional theme without employing certain 'stock' or standard literary devices," those devices get no protection from copyright. *Hoehling v Universal City Studios*, 618 F2d 972, 979 (2d Cir 1980). As the *Altai* court noted, it is often difficult to write a program to perform a particular function without utilizing standard techniques. 982 F2d at 709. For instance, in *Altai* itself, much of the similarity in the programs arose because both were designed to interact with the same operating systems. Under the *Altai* court's reading, traditional *scènes à faire* doctrine would exclude protection when any such situation arose. For example, when hardware requirements or the software demand that the program be written in a certain way, copyright would not protect the program's structure. Similarly, if widespread practices in the programming community point to a single way of approaching the design, that design approach cannot be copyrighted. This limitation seems very close to the constraint on protecting elements drawn from the public domain.

[45] *Altai*, 982 F2d at 707-09. In the *Altai* court's view, efficiency concerns may limit the choice of subroutines in a program. Since efficiency is supposedly the industry's goal, the court found that programmers may independently hit upon the same design method. Thus, the court concluded that a similar, efficient structure may just as likely be an independent creation as a copy. Id at 708. This rule also stems from the merger doctrine, which prevents copyright from extending a monopoly over a given idea when the idea can only be expressed in a limited number of ways. Id. See also *Concrete Machinery Co., Inc. v Classic Lawn Ornaments, Inc.*, 843 F2d 600, 606-07 (1st Cir 1988).

[46] *Altai*, 982 F2d at 710.

[47] Id at 712.

[48] Id.

ature and computer software. For software, unlike for literature, the creative expression is just a set of instructions aimed at a utilitarian goal.[49] Thus, while both literature and computer software require an inquiry into the proper border between idea and expression, software cases involve the additional difficulty of fixing the proper boundary between the mutually exclusive areas of patentable functionality and copyrightable expression. The importance of this last distinction becomes clearer after considering the reasons for protecting software structure.

A. Why Bother Protecting Non-Literal, Non-Visual Structure? The Case for Copyright Protection

The case for protection against infringement through non-literal similarity depends critically on the assumption that a second programmer who copies the structure of the first can cut the time and costs required to bring a similar product to market. Rapid innovation is a fact of life in the software industry. When a new product offering a unique feature or service is first introduced, it will attract both customers and imitators. Once those imitators bring their own products to market, the number of units the original developer can sell, or the price that he can charge, declines. In the absence of further innovation, sales of the original product will taper off over time.[50]

The window of opportunity that copyright law creates by protecting against imitations is crucial. If too short, then the rewards may be too small to merit production of the software in the first place. The need to preserve an adequate lead time suggests the need for intellectual property rights to protect the original programmer's innovation. This is a public good problem: since it is hard to exclude others from copying the innovation and marketing it themselves, the original programmer cannot reap the full value of his labor without copyright protection.

Articulating the need for protection, however, does not lead us to the rule best suited to fulfilling that need. Suppose, for example, that BlurtPerfect Corporation wants to add a new feature to its already existing software product, perhaps a spell-checker to its renowned BlurtPerfect word processor. Assume a

[49] This reasoning would apply to some written work as well, such as technical manuals or other instructional materials. See Goldstein, 2 *Copyright* § 8.4.1.3 at 109-11 (cited in note 22).

[50] See Anthony Lawrence Clapes, *Software, Copyright and Competition* 25-27 (Quorum, 1989).

world where BlurtPerfect enjoys only trade secret and literal copyright protection.[51] Further suppose that the spell-checking feature is the first of its kind on any word processor and that this development will give BlurtPerfect a competitive advantage, but one lasting only six months. Thereafter, other manufacturers will be able to develop spell-checkers independently for their word processors. BlurtPerfect calculates that the increased profits made during the six-month lead time will just exceed the costs of the improvement, so it decides to add the innovation.

In such a case, if BlurtPerfect's competitors, by whatever means, copy elements of the program's non-literal structure, thereby reducing their own development time by a month, then it would make no sense for BlurtPerfect to add the innovation. Two sorts of rules can remedy this outcome. A patent-like rule would grant BlurtPerfect exclusive use of the spell-checker feature whether its rivals develop it independently or not. Alternatively, one could protect BlurtPerfect against non-literal copying, leaving rivals free to develop the feature themselves. The second of these rules is superior. While both rules get a spell-checking feature into the marketplace, the first forces other manufacturers either to license the technology from BlurtPerfect or go without a spell-checker altogether. In a highly competitive industry where progress is incremental, that amounts to a recipe for stifling innovation.

Existing patent law reflects that conclusion. Although it is now possible to patent certain kinds of software and software features, protection is difficult to obtain and the application process is quite slow. Only a narrow range of software can satisfy the requirement that a program or feature be novel, non-obvious, and useful.[52] That may be as it should. If the software industry is

[51] Trade secret law arises from state common and statutory law and protects "[a]ny formula, pattern, device or compilation of information" used in business to give an advantage over competitors. Restatement of Torts § 757 comment b (1939). Trade secret law prevents anyone in a confidential relationship with the holder of the secret (an employee, for example) from disclosing that information. See, for example, *Q-Co Industries, Inc. v Hoffman*, 625 F Supp 608 (S D NY 1985).

[52] See 35 USC §§ 101-03. See also Clapes, *Software* at 208 (cited in note 50). Software patents are gaining in popularity in part because of the unsettled state of copyright law. Roughly 16,000 software-related patents have been issued to date, though their validity has yet to be tested in the courts. Mitch Betts, *Vendors seek patents as copyright suits grow*, Computerworld 109 (Oct 11, 1993). Patents have already been granted on a range of important pieces of software, including a data compression program, John Soat, *Patent Litigation; Did Microsoft 'Stac' Deck?*, InformationWeek 15 (Feb 1, 1993), and a multimedia encyclopedia. *Patent pyrotechnics; CD-ROM publisher claims rights to advanced multimedia retrieval technology*, Computerworld 28 (Nov 22, 1993).

such that protecting the developer's lead time is a sufficient incentive for innovation, then patent protection is unnecessary (and only a burden on subsequent innovation).[53] This seems to be the actual state of affairs. Software technology had exploded even before patent protection was generally available.[54]

Although providing copyright protection for some non-literal program elements appears the better regime, critics have raised two objections. First, some argue that a copyright rule would be superfluous. The initial justification for the rule depends on the assumption that non-literal copying somehow aids the development of rival programs. Critics of copyright protection argue that while this assumption is correct in a limited number of cases, it does not fit the vast majority. Commercial programs are generally distributed in object code form only. While expert programmers may be able to read small portions of such code, its helpfulness in reconstructing the original source code is limited. Indeed, it is telling that the two leading cases dealing with this copyright question, *Whelan* and *Altai*, involved insiders who stole company secrets. That observation has led some to conclude that trade secret protection is enough to solve the problem at hand.[55]

This argument against copyright protection suffers from several weaknesses. Unlike copyright, trade secret law makes no presumption of copying from access and similarity; the burden remains on the plaintiff to show misappropriation. Software companies may also have legitimate reasons for showing their source code to outsiders. For example, programmers from different companies often exchange ideas and code.[56] If companies had to rely solely on trade secret protection, this fruitful practice—which, under trade secret law, often constitutes waiver of trade secret protection[57]—might come to an end. Finally, trade secret protection cannot in any case protect against those instances in which a

[53] This does not render patent law entirely inappropriate for computer programs. Where the development costs are heavy or the potential for reward uncertain, then a lead time may not provide enough incentive to innovate. In such cases, patent law has a role to play in creating adequate incentives.

[54] The Patent and Trademark Office generally began to grant patents for software only after the Supreme Court's decisions in *Diamond v Diehr*, 450 US 175 (1981), and *Diamond v Bradley*, 450 US 381 (1981).

[55] See Note, *Copyright Protection for Software Architecture: Just Say No!*, 1988 Colum Bus L Rev 823, 849-51.

[56] See Maury M. Tepper, III, *Copyright Law: Integrating Successive Filtering into the Bifurcated Substantial Similarity Inquiry in Software Copyright Cases: A Standard for Determining the Scope of Copyright Protection for Non-Literal Elements of Computer Programs*, 14 Camp L Rev 1, 55 (1991).

[57] See Restatement of Torts § 757 comment b (1939).

programmer with a great deal of patience disassembles the object code of a program into a more intelligible source code.

Second, critics of copyright for non-literal structure object that such protection will lead to patent-like protection. When we allow the copyrighting of non-literal, non-visual structure, these critics note, we protect not only how the program works, but also what the program does. Yet this effectively circumvents the more stringent requirements of patent law. That increases the risk of providing too much protection, of awarding a monopoly where there should be none, and of stifling innovation.

This is the more difficult objection, and the remainder of this Comment responds to it. The above analysis of how to preserve incentives points us in the right direction. First, one must ask what sorts of things a court should look at when evaluating structural similarity. These will be the elements of a program which, if copied, allow an infringer to reproduce a program more easily. This inquiry, however, is distinct from the second problem: deciding which elements of a program should be subject to the more rigorous standards of patent law.

B. *Whelan* and Its Critics: A Defense against Overstatement

The analysis set forth in *Whelan* has drawn generous amounts of criticism.[58] I shall not explore most of these as this Comment is not so much concerned with defending *Whelan* as with offering a way for courts to approach the software copyright issue. Toward this end, however, it is still important to understand the chief criticism of *Whelan* and why it is in part misdirected.

Many have attacked *Whelan*'s distinction between idea and expression.[59] Under *Whelan*, a program's general function or purpose (for example, the efficient operation of a dental lab) is unprotectable. Everything that is not necessary to this function counts as protectable expression. Nimmer objects that the "crucial flaw" of this reasoning is that more than one idea underlies any given computer program.[60] This criticism is undoubtedly correct. Although a program's general function counts as uncopyrightable idea, there may be other functional elements in

[58] See *Altai*, 982 F2d at 705-06; Menell, 41 Stan L Rev at 1082-83 (cited in note 3); Note, *Idea, Process, or Protected Expression?: Determining the Scope of Copyright Protection of the Structure of Computer Programs*, 88 Mich L Rev 866, 881-82 (1990).

[59] *Altai*, 982 F2d at 705, citing *Nimmer* § 13.03[F] at 13-62 (cited in note 22).

[60] 3 *Nimmer* § 13.03[F][1] at 13-127.

the structure which also count as uncopyrightable ideas.[61] But fixing the proper distinction between what belongs to patent law and what belongs to copyright is not the only question that *Whelan* attempts to address. The same *Whelan* rule also tries to establish a guide as to which structural elements a judge should compare for copyright purposes. The *Whelan* answer is that a judge may compare structure at any level of abstraction, however high.

This is not an absurd position. By way of illustration, suppose that a court had to decide a case involving a program that allowed salesmen to calculate the proper size of replacement parts for a customer's machinery.[62] The heart of the program consists of several algorithms that engineers use to calculate the proper size. A rival company is now suspected of copying the original software into a similar product of its own. The court makes the following findings: no similarity between the two object codes and source codes; substantially similar data and control flow; substantially similar overall structure within the modules that perform the calculations; identical algorithms; and substantially similar ordering of some modules within the overall program when described at a high level of abstraction, but not so with the ordering on the lower, more detailed level.

Initially, a court should determine which elements of the program are even eligible for copyright protection. One might ask, for instance, whether the algorithms should be protected by copyright law at all. One might choose to characterize the algorithms as processes and thus only extend protection if they meet the criteria demanded by patent rules.

That inquiry is distinct from deciding which similarities one should compare in the first place. For instance, does substantial similarity of data flow count as an infringement? How similar does the ordering of modules have to be before one finds infringement? These questions are more closely akin to one-half of the problem faced by Learned Hand. At some high level of generality, copying is not a concern—only at the lower levels does it become

[61] For instance, if market necessities force a programmer to design cotton marketing software with certain functional similarities that all such programs must have, then the "idea" of the program includes those functions and not just its overall purpose. *Plains Cotton Cooperative Association of Lubbock, Texas v Goodpasture Computer Service, Inc.*, 807 F2d 1256, 1262 n 4 (5th Cir 1987).

[62] This situation is modeled on *Gates Rubber Co. v Bando American, Inc.*, 798 F Supp 1499 (D Colo 1992), vacated in part and remanded in *Gates Rubber Co. v Bando Chemical*, 9 F3d 823 (10th Cir 1993).

so. For example, if Shakespeare held an enforceable copyright for *Romeo and Juliet*, *West Side Story* would arguably infringe it. But *West Side Story* would infringe not because it, too, is a boy-meets-girl story; rather, it would infringe because it shares specific expressions of that general plot. Somewhere between a very abstract and a very specific description of plot lies a level at which a court should make a judgment as between the two stories. The *Whelan* test can best be understood as addressing this question of finding the right level of generality. Nimmer and *Altai* take it only as an attempt to sort out elements properly protected by copyright from those properly governed by patent.

This is not to suggest that the *Whelan* approach is without its difficulties. Nimmer and *Altai* correctly conclude that *Whelan* gives courts little reliable guidance on how to separate out the copyrightable elements from the patentable (though they do not word their objections in this way).[63] Also, *Whelan*'s insistence that the comparison between computer programs can take place at the highest level of abstraction appears rather rigid. While it is not necessarily absurd to compare programs at the highest levels,[64] it should not be mandatory. For its part, *Altai* improves on *Whelan* by better articulating the twin difficulties that lie behind the computer copyright problem. But *Altai* contains problems of its own.

C. *Altai* and Its Supporters: Caveats

The *Altai* decision made three innovations in the debate over non-literal software copyright: its adoption of its own version of the abstraction test;[65] its provision of a specific list of items to be filtered out as uncopyrightable; and its insistence that the filtration occur prior to the comparison of the two programs.[66] The decision is an important and positive contribution and was immediately recognized as such,[67] but its innovations are not entirely for the better.

[63] *Altai*, 982 F2d at 705; 3 *Nimmer* § 13.03[F] at 13-62.34 (cited in note 22).

[64] Note that the *Altai* decision does not disagree with this point. Its abstraction doctrine calls for a comparison of the two programs at every level, including the most general. 982 F2d at 707.

[65] Earlier courts had applied a similar analysis. See *Lotus Development v Paperback Software International*, 740 F Supp 37, 61 (D Mass 1990).

[66] *Altai*, 982 F2d at 706-12.

[67] See David Bender, Computer Associates v. Altai: *Rationality Prevails*, The Computer Lawyer 1 (Aug 1992).

The shortcomings result in part from unforeseen consequences of *Altai*'s doctrine.[68] They also result from the *Altai* court's failure to recognize the need to articulate a clear line between patent and copyright, and to adequately protect certain structural innovations that fall on the copyright side of the line.

Altai's formulation of the abstraction doctrine can be understood as an attempt to protect such structural innovations. Following Hand's approach, the court suggested that "a court should dissect the allegedly copied program's structure and isolate each level of abstraction contained within it."[69] Then, after filtering out unprotectable elements, a court must compare each of these levels to the allegedly infringing program.[70] The *Altai* test, though it looks theoretically tidy, becomes somewhat messy in practice. After all, comparing each level of abstraction for a given program is a cumbersome task—with perhaps millions of lines of code, multiple authors, and theoretically limitless ways of describing the program's functionality.

In defining the relevant levels of comparison, the appellate *Altai* decision followed the district court's lead, noting: "As applied to computer software programs, this abstraction test would

[68] One controversial element of *Altai* appears at first to be only an innocuous methodological innovation. The *Altai* court ruled that unprotectable elements of a computer program must be filtered out before a comparison is made to judge substantial similarity. The justification for this move seems straightforward: if one filters after comparing, unprotected expression may be included in the judgment as to substantial similarity. 3 *Nimmer* §§ 13.03[F] at 13-142 to 13-143 (cited in note 22).

The observation seems a near tautology, but it has important consequences, especially in disputes involving visual interfaces. For example, *Apple Computer, Inc. v Microsoft Corp.*, 799 F Supp 1006 (N D Cal 1992), hinged on the copyrightability of the look and feel of the Macintosh interface. Most of the elements of the Macintosh interface lacked the requisite originality for copyright protection, but arguably the look and feel of the interface was original and protectable. The district court, adopting *Altai*'s methodology, filtered out the unprotectable elements, leaving nothing behind to copyright (since a bare gestalt is rather difficult to identify in the absence of its components). Id at 1047.

This result seems contrary to the reason originally advanced for the filter/compare sequence. The test is not supposed to be substantive; it is not supposed to decide what is and is not copyrightable, but rather should only set the preconditions for comparison. By filtering before one compares, one effectively rules out the possibility of finding a copyright in the bare look and feel of an interface made up of unoriginal components. That amounts to a substantive conclusion about what should and should not be copyrighted. It may very well turn out that one wants to deny copyright protection to look or feel, but that should be an independent question from deciding whether to compare and then filter or vice versa. The district court in *Gates Rubber* recognized this problem. 798 F Supp at 1517. Although the Tenth Circuit vacated this part of the *Gates Rubber* opinion, 9 F3d at 849, it is important to note that filtration prior to comparison runs the risk of losing the forest for the trees. See also Miller, 106 Harv L Rev at 1006 (cited in note 10).

[69] *Altai*, 982 F2d at 707.

[70] Id.

progress in order of 'increasing generality' from object code, to source code, to parameter lists, to services required, to general outline."[71] The difficulty with this analysis is not just that the court mistakes what counts as different levels of generality—though it is partially that.[72] Nor is the difficulty just a lack of technical sophistication—though it is again some of that.[73]

The real difficulty lies with the Hand test itself—or more specifically, the *Altai* court's version of it. Learned Hand designed his abstraction test to solve a particular problem: to find *one* right level of generality at which a court could compare a potentially infringing literary work to the original. The *Altai* court revises the Hand test by abstracting out *each* level of generality in a given computer program. In so doing, it also removes the justification for applying the abstraction doctrine in the first place. Hand conceived of his abstraction test as a way to distinguish idea from expression by picking a level of generality to make a comparison. Under his test, a court fixed that level pragmatically—with an eye on the balance between incentives to produce and the costs of restricting access. As such, whatever level of generality a court ultimately chose, the Hand test held out the hope of an optimal balance of benefit and cost. *Altai*'s abstraction test attempts no such balance.[74] By insisting that a court compare each level of abstraction between programs, *Altai* abandoned the goal that copyright protection for a given level of abstraction will preserve incentives to create new software.[75]

A different problem infects the *Altai* court's understanding of those elements that should be filtered out to yield the golden nugget of protected expression. Articulating specific categories that fall beyond the scope of copyright amounts to a significant advance over *Whelan*; such categories go a long way toward delineating where copyright rules end and patent begins. Unfortu-

[71] Id at 714.

[72] Just as an English translation of the *Iliad* is not more "general" than the Greek original, source code is not more "general" than object code. It is simply a different language.

[73] For a more systematic treatment of how to delineate levels of abstraction within a computer program, see Note, *Defining Computer Program Parts Under Learned Hand's Abstraction Test in Software Copyright Infringement Cases*, 91 Mich L Rev 526, 533 (1992) (proposing a set of "computer part definitions" to be used in conjunction with Hand's abstraction test).

[74] To be fair, the *Altai* court did justify its overall test by balancing competing interests. 982 F2d at 711-12.

[75] The same criticism applies as well to the more sophisticated attempts to apply the *Altai* abstraction test. See, for example, Note, 91 Mich L Rev 526; *Gates Rubber*, 9 F3d at 834-36.

nately, one of the key *Altai* categories, elements dictated by efficiency, is not as well-crafted as one might like; the category is both too broad and too narrow.

The *Altai* court maintained that "efficiency concerns may so narrow the practical range of choice as to make only one or two forms of expression workable options."[76] As a result, the fact that two programmers have reached the same result may lead to an inference of independent creation rather than an inference of copying.[77] Nimmer throws some light on what the court intended by this argument, using the example of two well-known data sorting algorithms: a quicksort and a bubble sort.[78] When sorting, for example, a list of a thousand names into alphabetical order, the bubble sort requires one million permutations, but the quicksort only seven thousand. Thus the quicksort is significantly more efficient. Nimmer points out that any two sorting programs are likely to employ the quicksort algorithm and yet may well have been created separately; the similarity therefore does not give rise to an inference of copying.[79] This argument, however, only goes so far. Suppose the slower bubble sort algorithm is a well-known industry standard, but a new program enters the market utilizing the better quicksort technique. If shortly thereafter another program enters the market with the quicksort algorithm, then one might infer that the second program copied from the first.

The better justification for *Altai*'s denial of copyright to efficient elements of computer structure stems from the by-now-familiar worry about innovation.[80] Extending copyright protection to the non-literal aspects of a program's structure runs the risk of extending patent-like protection to a subject matter for which patent was not intended. This argument for the efficiency test, however, has an important drawback. There is a difference between an algorithm that the program executes and the algorithm embodied directly in the code. Suppose a programmer writes a program designed to alphabetize a list of names. Suppose further that he selects the quicksort algorithm as the most

[76] *Altai*, 982 F2d at 708. One commentator has defined efficiency concerns in programming as speed of execution, level of memory utilization, and level of compatibility with data storage methods. Menell, 41 Stan L Rev at 1085 (cited in note 3).

[77] *Altai*, 982 F2d at 708-09.

[78] 3 *Nimmer* § 13.03[F][2] at 13-131 to 13-132 (cited in note 22).

[79] Id at 13-132.

[80] See generally Menell, 41 Stan L Rev 1045. See also *Whelan*, 797 F2d at 1236-38; *Altai*, 982 F2d at 701.

efficient available technique. If he is a good programmer, then the coding of that algorithm will be efficient in a different sense—the code will take up little space in memory and will be able to execute the quicksort algorithm with a minimum number of instructions to the computer. Although an ideally coded version of the quicksort algorithm may theoretically exist, as a practical matter, competent programmers can encode that process in a variety of ways. A clumsy programmer's coding may be less efficient than that of a talented programmer, even though both use the efficient quicksort algorithm.

Copyright should not protect an efficient algorithm that the program merely executes (quicksort, for example). Such a sorting algorithm can appear in a variety of software, and allowing its inventor to protect it through copyright may well quash subsequent innovation. However, there is no similar reason against protecting an algorithm embodied directly in the code (since that algorithm reflects the programmer's skill). Generally, there are a variety of ways of coding any given algorithm, process, or function. Extending copyright to efficient code runs little risk of locking up the fundamental techniques of programming on which innovation and progress depends.

This realization is important because it illustrates a persistent problem. Courts rightly worry about granting copyright protection when only one route or a small number of routes are available to programmers. In such cases, the impulse is to provide protection, if at all, through patent law. The *Whelan* court addressed the problem by denying copyright protection to any structural element that was necessary to accomplish the general function of the program. The *Altai* court tackled the problem by declaring certain elements of a program to be unprotectable based on efficiency concerns, external concerns, and the merger and *scènes à faire* doctrines. Neither approach is free from difficulty, but the general goal of each is worth pursuing further.

IV. Rounding the Square Peg

For literary works, Learned Hand's abstraction test allows a judge to balance the incentives to produce against the costs of restricted access by picking a level of generality along a single continuum. Computer program copyright also involves a tradeoff between incentives and access costs, but the argument of this Comment thus far shows that it makes no sense to think about that tradeoff in terms of a single continuum. Rather it is best to break the problem into its two constitutive parts: guaranteeing that software companies have an incentive to create new software and preserving the ability of newcomers to innovate on existing technology. The *Whelan* rule attends to the former concern but slights the latter, while the *Altai* rule has the reverse weakness. A better approach treats each half of the tradeoff in turn, and this section proposes a test to do just that.

A. Abstraction or Selection?

The challenge when comparing literary plots is to pick a point of abstraction that is specific enough so that not all works of a literary genre count as substantially similar, yet general enough so that a work may be protected against "immaterial variations."[81] The analogous problem in the computer context is to pick out those non-literal structural features that competitors would want to pirate. Although the complexity of software in turn makes the determination more complex, courts are not quite as much at sea as we might think. Courts have found ready-made handrails. Hierarchy of modules,[82] file structure,[83] data flow,[84] command flow,[85] and parameter lists[86] all help guide a decision. One commentator has even described fourteen characteristics or levels of abstraction that courts can use to make comparisons.[87]

As with literature, a hard and fast rule is of little use in the software context. Unfortunately, while judges are commonly familiar with literature, they are not necessarily familiar with the intricacies of computer technology. Judges have well-developed intuitions about what is and is not important in comparing two works of literature. One cannot hope for a similar understanding of computer programming, due to its more technical nature.

Inevitably, judges will have to rely on expert testimony and past precedent to guide the selection of structural elements. Even so, the underlying goals of copyright law should guide their discretion. Copyright protection must reward initial market entrants

[81] *Nichols v Universal Pictures Corp.*, 45 F2d 119, 121 (2d Cir 1939).

[82] See, for example, *Pearl Systems v Competition Electronics*, 1988 US Dist LEXIS 15428, *7-12 (S D Fla 1988).

[83] See, for example, *Whelan*, 797 F2d at 1242-43; *CMAX/Cleveland v UCR*, 804 F Supp 337, 355 (M D Ga 1992).

[84] See, for example, *Whelan*, 797 F2d at 1242-43; *CMAX*, 804 F Supp at 355.

[85] See, for example, *Altai*, 982 F2d at 714-15.

[86] Id.

[87] Davis, *The Nature of Software*, 5 Software L J at 317-25 (cited in note 12).

without locking out later innovators. Given this goal, courts should compare two programs with an eye toward whether their similarities are ones that reduce the cost and time of producing the infringing program so much as to create a disincentive for the initial programmer to enter the market. If the similarities are not of this sort, then the old program should not stand in the way of the new—just as one would not object to a play whose plot similarities are not extensive enough to interfere with the commercial prospects of the book that it resembles.

This approach takes the place of the *Altai* court's version of the abstraction test, and in doing so frees courts from the notion that their job is to pick out software features on the basis of an inquiry into levels of generality. Learned Hand's original justification for talking about levels of generality was closely tied to works of literature. For computer software it makes sense to drop talk of an abstraction test altogether and instead frame the problem as one of selection.

B. The Application of Merger Doctrine to Clarify the Patent/Copyright Border

Some courts have implicitly used Hand's abstraction test or the *Whelan* rule when trying to fix the boundary between patent and copyright.[88] The above analysis of *Whelan* and *Altai* suggests that such an approach may be misguided. Therefore, rather than depending on the admittedly ad hoc abstraction test, it is better to try to formulate rules with sharper edges. Here the *Altai* court was on the right course insofar as it attempted to employ the merger and *scènes à faire* doctrines to create clear rules governing what should and should not be copyrightable: elements in the public domain, elements dictated by external factors, and elements mandated by efficiency. What the *Altai* court failed to do was to recognize that all these rules are just particular instances of a broader merger doctrine—a doctrine that extends beyond the categories that the *Altai* court would filter out.

The merger doctrine prevents copyright from attaching to any expression when the underlying idea can only be expressed in a limited number of ways.[89] When this condition holds, the

[88] See, for example, *Altai*, 982 F2d at 706-07; *Atari Games Corp. v Nintendo of America Inc.*, 975 F2d 832, 839 (Fed Cir 1992); *Lotus Development Corp. v Paperback Software International*, 740 F Supp 37, 60 (D Mass 1990).

[89] See, for example, *Concrete Machinery Co., Inc. v Classic Lawn Ornaments, Inc.*, 843

idea and expression have in essence "merged"; thus, protecting the expression would necessarily protect the idea. Unfortunately, the doctrine is somewhat slippery. For example, one court has held that copyright does not protect a jewel-encrusted, life-like bee pin since there are only a limited number of ways to express the idea of such a pin.[90] This particular outcome depends, of course, on what the court picks as the relevant idea. If the court had decided that the idea was that of a bejewelled pin, then merger would not apply to block a copyright on this particular bee, because there are many kinds of bejewelled pins. This shows a general limitation on the usefulness of merger: since the idea of an expression is often debatable, other considerations must be used to determine what will be considered the relevant idea.

The trick is to identify those elements of a computer program that we fear would stifle innovation if protected by copyright rules.[91] These elements should count as the "ideas" of a program. If those elements can be encoded in only a small number of ways, then the merger doctrine would properly prevent a monopoly of that "idea."

There is some agreement as to what should count as an idea for these purposes. The kind of program is one example. If a program that calculates cotton futures, for instance, can have only one structural configuration, then that structure cannot be copyrighted since it would amount to patent-like protection.[92] Likewise, certain program features should also be treated as "ideas." Consider a standard word processor. Features such as spell checking, grammar checking, outlining, footnoting, and underlining are standard on any mainstream product. Again, copyright protection for any of these would tend to stifle competition as surely as if a patent were granted. Algorithms which have a general use in a number of different sorts of programs should also be counted as ideas. For example, bubble sorts and quicksorts are useful in data base programs, word processors, spreadsheets, and any other software that requires the sorting of data. If these could only be expressed in a small number of ways, then they too look like favorable candidates for application of the merger doctrine.

F2d 600, 606-07 (1st Cir 1988).

[90] *Herbert Rosenthal Jewelry Corp. v Kalpakian*, 446 F2d 738, 742 (9th Cir 1971).

[91] For a discussion of the same problem in the context of computer compatibility, see Note, *Merger and the Machines: An Analysis of the Pro-Compatibility Trend in Computer Software Copyright Cases*, 45 Stan L Rev 1061, 1072-97 (1993).

[92] See *Plains Cotton*, 807 F2d at 1262.

Yet the merger doctrine would not exclude all program elements from copyright protection. Elements that a programmer could design in different ways would still be protected. Hence, many of the structural features of a computer program seen in *Whelan* and *Altai*—such as data flow, data structure, macros, and parameter lists—should all be deemed copyrightable. As a general rule, courts should seek to distinguish program elements which display the exercise of the programmer's art from those program elements which are the objectives of his art. The merger doctrine fleshes out that rough distinction.

While the distinction must remain somewhat ad hoc, its application is now at least guided by the understanding that copyright should not stand in the way of progress in software technology. This rule does not eliminate discretion in making that choice, but it does frame the issue more clearly than previous tests.

CONCLUSION

Whelan and *Altai* are perhaps best viewed not as opposing cases, but as points on a continuum in the development of computer copyright. Both agree that the non-literal structure of a computer program deserves protection. *Whelan* articulated a rule whereby copyright would protect everything that is not necessary to the purpose or function of the program. *Altai* did not altogether disagree with that rule, but attempted to add hard edges to a rule by which courts could sort out protectable from unprotectable elements. The rules proposed above, the application of the merger doctrine and the selection test, do not dramatically differ from the conclusion reached by the *Altai* court. But they go further to resolve the confusion that *Altai* only partially addressed. By making these distinctions more reliable, courts can go a long way to shape proper incentives for future innovation.