January 1996

A Model Software Petite Patent Act

Mark Paley

Follow this and additional works at: http://digitalcommons.law.scu.edu/chtlj

Part of the Law Commons

Recommended Citation
Available at: http://digitalcommons.law.scu.edu/chtlj/vol12/iss2/3
A MODEL SOFTWARE PETITE PATENT ACT*

Mark Aaron Paley†

TABLE OF CONTENTS

I. INTRODUCTION ........................................... 303
II. WHY WE NEED THIS ACT .................................... 304
   A. What is the Nature of Computer Software? ......... 304
   B. Software Falls Into a Gap Between Copyright and Patent Law .......................... 305
   C. Need for Harmonization with Other Countries .... 311
III. SOFTWARE AS COPYRIGHTABLE MATERIAL .............. 313
   A. Software as Art ........................................ 313
      1. Software's Dual-Natured Literal and Nonliteral Expressiveness ...................... 313
      2. Judicial Acceptance of the Notion of Duality of Expression ...................... 314
   B. Why Copyright Law Does Not Protect Software Well ........................................ 315
      1. Derivative Works ...................................... 315
      2. Copyright's Duration is Excessively Long for Software ............................ 317
IV. SOFTWARE AS A USEFUL OBJECT (PATENTABLE MATERIAL) ........................................... 318
   A. What is Patentable? ...................................... 318
   B. What Part of Software is Patentable? ............ 318
      1. Patentable Algorithms ................................ 318
      2. Algorithms Are the Same as Nonliteral Elements in Copyright Law ................ 319
   C. Does a Software Algorithm Qualify as a Patent Act Process? ........................... 320
      1. Supreme Court Definition of “Process” ............. 320
      2. Federal Circuit Court’s Special Relationship to Software Algorithms ............ 322
      3. RAM/ROM Exemplifies “Process/Machine” Definitional Conundrum ................ 322

* Copyright © 1996 Mark Aaron Paley.
† B.S., Magna Cum Laude, May 1986, State University of New York at Albany; J.D., Magna Cum Laude, May 1992, State University of New York at Buffalo; Candidate, L.L.M., George Washington University, 1996.
4. The Federal Circuit Court’s Definition of “Process” .......................... 324

D. Does a Software Algorithm Qualify as a Patent Act §101 “Machine?” ............ 325

E. Why Patent Law Does Not Protect Software Well. 327
   1. Current Patent Protection is Both Too Narrow and Too Broad ........................ 327
   2. Modern Software Breaks All the Rules ........................................... 328
   3. It Is Hard to Win a Software Patent Infringement Suit ............................. 329
      a. The Software Act is Better Than Current Law’s Best Case Scenario ............ 329
      b. Under the Current Law—Even When You Win, You Lose ......................... 331
   4. Overreaching Patent Claims ......................................................... 332
   6. Supreme Court Calls for Legislation .............................................. 334

V. SOFTWARE AS COMMERCE (TRADEMARK MATERIAL) ...... 334
   A. Protect Only Software Used In Commerce ........................................ 334
   B. Noncommercial Use Is Not Infringement .......................................... 336
   C. Vaporware .................................................................................. 336
   D. Insufficient Lead Time to Recapture Software Development Costs ............... 337
   E. Orphanware ................................................................................ 338
   F. Patent “Bracketing” and “Mapping” .................................................. 338

VI. NEW FEATURES UNDER THE SOFTWARE ACT .......... 339
   A. The Distinction Between Printware and Machineware .............................. 339
   B. Compulsory Licensing of Algorithms .................................................. 340
   C. Blanket Licensing .......................................................................... 341
   D. Reverse Engineering/Decompilation Right ............................................ 343
   E. Compulsory License Right to Port an Algorithm into New Hardware .......... 345
   F. Exhaustion ..................................................................................... 346
   G. Cheap Easy Filing ........................................................................... 347

VII. WHAT CONSTITUTES INFRINGEMENT UNDER THE SOFTWARE ACT ............... 348
   A. What is the Purpose of an Infringement Action? .................................... 348
   B. Overview of the Abstraction-Filtration-Comparison Test to be Codified ......... 349
   C. The Abstraction Step ...................................................................... 352
   D. The Filtration Step ........................................................................... 352
   E. Not Filters ...................................................................................... 354
I. INTRODUCTION

Computer software is very important. The Supreme Court took judicial notice that the value of computer programs used in the United States was $43.1 billion in 1976 and projected to be $70.1 billion in 1980. Software will only grow in importance as every aspect of the world's society and economy becomes dependent upon computer technology in the Information Age. Yet U.S. law is criticized as providing uncertain protection of software's value. Confusion has resulted from Congressional and Court inaction. The world takes its cues from U.S. law, as the United States is the world leader in

---


This paper is consistent with the *Manifesto* published by four authors traditionally identified as opposed to broad copyright protection of software, wherein they advocate a *sui generis* approach to software protection. Pamela Samuelson et al., *A Manifesto Concerning the Legal Protection of Computer Programs*, 94 *Columbia L. Rev.* 2308 (1994) (The key authors, Samuelson and Reichman, propose a fuzzy outline of what a law like this should look like, while at the same time other key authors in the field (Miller, Ginsberg, Goldstein) criticized the proposal as being too generic, not specific enough, and confusing. The *Manifesto* called upon other writers to resolve these criticisms by drafting a specific, concrete, unconfusing model act. This article does exactly that.)
software. Thus, it is increasingly important that a concrete proposal be drafted to redefine software's role in the intellectual property statutory scheme. A law to protect software is conceptually difficult, however, because software itself breaks the boundaries of traditional copyright, patent, and trademark law. Software is an amalgam that blurs the line between copyrightable art and patentable utility. It blurs the line between intellectual property (valuable information) and physical property (nuts and bolts). Traditional physical objects have transcended their physicality by becoming so charged with information that they are now perceived in terms of what they do, rather than what they are.4

This paper explores that blurred line by examining the confused state of current law. The realities of the nature of software are examined. Preconceived traditional notions of intellectual property law are discarded where they do not match software's nature. The result is a proposed Software Act5 that is as much a unique hybrid as the software it seeks to protect. The new framework removes much of the current overlap in laws by borrowing the best features offered by copyright, patent, and trademark law. The theories underlying these draft provisions are explained, and the relationship between this proposal and international conventions is examined.

II. WHY WE NEED THIS ACT

A. What is the Nature of Computer Software?

The preliminary problem confronting courts addressing the issue of computer software protection is determining simply what computer software is. Computer software defies categorization into the neat legal compartments drawn to fit copyright, patent, and trademark law. There is no existing body of law that can be incorporated by reference to obtain an adequate, ready-made solution.6 Software is a hybrid exemplifying conflicting attributes within each of the traditional intellectual property fields. Software's literary art, the words or computer code, is protected by copyright law.7 Such art exhibits imagination, originality, creativity, and individuality often far in excess of that

5. See infra Appendix.
found in the mundane works that receive time-honored copyright protection. Yet, there is no clear consensus as to what software is.

Section 102(b) of the Copyright Act forbids protection of "any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described." Software may be logically viewed as comprising any of these, because software is such a useful form of art. Software's utility, where protected at all, is protected separately under patent law. In limited situations, patents have been granted upon aspects of computer software—primarily comprising those forbidden by the Copyright Act. Yet, the scope of patent law does not clearly encompass perhaps the most useful aspect of software—the algorithms. Section 101 of the Patent Act protects only a "new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof." Hence, much software falls within a no-man's land—sharing art and utility aspects of both copyright law and patent law, yet encompassed within neither.

B. Software Falls Into a Gap Between Copyright and Patent Law

Copyright and patent law pull software apart to protect it. They inconsistently treat software like a book for one purpose and like a machine for another. Copyright law prevents others from copying the words comprising software as though they were text and the images on the screen as though they were pictures. Yet, literal copying is rare. Far more often, a competitor uses new text to achieve a workalike program. Courts are split on how to protect the way a program interacts with a user, since the menu items, command choices, and the visual or audible elements can be characterized as copyrightable text, pictures, sound, or a noncopyrightable functional method of operation. Where patent law does attempt to protect

8. Miller, supra note 3, at 983-84.
12. Id.
software's functionality, it may only do so by stretching the meaning of protectable "process" or "machine" matter beyond traditional recognition.

Where copyright and patent law protect software, they do both too much and too little. Copyright and patent law do too much by providing a grant of total exclusion.16 Computers were deliberately designed to share software, hardware, information, and data. A far more appropriate proprietary grant for software would be the limited right to compensation when software is used in a commercially shared way. Copyright and patent law also do too much by protecting software for a duration far in excess of its useful life.17 Simultaneously, copyright and patent law do too little to protect software, because their terminology does not reach far enough to encompass aspects of software close to the borders of unprotectable matter under either law. For example, neither copyright nor patent protects ideas per se. Yet, the border between a patentable algorithm and a nonpatentable idea is a very thin line. Likewise, the border between these arguably copyrightable expressions and a clearly noncopyrightable function is blurred. Even when there are multiple ways like this to express a noncopyrightable idea, does it make any sense for the law to require the last comer to use the least efficient of such expressions simply because a first comer copyrighted the most efficient? Efficiency of prose is critically important for software. Computer copyright cases also make the distinction between literal prose and nonliteral software expressions.18 To avoid a multitude of categorical schisms, software must now be recognized as its own unique category of intellectual property with its own unified theory of protection, including detailed statutory definitions to illuminate this otherwise obscured field. The Software Act does so.

Courts have expressed the belief that applying copyright law to computer programs is like attempting to fit the "proverbial square peg into a round hole," since computer programs are simultaneously highly functional utilitarian objects comprised of literary expressions.19 The Second Circuit, in Computer Associates International, Inc. v. Altai, Inc.,20 found that the Copyright Act is not ideally suited to deal with the dynamic technology of computer science, because it

20. 982 F.2d 693.
serves as a relatively weak barrier against infringement of a program’s theoretical interstices. The Ninth Circuit, in Sega Enterprises Ltd. v. Accolade, Inc.,22 adopted with “wholehearted agreement” the Second Circuit’s reasoning above, finding that due to the essentially utilitarian hybrid nature of computer programs, no settled standard exists for distinguishing a protected expression from an unprotected idea in computer software.23 It embraced the Altai approach to breaking down a computer program into component subroutines and identifying the idea or core functional element of each.24 The Third Circuit, in Apple Computer, Inc. v. Franklin Computer Corp.,25 examined the copyright/patent gap and noted that copyright protects expression, while patent protects functionality.26 Therefore, two different expressions each performing the same function are equally copyrightable. Apple Computer could copyright the specific instructions controlling its Apple computer, but Franklin Computer would have been free to create a new, different set of instructions providing the same functionality to control its Apple compatible computer, the Pineapple (however in this case, Franklin did not write new code).27 In another case regarding Apple Corporation, Apple Computer, Inc. v. Formula International, Inc.,28 a California district court called for specialized legislation recognizing the unique dual art/utility nature of computer software:

It may be that the copyright law is not the most appropriate shelter under which to place this form of new technology and . . . a hybrid or entirely new form of protection will have to be devised. That, however, is up to Congress. This Court does not have the prerogative of investing a new form of protection even if it wanted to. To the extent it is free to express public policy, its choice is to place computer programs into an existing category of legal protection as against affording them no protection at all.29

The chief opponent to such a new statutory solution is Harvard Law School Professor Arthur Miller. Professor Miller served as a commissioner on the National Commission on New Technological Uses of Copyrighted Works (CONTU), which Congress created in 1974 to recommend new technology amendments to Copyright Act.

21. Id. at 712.
22. 977 F.2d 1510 (9th Cir. 1992).
23. Id. at 1524.
24. Id. at 1525.
25. 714 F.2d 1240 (3d Cir. 1983).
26. Id. at 816.
27. Apple v. Franklin, 714 F.2d at 1253.
Professor Miller argues against new software legislation because Congress, via CONTU, has already foreseen, considered, and rejected a large statutory overhaul for software.\(^{31}\) CONTU felt that the boundaries of copyright law had recurrently been expanded to engulf new technologies, such as, photographs, motion pictures, radio, television, photocopying, and telecommunications.\(^{32}\) Those boundaries were thought to be flexible enough to expand yet again to accommodate software.\(^{33}\) Therefore, CONTU's recommendations to Congress were minimal, suggesting merely a new definition for the term "computer program," as well as a provision permitting consumers to archive software and adapt software to be compatible with their hardware.\(^{34}\)

Congress accepted CONTU's modest recommendations and enacted them verbatim as the 1980 Amendments to the Copyright Act.\(^{35}\) Congress's decision is vindicated in hindsight by the flourishing success of the software industry since the 1980 amendments.

Congress amended the Copyright Act in 1980 in order to facilitate judicial clarification of the gaps in the statute with new theoretical tests of infringement. Even today the 1980 amendments continue to serve the purpose that Congress intended.\(^{36}\) Professor Miller argues that courts have proven skillful at devising an analytical framework to distinguish between copyrightable expression and uncopyrightable function, and are also competent in sifting cases through this matrix.\(^{37}\) He credits federal judges with making extraordinary efforts to understand new software technologies,\(^{38}\) concluding that the resulting case law has simplified copyright principles into "an understandable and sensible doctrinal matrix, obviating any need for a *sui generis* approach."\(^{39}\) Professor Miller aptly notes that today's statutory bright-line test could become tomorrow's Maginot Line, due to the highly fact-specific nature of software protection comparisons.\(^{40}\)

Yet Professor Miller's argument is logically inconsistent and not totally squared by the court decisions themselves. Logically, if

\(^{32}\) Id. at 982.
\(^{33}\) Id. at 981.
\(^{36}\) Miller, *supra* note 3, at 980-81, 1009 n.150.
\(^{37}\) Id. at 992, 998.
\(^{39}\) Miller, *supra* note 3, at 1035-36.
\(^{40}\) Id. at 1035.
software cases are so inherently ad hoc based upon their fact-specific nature, then how can any unified doctrinal matrix theory ever develop? Moreover, much of the Software Act proposed in this paper is primarily a codification of the doctrinal matrix that has already evolved in the courts. This should not be objectionable to the wait-and-see theory. Mere legislative approval of current case law should assuage courts’ fears of treading on unstable ground.

Finally, the wait-and-see theory is at odds with an increasing number of court decisions wherein the courts themselves are crying out for statutory guidance. In developing much of the case law to be codified, Judge Keeton saw the court as a surveyor examining markers and guideposts placed by Congress.41 The Second Circuit went even further than the district court above. The Second Circuit found the results of CONTU so ill-suited to computer software that it proposed a second round of legislative changes.42

The proposed Software Act fulfills the concerns above by requiring novelty and nonobviousness for protection. The Software Act also creates bright-line statutory markers that go beyond the doctrinal matrix devised by the courts. These markers should make a court’s determination more swift and sure, while clearing public confusion over what conduct constitutes infringement. Nor does the Software Act deprive courts of the discretion that Professor Miller urges is needed to resolve highly fact-specific cases. The Software Act bolsters a court’s ability to fashion stronger decisions based upon a new statutory frame-

41. Bemoaning a lack of statutory “markers” defining software infringement, Judge Keeton saw the court’s responsibility this way:

No marker Congress has placed may be disregarded or relocated by courts. Even in those instances where text and context make clear that literal description of a marker is contrary to manifested meaning (as where “not” must be inserted or deleted to make sense of the statutory language) what the court is doing is aptly described as “locating,” not “relocating,” the marker Congress mandated. The fewer the markers Congress has placed, the more critical it becomes that courts assure that no marker escapes notice.


42. Generally, we think that copyright registration—with its indiscriminating availability—is not ideally suited to deal with the highly dynamic technology of computer science. Thus far, many of the decisions in this area reflect the courts’ attempt to fit the proverbial square peg in a round hole . . . . In any event, now that more than 12 years have passed since CONTU issued its final report, the resolution of this specific issue could benefit from further legislative investigation—perhaps a CONTU II.

work. The presence of patent eligible subject matter must always be determined upon the individual facts of each case.\textsuperscript{43}

This approach is in keeping with the widely accepted determination of Second Circuit Judge Learned Hand that drawing the line between nonprotected idea and protected expression is a tricky business: "Nobody has ever been able to fix that boundary, and nobody ever can."\textsuperscript{44} Judge Hand went on to determine that, "Obviously, no principle can be stated as to when an imitator has gone beyond copying the 'idea,' and has borrowed its 'expression.' 'Decisions must therefore inevitably be \textit{ad hoc}.'\textsuperscript{45} The imprecise standard allowing a court to search between the lines of a complaint for the scent of meritorious claims is sometimes referred to as the "smell test."\textsuperscript{46} Under the \textit{Software Act}, a judge retains the flexibility to determine whether something smells rotten, and thereafter fashion appropriate relief via this statute.

Ironically, Professor Miller also faults courts for applying too much \textit{ad hoc} discretion when they create a new reverse engineering fair use exception to infringement.\textsuperscript{47} First, he argues that such an exception is in apparent contradiction the Congressional intent displayed in adopting CONTU's recommendations opposing reverse engineering.\textsuperscript{48} Additionally, he eschews new statutory boundaries, which might constrain courts from such context-driven, highly fact-specific based expansions.\textsuperscript{49} Yet, Professor Miller is critical of courts that fail to restrict themselves to Congress' implied statutory restrictions.\textsuperscript{50}

---


\textsuperscript{44.} Nichols v. Universal Pictures Co., 45 F.2d 119, 121 (2d Cir. 1930) (L. Hand, J.), \textit{cert. denied}, 282 U.S. 902 (1931).

\textsuperscript{45.} Peter Pan Fabrics, Inc. v. Martin Weiner Corp., 274 F.2d 487, 489 (2d Cir. 1960) (L. Hand, J.).

\textsuperscript{46.} Morgan Fiduciary, Ltd. v. Citizens & Southern Int'l Bank, 95 B.R. 232, 234 (S.D.Fla.1988) (paraphrasing Irwin Younger) ("Any trial judge will inevitably come to the conclusion on occasion that a certain case or claim or defense has a bad odor. Simply put, a matter smells. Some smell so bad they stink.").

\textsuperscript{47.} \textit{See infra} part VI.D (discussing Sega Enters. Ltd. v. Accolade, Inc., 977 F.2d 1510 (9th Cir. 1993)).

\textsuperscript{48.} Miller, \textit{supra} note 3, at 1013-32.


\textsuperscript{50.} Miller, \textit{supra} note 3, at 1035.
Professor Miller’s views also discount the time and effort lost in sidetracking eddies of case law, such as, the now often rejected structure-sequence-organization analysis found in Whelan Associates v. Jaslow Dental Laboratory, Inc. The Software Act revives the essence of the doctrine, while avoiding its pitfalls, by making the applicant specifically enumerate for the court which structure-sequence-organization elements he seeks to protect.

C. Need for Harmonization with Other Countries

The United States must seek international solutions to its domestic intellectual property problems, given the rapidly increasing importance of intellectual property on worldwide trade, national identity, and economic well-being of nations. Harmonization would provide U.S. attorneys with greater expertise in protecting their U.S. clients’ interests abroad, since the attorneys would need to learn only a single legal system of protection. The Software Act is a step toward achieving such overall intellectual property law harmonization. The Software Act advances the United States toward the patent system followed by the rest of the world, the first-to-file system.

Scholars in the United States have cheered the recent TRIPs agreement as a giant step forward for protection of domestic intellectual property abroad, since it forces other countries to protect computer programs. TRIPs requires member countries to treat computer programs for copyright purposes as literary works, like books. Such cheering is misplaced. Although copyright protection is better than no protection, copyright is as much the wrong law abroad as it is domestic.

53. Ralph Oman, Berne Revision: The Continuing Drama, 4 FORDHAM INTELL. PROP. MEDIA & ENT. L.J. 139, 142 (1993) (suggesting that the United States proposed TRIPs is a "Berne Plus" package requiring GATT members to recognize all of the economic rights guaranteed by the Berne Convention).
54. Teresa Riodan, Patents; An Outspoken Inventor Protests Efforts to 'Harmonize' Global Rules, N.Y. Times, Nov. 29, 1993, at D2. Only the United States, Jordan and the Philippines use the first-to-invent system. Id.
55. General Agreement on Tariffs and Trade – Multilateral Negotiations (the Uruguay Round: Agreement on Trade-Related Aspects of Intellectual Property Rights, April 15, 1994, 33 I.L.M. 81 [hereinafter TRIPs]; see also Oman, supra note 53.
56. See Oman, supra note 53; see also Monique L. Cordray, GATT v. WIPO, 76 J. PAT. & TRADEMARK OFF. SOC’y 121 (1994).
57. See Oman, supra note 53; Riodan, supra note 54.
cally to protect software. The TRIPs agreement was achieved by the U.S.'s strong-arm technique of dangling trade concession carrots in exchange for software protection. However, loopholes within TRIPs allow foreign countries to delay for years the implementation of what was achieved. Even under the century-old Berne agreement, fewer than half of the Berne signatory nations actually provide adequate copyright protection, even though their regulations technically comply with the agreement. By marrying TRIPs compliance to Berne compliance, the TRIPs achievements might be equally toothless.

The Software Act provides a better solution by providing lesser-developed countries (LCDs) a real incentive for software protection, since it gives them easier access to software licenses at a reasonable fee. Furthermore, foreign countries might actually be more comfortable enforcing the Software Act than they are enforcing laws crammed down their throats, since these countries already have similar petite patent acts for other nonsoftware fields.

58. Cordray, supra note 56, at 143 (stating that the Uruguay Round package deal gave the United States the bargaining power to strong arm countries willing to make concessions in the area of intellectual property for gains in other areas, such as, agriculture and textiles).

59. TRIPs, supra note 55, art. 66 (generally allowing countries 10 years to amend their domestic laws to be consistent with the new TRIPs requirements, while it further allows LDCs an indefinite extension).


62. TRIPs, supra note 55, art. 9; cf. TRIPs, art. 2 (requiring compliance with the Berne Convention).

63. See infra Appendix §§ 236-37.

64. The Software Act would be consistent with what already exists abroad. For example, the German Utility Model Act of 1968, protects inventive concepts related to implements or commodities. Feidrich-Karl Beir, et al., German Industrial Property, Copyright and Antitrust Laws, Vol 6. STUDIES IN INDUSTRIAL PROPERTY AND COPYRIGHT LAW (Max Planck Institute 1983). A computer program is like an implement. Since it is essentially valueless in and of itself, its only value is for what it does. The German Utility Model petty patent confers short-term, simple, and inexpensive protection for smaller technical inventions. It is conferred upon registration, without substantive examination, lasting for three years from application date, plus a possible three-year extension. German petty patents grant the same rights as a regular patent. However, they are enforceable only against third parties. Examination of the German petty patent takes place only upon an infringement or cancellation proceeding. The examination must find a small, new, commercially-applicable inventive step, not already the subject of a prior patent – that has not been misappropriated from a third party. Id. The Software Act mirrors most of these German petite patent act requirements.
III. SOFTWARE AS COPYRIGHTABLE MATERIAL

A. Software as Art

1. Software's Dual-Natured Literal and Nonliteral Expressiveness

Software's nature is intrinsically a hybrid combining both text and action. Software's text is its literal expression, while software's action is its nonliteral expression. Courts have found both forms of expression to be copyrightable. Software text consists of a complex set of instructions written in a formal computer language. The Third Circuit, in *Apple Computer v. Franklin Computer*, held that both source code and object code are protected as literary works. The Copyright Act's definition of "literary works" includes expression not only in words but also "numbers, or other . . . numerical symbols or indicia." Ironically, the typical software customer never sees the literal expression that the author has diligently provided in the purchased software. The process of compiling boils away human-readable expression by converting source code into its machine-readable object code equivalent. Virtually all mass-distributed software is distributed in object code form. Hence, the expressive textual distinctiveness is filtered out before the customer ever comes in contact with the software.

Computer programs also have inherent nonliteral aspects, which make them different from ordinary works of literature. Computer programs take kinetic action when electricity is applied (not unlike activating the sounds and images captured in a sound recording or motion pictures). Yet, computer programs go beyond mere replication of prior performance. They interact with and adapt to the environment around them and perform differently depending upon changing circumstances. Software literature's self-actuation feature distinguishes it from conventional literary writing.

---

66. *Id.* at 1249.
67. Object code binary language consists entirely of the numerals 0 and 1.
68. A compiler is defined as "a program designed to translate a high level language source program into a corresponding machine code program. The compiler checks for, and reports, any syntax errors in the source program. If the source program is syntax error free, then a complete object code program is produced." Dennis Longley & Michael Shain, *DICTIONARY OF INFORMATION TECHNOLOGY* 64 (2d ed. 1986).
Software's duality of expression is referred to as the "literary/nonliterary" dichotomy. Not only is the text expressive in the way it is written, but what the text does attains its own expressiveness. What is particularly ironic about the recent judicial acceptance of this dichotomy is that the average user is interested only in the nonliteral elements expressed by the software, i.e., the on-screen displays with which he or she can interact. Nonliteral software expressions are often as stylistically characteristic of their authors as is the text.  

2. Judicial Acceptance of the Notion of Duality of Expression

Courts have recognized the twin modes of literal and the nonliteral software expression. Courts have granted separate copyright protection to nonliteral elements, such as, a program's overall organization, the structure of its command system, and its presentation of information on the screen. However, Congress has not explicitly recognized nonliteral software elements as copyrightable. The 1980 amendment to the Copyright Act defines "computer program" as the literal elements, rather than nonliteral elements. There is no question, however, that the music, pictorial, graphic, audiovisual, or other expressions can be copyrighted to the extent they can be identified separately from the software. The legislative history of § 102 clearly shows that it is intended to be a suggestive rather than an exhaustive list. Courts have occasionally anticipated Congress adding

70. For example, programs written by Peter Norton (e.g., Norton's Utilities) are well known for their powerful user friendliness in performing different systems maintenance tasks. A single isolated program from this series might be recognizable on sight as a "Norton" merely from its manner of display screens and method of operation.


73. 17 U.S.C. § 101 ("a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result.").


75. See Mazer v. Stein, 347 U.S. 201 (1954) (holding that the artist's ballerina shape of a lamp base can be copyrighted because it is expressively separable from the noncopyrightable, possibly patentable, useful object).

76. The use of the word "include," as defined in Section 101, makes clear that the listing is "illustrative and not limitative," and that the seven categories do not exhaust the scope of "original works of authorship" that the bill is intended to protect. Rather, the list sets out the general area of copyrightable subject matter, but with sufficient flexibility to free the courts from rigid or outdated concepts of the scope of particular categories.

Notes of the Committee on the Judiciary, House Report No. 94-1476 (reprinted in 17 U.S.C.A. § 102 at 47, 49 (West 1996)).
new technology and media to the list before Congress expressly does so.\textsuperscript{77} It seems likely that the courts are empowered to extend copyright to nonliteral software expression that can be separately identified from the software itself.\textsuperscript{78}

The current statutory dichotomy between useful articles\textsuperscript{79} and copyrightable expression is not a helpful distinction for computer programs. While it may be that some arrangements or sequences of text or graphics in the user interface may make the software easier to use than others, the ultimate goal is usefulness. As Judge Keeton pointed out:

> It does not follow that when an intellectual work achieves the feat of being useful as well as expressive and original, the moment of creative triumph is also a moment of devastating financial loss—because the triumph destroys copyrightability of all expressive elements which would have been protected if only they had not contributed so much to the public interest by helping to make some article useful.\textsuperscript{80}

The law should not require products to fall short of being the best possible to achieve protection. This can hardly be the system that Thomas Jefferson envisioned as our first patent Commissioner when he authored the 1793 Patent Act.\textsuperscript{81}

B. Why Copyright Law Does Not Protect Software Well

1. Derivative Works

In addition to utility limitations, copyright law is also difficult to apply to software because of the difficulty in determining what a derivative work of software is. Section 106(2), provides that only the copyright owner may "prepare derivative works based upon the copyrighted work."\textsuperscript{82} Section 101 defines a "derivative work" as "a work based upon one or more preexisting works, such as, a translation, motion picture version, sound recording, abridgment, condensation, or

\begin{itemize}
  \item \textsuperscript{77} E.g., Burrow-Giles Lithographic v. Sarony, 111 U.S. 53 (1884) (finding that photographs were "writings" before Congress codified this in Copyright Act of 1909 § 5(j)).
  \item \textsuperscript{78} Compare this to the contortions the courts have been forced to perform to squeeze software algorithms within the Patent Act § 101 terms "process" and "machine," since Patent Act § 101 is considered to be an exhaustive list of protectable subject matter. The courts cannot simply invent a term like "nonliteral process" or "nonliteral machine" to encompass software algorithms.
  \item \textsuperscript{79} 17 U.S.C. § 101 ("an article having intrinsic utilitarian function that is not merely to portray the appearance of the article or to convey information.").
  \item \textsuperscript{81} Diamond v. Chakrabarty, 447 U.S. 303, 308 (1980).
  \item \textsuperscript{82} 17 U.S.C. § 106(2).
\end{itemize}
any other form in which a work may be adapted." It is easy to determine that a motion picture *The Bridges of Madison County* is a derivative work of the book. Is a program a derivative work when it is designed to work alongside another program? Is a program a derivative work if it adapts software for a new piece of hardware? In *Whelan Associates v. Jaslow Dental Laboratory*, the district court clarified that translating one computer language into another is not comparable to translation from English to French, because it is not generally possible to translate computer code into a line-by-line duplicate in another language. This is due to the different methods and manners in which each computer language operates. Many computer languages are structurally dissimilar, so that accomplishing the same algorithm in each requires wholly dissimilar instruction patterns. However, many computer languages are similar, just as some human languages are sentence-by-sentence similar to English.

The complication with software is that all software is comprised of algorithms. Generally, the same algorithm can be written in a number of different source code and object code implementations. Unlike book-to-movie transformations, software derivative works generally result in the same medium — more original software code. In some respects, every piece of software is derived from some other software. Developers of later versions of software products benefit greatly from the insights and mistakes of earlier programmers. Judge Keeton, in *Lotus Development Corp. v. Paperback Software International*, noted the virtually unchallenged premise of intellectual property law is that progress of science and useful arts can only occur where authors and inventors are privileged to build upon the earlier progress of others: "If I have seen further it is by standing on ye shoulders of Giants." Modern software adopts cooperative techniques, such as, object-oriented programming and object-linking-and-embedding. This gives today’s software the capability of actually wrapping itself around the other software’s literal code.

85. Id. at 1320.
86. Id. at 1320-21.
88. Id. at 77 (quoting Sir Isaac Newton, Letter to Robert Hooke, February 5, 1675/1676, as quoted in ROBERT K. MERTON, ON THE SHOULDERS OF GIANTS: A SHANDEAN POSTSCRIPT 31 (1965)).
The concept of derivative work infringement is also difficult to apply to software because software is commonly released in updated versions over time. Users continually demand more powerful features as they become accustomed to the software and they envision new uses for the software. Today there are multiple new types of hardware that did not exist ten or fifteen years ago. Software must be updated to know how to operate the new hardware. The only finished software (nonderivative software) in existence is dead software (i.e., no one uses it). Legislation is not needed to protect obsolete software at the expense of new useful derivative software.

2. Copyright’s Duration is Excessively Long for Software

Current U.S. copyright protection generally lasts for a term of the life of the author plus fifty years. This is an excessive period of time to protect software. Even the twenty-year period of protection for algorithms under the current Patent Act is inappropriately long for software. In the fast-paced world of software, even the shorter pre-GATT patent term of seventeen years “might as well be a millennium.” Therefore, the Software Act protects software for only ten years. Arguably, the petite protection could be shorter. However, since many algorithms are reused, and since much or the older versions of software continue to be used long after they are no longer sold, ten years seems like a reasonable period to earn royalties to compensate for high development costs.

94. See infra Appendix § 230. This term is analogous to the Semiconductor Chip Protection Act’s 10-year duration of protection commencing on the date of registration or date of commercial exploitation, whichever occurs first. 17 U.S.C. § 904. It also harmonizes with the Draft European Community Directive 10-year protection against unauthorized data extraction. MICHAEL A. EPSTEIN et al., INTERNATIONAL INTELLECTUAL PROPERTY: THE EUROPEAN COMMUNITY AND EASTERN EUROPE 3-11 (1993). Under Article 9-1, a database itself gets the same protection period as provided by the member country’s copyright for literary works. Id. at 3-89. However, under Article 9.3 an additional sui generis Unfair Extraction Right arises upon creation of the database, continuing 10 years from January 1st of the year following the date the database is first lawfully made available to the public. Id. at 3-90. Unfair Extraction is defined by Article 2.5 as “unauthorized extraction or reutilization, from that database, of its contents, in whole or in substantial part, for commercial purposes.” Id. at 3-85.
IV. SOFTWARE AS A USEFUL OBJECT (PATENTABLE MATERIAL)

A. What is Patentable?

The Patent Act protects any "process, machine, manufacture, or composition of matter" or improvement thereof, that is novel and non-obvious. An invention must pass each of the above three tests, or doors, to receive patent protection. Anything not included in the §101 list of "process, machine, manufacture, or composition of matter" is simply not eligible for patent, since inclusion of these things in the list necessarily excludes everything else. Because of the unique nature of software, the availability of patent protection for software has historically been uncertain.

The preliminary question for software patent protection must be, what is software? Is it a process, a machine, or is software merely an unpatentable idea? Article I, § 8, cl. 8 (the constitutional command to "promote the Progress of ... useful Arts") is both a grant and limitation of power. Congress may not grant patents that inhibit (rather than promote) the progress of the useful arts. A patent on an idea would inhibit the useful arts because it would "remove existent knowledge from the public domain, or restrict free access to materials already available." Yet, society chooses for its own benefit to grant unnatural patent rights on nonidea items, as an economic incentive to develop and disclose these useful applications of ideas.

B. What Part of Software is Patentable?

1. Patentable Algorithms

In Gottschalk v. Benson, the Supreme Court held that a patent claim for a method of programming a general purpose digital computer is a claim for an algorithm. The Court defined an "algorithm" as "[a] procedure for solving a given type of mathematical problem." An algorithm is more comprehensively defined by computer scientists...
as: "a finite set of well defined rules for the solution of a problem in a
finite number of steps, for example, a precise description of the steps
involved in determining the record with the highest value of a speci-
fied numerical attribute." Algorithms may be thought of as the part
of software containing its primary economic value, other than the
value attributed to the specific words comprising the program or its
user interface.

2. Algorithms Are the Same as Nonliteral Elements in
   Copyright Law

With a regular patent, the description of the process is separable
from the process itself. A claimant's mere written instructions
descrying a new way to tag deoxyribonucleic acid (DNA), for ex-
ample, do not actually produce any tagged DNA. On the other hand, a
written claim for an algorithm can be expressed either as a literal or
nonliteral expression of the algorithm. The literal description of the
algorithm is the algorithm itself. Hence, the mere literal description of
a software process is the process.

For example, the line of Apple Computer, Inc. v. Microsoft
Corp. (and Hewlett-Packard (HP)) infringement cases also could
have been analyzed as algorithm cases. The nonliteral elements at is-
ue were the shapes and tasks used to open, close, and move windows
and graphically representative icons. Apple's garbage-can deletion al-
grithm might survive the filtration step. Hewlett-Packard's chal-
enged wastebasket algorithm might be held substantially similar
under the comparison step. Thus, HP might be required to pay a li-

107. For example, a literal algorithm to switch the contents of two variables could be
claimed in code written like this:

10 LET C = A
20 LET A = B
30 LET B = C

The same algorithm could also be claimed nonliterally as series of narrative instructions, "take a
third variable and replace its contents with the value of the first variable, then replace the value
of the first variable with the value of the second variable, then replace the value of the second
variable with the third variable." Narrative expression of an algorithm can be a nonliteral ex-
pession of the literal code. George Ledin, Jr., A STRUCTURED APPROACH TO GENERAL BASIC
57 (1978).

108. See Apple Computer, Inc. v. Microsoft Corp., 709 F. Supp. 925 (N.D. Cal. Mar. 20,
616 (N.D. Cal. May 18, 1993), aff'd by 35 F.3d 1435 (9th Cir. Sep. 19, 1994), cert. denied 115
S.Ct. 1176 (Feb. 21, 1995); see also Apple Computer, Inc. v. Microsoft Corp., 1992 WL 75423,
109. See infra Appendix § 233.
censing fee for the algorithm for a short time. Analyzed under the Copyright Act, HP's indoor kitchen-style flip-top wastebasket was held to have a dissimilar artistic appearance, and thus was not a copy of Apple's outdoor alley-style cylindrical garbage can. The Software Act, however, might ignore the slight difference in appearance and focus instead on the precise series of steps taken to perform the same function. However, if HP were to demonstrate that its algorithm required a step of clicking a mouse twice, rather than Apple's once, this step could be dissimilar enough functionally that the same conclusion of noninfringement might result under the Software Act.

C. Does a Software Algorithm Qualify as a Patent Act Process?

1. Supreme Court Definition of "Process"

The Court has held that software algorithms can be included as part of a Patent Act § 101 process, but by themselves they do not constitute an eligible process. The term "process" is a term of art with a meaning different from the standard dictionary meaning. The Patent Act provides the circular definition that, "the term 'process' means process, art or method, and includes a new use of a known process, machine, manufacture, composition of matter, or material." Thus, the meaning of "process" must be divined from case law.

The Court, in Gottschalk v. Benson, ruled that the term "process" does not include purely mathematical digital computer software algorithms. The term "process" traditionally requires physical transformation of matter from one physical state into another physical state. A process does not include mathematical formulas, because they merely transform one nonphysical form of numbers into a different nonphysical form of numbers.

In Dann v. Johnston, the government's brief suggests that Benson may be overcome by showing software that synergizes with computer hardware in a novel and nonobvious way in which "the whole in some way exceeds the sum of the parts." Where an algorithm is realistically unsolvable without a computer, and where no computer

112. 409 U.S. 63 (1972).
113. Id. at 71-72.
114. Id. at 71.
115. See e.g., Gottschalk v. Benson, 409 U.S. 63 (1972) (finding the conversion of binary-coded decimal into pure binary to be a mathematical formula not a §101 'process').
had been able to solve this type of problem without this algorithm, then the two together become more than the sum of their parts.\textsuperscript{118} The counterargument is that programs do not synergize when they simply do what the hardware allows them to do, and where the hardware simply does the job it was instructed to do, without any surprising or unexpected result.\textsuperscript{119}

The Court, in \textit{Parker v. Flook},\textsuperscript{120} found algorithms to be patentable as part of a process, but "[t]he process itself, not merely the mathematical algorithm, must be new and useful."\textsuperscript{121} Moreover, laws of nature, natural phenomena, and mathematical formulas by themselves are not the "\textit{kind} of 'discoveries'" that the Patent Act was designed to protect.\textsuperscript{122} Merely moving an algorithm to the middle of the process by tacking on some post-solution activity does not make the algorithm patentable.\textsuperscript{123} So, a new and useful process applying a well-known algorithm may be patentable, while a new and useful algorithm applying a well-known process is not. However, it may be a close call whether the process is new and useful or whether the algorithm is new and useful. The Court feared that a patent on a mathematical formula would wholly preempt use of that formula and effectively remove it from the public domain, even if the applicant limited his claim to the particular use he was making of it.\textsuperscript{124}

The \textit{Software Act} overrides \textit{Flook}'s mathematical formula rule by resolving its concern over preemption. No one will be able to wholly preempt a field where compulsory licensing allows anyone to enter the field by paying the first comer a reasonable price; the first comer is compensated for his initial investment of research and development.\textsuperscript{125} The compulsory licensing fee provides the first comer the needed incentive to perform the initial expensive research.

The Court, in \textit{Diamond v. Diehr},\textsuperscript{126} modified \textit{Benson} and \textit{Flook} by holding that the claims should be considered as a whole to determine whether the algorithm performs a traditionally patentable manu-

\textsuperscript{118} \textit{Id.}
\textsuperscript{119} \textit{In re Freeman, 573 F.2d 1237, 1243 n.2 (C.C.P.A. 1978).}
\textsuperscript{120} 437 U.S. 584 (1978).
\textsuperscript{121} \textit{Id.} at 591 (emphasis added).
\textsuperscript{122} \textit{Id.} at 593 (emphasis added).
\textsuperscript{123} \textit{Id.} The algorithm in \textit{Flook} calculated alarm limits from time, temperature, pressure, and flow rate variables, and then tacked on the activity of applying those calculated limits. \textit{Id.} at 585.
\textsuperscript{124} 437 U.S. at 590.
\textsuperscript{125} \textit{See infra} Appendix § 236.
\textsuperscript{126} \textit{Diamond v. Diehr, 450 U.S. 175 (1981).}
facturing process. In Diehr, the manufacturing process transformed raw rubber granules into a baked solid mass, using an algorithm to continuously measure temperature and calculate the remaining baking time needed without overcooking or undercooking.

Many software algorithms fail as Diehr patentable processes because they do not make physical changes to raw materials. Instead, they merely change a transistor from a negative state to a positive charge (a "0" to a "1"). Many algorithms merely speed up data processing in a computer or are only practical when used in a computer.

2. Federal Circuit Court’s Special Relationship to Software Algorithms

The Federal Courts Improvement Act of 1982 expressly created the Circuit Court for the Federal Circuit ("Federal Circuit") to speak as a nationwide voice filling gaps left by Supreme Court interpretations of patent (and specific other) issues. Congress realized that the Supreme Court was operating at, or close to, full capacity. Therefore, the Supreme Court’s tie-breaking function was deliberately reduced on patent issues by making the intermediate appellate level Federal Circuit a junior supreme court that eliminates inconsistent decisions among the circuits. Congress replaced intermediate appellate level courts of narrow jurisdiction with a single court providing uniform answers to questions specifically enumerated by Congress.

The Federal Circuit differs from its sister federal courts due to its exclusive jurisdiction defined in subject matter rather than geographic terms.

3. RAM/ROM Exemplifies "Process/Machine" Definitional Conundrum

Lacking a detailed statute to guide the Federal Circuit in its determination of which algorithms are protectable subject matter under the Patent Act has forced the court to take erratic positions in different
cases with similar facts. The Patent Act states processes and machines may be claimed for patent. In In re Iwahashi, Judge Rich, writing for two of a three-judge panel, found an algorithm for voice recognition was a patentable "machine" or "manufacture" under § 101, since it depended upon numbers permanently stored in ROM (read only memory). Because ROM is physical hardware, thus "a specific piece of apparatus," it is a machine. However, five years later, in In re Alappat, Judge Rich, writing the majority in an en banc court, wavered from his earlier stance and found:

The Iwahashi court clearly did not find patentable subject matter merely because a ROM was recited in the claim at issue; rather the court held that the claim as a whole, directed to the combination of the claimed means elements, including the claimed ROM as one element, was directed to statutory subject matter. It was not the ROM alone that carried the day.

What makes this turnabout unconvincing is that hardware and software are logically equivalent. Any software operation can be built directly as hardware, while any hardware instruction can be simulated by software. This logical inconsistency of making a ROM algorithm protectable and a RAM algorithm unprotectable, exacerbated by the court's reversal, might lead one to question whether the Federal Circuit has indeed provided the certainty and consistency sought by the Federal Court Improvement Act of 1982 in hearing all patent appeals under one roof. Yet, it seems likely that the problem lies not with the structure of the court, but rather with the structure of an outdated statute.

The Software Act would assist courts in fashioning concrete distinctions of patentability by hanging them on a more detailed statutory frame. It would eliminate the bothersome conundrum between non-physical mathematical algorithms and physical/hardware algorithms. Furthermore, one of its goals is to protect all software algorithms, mathematical or not, with a weaker shorter-term protection, rather than the full-strength, long-term protection of a regular patent. Finally, the Software Act addresses the Supreme Court's constitutional concern regarding the Writing Clause's prohibition against removing ideas from the public domain. Any software ideas that the Software Act might protect would not already be in the public domain (hence

134. Id. at 1375.
135. Id.
136. In re Alappat, 33 F.3d 1526 (Fed. Cir. 1994).
137. Id. at 1544 n.24.
they could not be removed), and anyone is allowed full access to the patented algorithms for commercial use upon payment of a reasonable fee (while no fee is required for noncommercial use).

4. The Federal Circuit Court's Definition of "Process"

The Federal Circuit elaborated upon the Supreme Court's definition of "process" by finding in In re Schrader that a claim is not a protectable process if it mentions only mathematical algorithms and fails to refer to physical acts of transforming materials into a different state. The court in Schrader advises that the applicant must state that physical effect in the claims where an algorithm does create a physical transformation. Otherwise, any post-solution activity could transform the otherwise unpatentable process into a patentable process, thereby sanctioning form over substance. The so-called Freeman-Walter-Abele two-step test determines compliance of both process and machine claims with § 101. The first step is to determine whether a mathematical algorithm is recited directly or indirectly in the claim. If so, then the second step is to determine whether the invention as a whole is more than an algorithm itself. In In re Warmerdam, the Federal Circuit found that a claim as a whole is for nothing more than an algorithm itself where it fails to claim any physical activity or limitations beyond the mere manipulation of data. Thus, under the current law, mathematical algorithms fail the Freeman-Walter-Abele test where the algorithms are not applied to, or limited by, physical elements or process steps.

These decisions seem to open the door for claims of a process wherein a programmed machine has its physical memory chips transformed by the algorithm or process from one configuration of particular data and mathematical constructs into another. Yet, one must ask whether a computer memory chip altering its physical state from a

139. Id. at 293-94.
140. Id.; see also In re Abele, 684 F.2d 902, 908-09 (C.C.P.A. 1982) (concluding that while merely displaying the result of a calculation is not a sufficient physical transformation, use of an algorithm as part of a process of displaying data may be patentable as a part of that process).
141. Schrader, 22 F.3d at 294 (quoting Parker v. Flook, 437 U.S. 584, 590 (1978)) ("The concept of patentable subject matter under § 101 is not "like a nose of wax which may be turned and twisted in any direction. . . ."").
142. Id. at 292.
143. 33 F.3d 1354.
144. Id. at 1360 (sustained rejection of "bubble hierarchy" claims (i.e., collision avoidance models enclosing objects inside imaginary mathematical spheres) because claims did not refer to physical activity).
145. Id. at 1361.
"1" to a "0," or a positive to a negative charge, is really the sort of traditional physical transformation of raw material envisioned by the Supreme Court. Is there really any physical transformation when the tiny transistors in the chips can change back and forth from "1" to "0" thousands of times per second? Conceptually, the software process is a game of juggling algebraic variables that measure real-world physical objects, but which themselves are not physical objects, no matter how the claims are redrafted to try to meet the court definition of process.

D. Does a Software Algorithm Qualify as a Patent Act §101 "Machine?"

Since the Supreme Court has effectively narrowed the meaning of the term "process" in § 101 (despite its insistence that it has not done so), the Federal Circuit has demonstrated a willingness to broaden the meaning of the coequal term "machine." The difference between the terms "process" and "machine" is at best illusory, since any competent draftsman can readily convert from one form to another. Draftsmanship can make an invention either a method or an apparatus.

In 1969, the Court of Customs and Patent Appeals found in In re Prater that there is no constitutional, statutory, or case law support for the proposition that a "programmed general-purpose digital computer [is] necessarily unpatentable." This novel approach likens a general-purpose computer to a storeroom of electrical components. The components are taken by a computer program out of the storeroom as needed, connected together, and a new machine built from those parts. This theory was reiterated in In re Noll, where a programmed machine was found to be different from a machine without the program. Physical electrical circuits configured the machine’s physical storage devices and electrical components to achieve con-

147. E.g., In re Alappat, 33 F.3d 1526 (Fed. Cir. 1994).
148. See Richard Stem, Tales from the Algorithm War: Benson to Iwahashi, It's Deja Vu All Over Again, 18 AIPLA Q.J. 371, 377-78 (1991). Process format and means-for format are isomorphic. The algorithm to making a peanut-butter sandwich can be expressed either as a "process" or a "machine." Id. at 377. The difference between an "idea" and a "process" or "machine" may be even more tenuous, given the generally intangible nature of software.
151. Id. at 1403 n.29.
trolled results. Preemption was avoided because the patent application excluded claims for hardware machines achieving the same results without software.

In *In re Alappat*, the Federal Circuit adopted the theory set forth by the Court of Customs and Patent Appeals. Therein, the court held that a claim reading on a general purpose computer to carry out its invention was in effect claiming a new machine created by the instructions from the program software embodying the invention. Since the computer implementing the claim was an apparatus, not mathematics, it was a machine, not a process. The result in *Alappat* sends a clear message to the Patent and Trademark Office that claims drafted in apparatus form are very likely to meet the machine requirement, even if read to cover a general purpose computer consisting entirely of old hardware elements. However, pursuant to *In re Trovato*, the claimant must be certain to disclose at least a specific hardware embodiment in the specification, such as, *Alappat*'s "arithmetic logic circuits, barrel shifters, and a read only memory." Otherwise, where an application claims only software instructions without tying them to a specification disclosing a structure physically manipulating something in the real world, then the application will be rejected as "drafted in an illusory apparatus format." In other words, the Federal Circuit slapped Trovato on the wrist, giving her (and the world) the precise corrective wording needed to redraft any algorithm as a machine.

The Federal Circuit's interpretation of the term "machine" is tortured at best. While it may be true that a computer program electronically connects the physical components differently from a computer absent such program, it is not true that two computers containing the same program will have components configured exactly same way.

---

153. Id.
154. Id.
155. 33 F.3d 1526 (Fed. Cir. 1994).
156. The Federal Circuit has adopted all of the holdings of its predecessor court, the C.C.P.A. See South Corp. and Seal Fleet, Inc. v. U.S., 690 F.2d 1368 (Fed. Cir. 1982) (en banc) ("We hold that the holdings of our predecessor courts, the U.S. Court of Claims and the U.S. Court of Customs and Patent Appeals, announced by those courts before the close of business September 30, 1982, shall be binding precedent in this court.").
157. *In re Alappat*, 33 F.3d 1526, 1544 (Fed. Cir. 1994).
160. 42 F.3d at 1383 (citing *In re Alappat*, 33 F.3d 1526, 1541 (Fed. Cir. 1994) (en banc)).
161. Id. (citing Application of Walter, 618 F.2d 758, 769 (C.C.P.A. 1980)).
By deliberate design, software may not create exactly the same machine each time, because the program must cooperate with other software and hardware also sharing the same storehouse of computer components. Frequently, the programmer would have no ability to make the software operate identically each time, even if the programmer had wanted to do so. A programmer will not know exactly where the input files, output files, and temporary working files are physically located on a disk, because it is the software's job to determine such physical layout. The software may not use exactly the same RAM components each time, may not display the same results on the monitor each time, and may even utilize the CPU component quite differently to perform calculations. Even an identical program will create very different machines when operating on an IBM-PC, Macintosh, or parallel/vector computer, for the simple reason that each type of computer contains a unique storeroom of components.

E. Why Patent Law Does Not Protect Software Well

1. Current Patent Protection is Both Too Narrow and Too Broad

The present Patent Act is both too narrow and too broad to adequately protect software and algorithms. Regular patents simply do not fit the basic nature of algorithms and software. The apparent problem is that the current text of the Patent Act is too narrow to create meaningful distinctions between what is patentable and what is not. The Federal Circuit is making valiant attempts to shoehorn or stretch algorithms to fit somewhere between the words "process" and "machine."\(^\text{162}\) While the court's job could be made much easier with a statutory change actually adding the word "algorithm" to the §101 list, this alone would not solve all of the court's problems. Many more definitions need to be added to create a statute upon which the courts can rely.

Even if these changes were made, the Patent Act is still ill-suited to protect software because its protection is far too broad. The seventeen-year length of patent protection (now twenty years under GATT) far exceeds the useful life span of most software and many algorithms.\(^\text{163}\) Likewise, the Patent Act's breadth of protection is designed to provide total exclusion of use throughout the United States.\(^\text{164}\) This is unrealistic for software and algorithms due to independent reinven-

\(^{162}\) See supra parts IV.C-D.


tion, the necessity for both hardware and software compatibility, and intermodular software – where the user may not even know that the software or algorithm is in use.

Furthermore, the time and cost of obtaining a patent are far too high. A simple patent search can take more time than developing the software.\textsuperscript{165} Prior art searches are difficult for software because many innovations are never published.\textsuperscript{166} The Software Act proposes a solution to the searching problem by making a search unnecessary. An applicant can file a standard application form, rather than a current application of complex patent claims. Applications are not examined in detail until a litigation conflict develops in the future.\textsuperscript{167} Where conflict develops, a court may invalidate software petite patents issued to everyone after the first-to-file. Second comers need only pay a reasonable compulsory licensing fee to the winner, or they may negotiate a lower direct licensing amount.

2. Modern Software Breaks All the Rules

The Federal Circuit’s standard that algorithms are processes or machines hopelessly breaks down when applied to tomorrow’s generation of software. A new generation of products has been announced that allows the user to build compound applications by literally dragging and dropping modules of one application into another.\textsuperscript{168} Object Linking and Embedding (OLE) is a technology pioneered by Microsoft Corporation that allows applications based on prebuilt components.\textsuperscript{169} OLE increases the need for mandatory licensing because it is so easy to put a little program inside another program.\textsuperscript{170}

Compounding the OLE problem is LAN-to-LAN (local area network) communications and remote access.\textsuperscript{171} With this technology, the question becomes where is the software? Is it being used at the remote location, the local location, or both? Who is using the algorithms? It seems likely that in the near future, software will be sold

\begin{enumerate}
\item Glass, supra note 93.
\item Jube Shiver, Jr., \textit{Low-tech Problems with High-tech Patents}, L.A. Times, Jan. 9, 1994, at D1 ("A computer program either works or it doesn't work. You don't need the certification of your peers to prove that.").
\item It is anticipated that the vast majority of software petite patents will not be litigated.
\item McLachlan, supra note 89.
\item Id.
\end{enumerate}
and distributed electronically over the Internet, rather than in boxes over the counter.\textsuperscript{172}

This marriage of algorithm and hardware is further complicated by OLE’s ability to place motion-video files within other OLE-compliant applications. The public becomes confused when one vendor drops the proprietary video playback methods of another into a consumer’s innocent applications.

One solution to determining whose algorithm is being run in OLE mix-and-match applications would be a simple metering scheme. Each algorithm could trip a single mathematical flag each time it is completed. This tally would be stored in a special database, file, chip, or hardware. Royalties would then be contingent upon the metered tally.

3. It Is Hard to Win a Software Patent Infringement Suit

\textit{a. The Software Act is Better Than Current Law’s Best Case Scenario}

Software patent infringement suits are hard to win under the current system, particularly when the accuser is much smaller than the infringer.\textsuperscript{173} Software CEO’s say it is easy to get a software patent, but hard to enforce it without a protracted, costly legal battle, and the U.S. Supreme Court has never sided with a patent holder.\textsuperscript{174} A small company with limited resources can be defeated long before the trial date by a large company that strategically makes litigation expensive and time-consuming through delaying discovery and motions. A small company unable to continue litigation may be forced to settle on unfavorable terms. This was exemplified by the David and Goliath suit of \textit{Stac Electronics v. Microsoft Corp.}\textsuperscript{175}

Stac patented an LZS algorithm which effectively doubled the capacity of a computer hard drive by using transparent on-the-fly mathematics to compress data when stored and decompress data when used.\textsuperscript{176} Stac’s LZS algorithm consistently made the company’s "best-of-breed" and was the only product ever to win a second PC Magazine Technical Excellence Award.\textsuperscript{177} This type of compression

\begin{itemize}
  \item \textsuperscript{174} \textit{Id.}
  \item \textsuperscript{175} \textit{Id.}
  \item \textsuperscript{176} \textit{Id.}
  \item \textsuperscript{177} \textit{Id.}
\end{itemize}
algorithm is important with today’s computers because operating systems alone, like MS-DOS or IBM’s OS/2, consume an ever increasing percentage of a user’s hard disk (typically 35Mb to 60Mb).

Meanwhile, Microsoft programs continued to be plagued with bugs which frequently produced erroneous calculations. Microsoft sought a license for Stac’s algorithm, but no deal was reached despite Microsoft’s offer of $1 million per month. Furthermore, Microsoft was criticized for writing inefficient, bloated software, for both IBM compatible and Macintosh computers. As a result of the increased need for compression of its disk-intensive Windows programs, Microsoft incorporated Stac’s algorithm into its MS-DOS 6. The ensuing litigation that would either make or break Stac represented little more than a minor irritation to the giant Microsoft. Stac’s lawyer, Morgan Chu, worried that a jury would think Stac was seeking too much when asking for $110 million in damages. Ironically, the Los Angeles jury awarded Stac $120 million for infringement. Stac’s total award was reduced to $106.3 million.

Stac could not afford the expense, uncertainty, or the three to four year delay that would have resulted from an appeal to the U.S. Court of Appeals.

---

178. For example, several versions of Microsoft Windows contained a bug in the Calculator applet. It frequently gave the erroneous result of .00 when the correct result should be .01. (Try the simple subtraction of 2.01 - 2.00). Buzz Hunter, Want to Be Recognized as a Database Confidential Investigative Reporter? Earn Your Credentials and a Shirt With a Good Tip, DATA BASED ADVISOR, Feb. 1995, at 154.

179. Which is ironically what Microsoft ended up paying for the algorithm despite losing one of the most contentious legal battles in software history.

180. James W. Crawley, It Was a Big Deal for Stac, Microsoft After a $120 Million Suit, David and Goliath Forge Software Industry Alliance, SAN DIEGO UNION-Trib., Jul. 26, 1994, at C-1 (quoting Rikki Kirzner, analyst for Dataquest, a Silicon Valley market research firm) (“Microsoft has always been known for pretty sloppy code; it eats the hell out of everything . . . . It’s not efficient, it’s not well-written, it’s not well designed.”); Andy Ihnatko, Plaque Buildup, MacUSER, Jan. 1995, at 23 (“[T]he air has always been thick with rumors of Microsoft not caring about the Mac market or even of its taking active steps to kill it off.”).

181. Victoria Slind-Flor, The Lawyer Who Jammed Microsoft's Antitrust Deal, NAT’L L.J., Mar. 6, 1995, at B1 (quoting Mark A. Flagg, head of Los Angeles' Irell & Manella's intellectual property firm which represented Stac against Microsoft) (“It is difficult to take on Microsoft. They are very arrogant and they leave no stone unturned.”).


184. Id. Stac was penalized $13.7 million for improper use of a Microsoft secret. Crawley, supra note 180.

185. Crawley, supra note 180.
Therefore, Stac settled Microsoft's appeal, receiving $83 million (or about 76 cents on the dollar). Microsoft also bought a fifteen percent share of Stac. The settlement also included cross-licensing agreements that prohibit Microsoft from including Stac's source or object code in its own products, but that do allow Microsoft to write its own original data compression software utilizing Stac's patented technology. This license requiring payment for new code using old algorithms (and forbidding wholesale copying of code) is exactly the scheme that the Software Act envisions via statutory compulsory licensing.

Most firms would not do as well in similar litigation. Yet Stac's final result was still a cross-license with royalty fees. Would it not have been better to simply impose a statutory license, and leave it to the court to determine the adequacy of the compensation? The Software Act proposes such an approach.

b. Under the Current Law—Even When You Win, You Lose

Litton Industries, Inc. v. Honeywell, Inc. demonstrated that under the current law, even in the best case scenario, victory in an infringement suit is illusory. Litton won what is believed to be the largest patent infringement damages ever in the United States. The jury awarded damages of $1.2 billion. Judge Mariana Pfaelzer of the U.S. District Court for the Southern District of California completely dismissed the verdict upon a finding that the patent was invalid as a matter of law due to Litton's inequitable conduct in obtaining the patent without informing the Patent Office of a published article related to its technology. While this particular patent had nothing to do with software algorithms, the monumental jury award demon-

186. Stac incurred more than $8 million in legal fees from this conflict, while Microsoft's seven counterclaims and declaratory judgment motion to invalidate Stac's patents for inequitable conduct, increased both time and costs. Corporate Profiles 1995: Microsoft Can't Beat Stac, So It Pays Stac, SAN DIEGO DAILY TRANSCRIPT, Jan. 23, 1995, at S48 [hereinafter Corporate Profiles].
187. Crawley, supra note 180.
188. Corporate Profiles, supra note 186.
190. Susan Orenstein, Judge Throws Out $1.2 Billion Award, Invalidates Patent; Says Litton Withheld Information From PTO, RECORDEmER, Jan. 10, 1995, at 1.
191. Id.
192. Id. (quoting Judge Pfaelzer) ("[T]he jury's August 1993 award was 'inconsistent with the clear weight of the evidence ... to permit it to stand would constitute a miscarriage of justice.'").
193. It was on coating mirrors used in aircraft navigation systems. Litton v. Honeywell, 1995 U.S. Dist. LEXIS at *2.
strates the potential for factually and legally complex cases to result in a windfall, equivalent to winning the lottery.\textsuperscript{194}

The \textit{Software Act} would solve these problems by providing a vendor the certainty needed as an incentive to invent in the first place. Compulsory licensing allows the big competitor to use the technology, but it requires the competitor to pay a reasonable price for it. The courtroom may continue to serve as a battlefield where the parties wrangle about what amounts to a reasonable price, but the statute would be favorable to the vendor. The \textit{Software Act} will save the parties, the government, and the public, unnecessary costs wasted on court expenses and stifled product development.

4. Overreaching Patent Claims

Regular utility patents are inappropriate for algorithms and software due to overreaching patent claims.\textsuperscript{195} A prime example is Compton Encyclopedia's broadly worded patent that claimed techniques used by multimedia software to search for data.\textsuperscript{196} These claims seemed to cover virtually all multimedia and hypertext.\textsuperscript{197} When Compton tried to enforce this patent, its multimedia competitors raised such an uproar that PTO Commissioner Lehman took the extraordinary step of \textit{sua sponte} reexamining the patent.\textsuperscript{198} The PTO ultimately found significant prior art and vastly reduced the patent's scope.\textsuperscript{199}

Overreaching claims will probably exist in every patent system. However, the \textit{Software Act} makes an effort to reduce the occurrence and effect of overreaching claims because overreaching claims have a more severe impact upon the software market than upon other markets. Because the software industry moves faster than other markets, vendors are less likely to challenge overreaching claims of competitors. They are more likely to simply avoid development and introduction of a product whose entire life cycle may be shorter than the patent examination-litigation cycle. The \textit{Software Act} seeks to prevent this by limiting the prevalence of overarching claims through central claiming rather than peripheral claiming. The \textit{Software Act} increases the speed of the patent cycle to match that of software development by

\begin{thebibliography}{9}
\expandafter\ifx\csname natexlab\endcsname\relax\def\natexlab#1{#1}\fi
\bibitem{orenstein} Orenstein, \textit{supra} note 190 (quoting Matthew Powers, a partner in the Menlo Park office of New York's Weil, Gotshal & Manges, unrelated to the case).
\bibitem{helyar2} \textit{Id.}
\bibitem{hypertext} Hypertext uses text or graphics as entry points into a data set.
\bibitem{helyar3} Helyar & Doudnikoff, \textit{supra} note 195.
\bibitem{helyar4} \textit{Id.}
\end{thebibliography}
eliminating the examination phase until challenged. Where an examining court finds overreaching claims, it may modify or strike them to match what a claimant has actually implemented beyond a statutory filter. The risk of infringement is diminished to reasonable compulsory fees. The fees should be greater than the cost of a directly negotiated license (to encourage blanket clearing house licensing), but they should be less than a cost that would impede development and stifle innovation.

5. Patent Antitrust Issues

The Software Act is needed to alleviate antitrust problems now occurring in the software industry. The software industry is experiencing rapid consolidation toward a handful of vendors.

In today’s market the small developer who does manage to invent a software algorithm finds the algorithm too expensive to protect and too expensive to litigate once protection is attained. The Supreme Court has expressed its concern that small businessmen should be protected from elimination by large competitors, even where their destruction makes little difference to the overall economy. For example, Justice Douglas’ concurrence in United States v. Falstaff Brewing described the crippling effect that the closing of the local sawmill by large New York auditors had on Goldendale, Washington. Douglas forcefully noted that “a nation of clerks [rather than entrepreneurs] is anathema to the American . . . dream.” Furthermore, he predicted that unhindered concentrations of power will lead to big business owners who regulate markets like socialist production commissars, in a way that is antagonistic to our system. The Software Act would appease the Court’s concern that antitrust laws should promote avenues for small businesses to succeed. In the software industry, there is not necessarily anything wrong with a big company gobbling up all the good ideas of a little company (or even gobbling up the little company itself), provided that the small vendor is compensated with compulsory license fees (or better, negotiated license fees). The public users benefit from the latest and best features conceived by the small businesses combined with the most popular software produced by the largest vendors.

202. Id. at 543.
203. Id.
204. Id. at 543 (citing The Goal of Antitrust: A Dialogue on Policy – In Defence of Antitrust, 65 COLUM. L. REV. 377 (1965)).
6. Supreme Court Calls for Legislation

The Supreme Court, in *Benson*, stated that Congress has the authority to authorize software algorithm patents. Congress may draft such a statute since it is within Congress' purview to determine what "promotes" the useful arts, as directed by the Constitution.

It may be that the patent laws should be extended to cover these programs, a policy matter to which we are not competent to speak . . . . [But if] these programs are to be patentable, considerable problems are raised which only committees of Congress can manage, for broad powers of investigation are needed including hearings which canvass the wide variety of views which those operating in the field entertain.

The Court reiterated this suggestion in *Flook* stating: "Difficult questions of policy concerning the kinds of programs that may be appropriate for patent protection and the form and duration of such protection can be answered by Congress on the basis of current data not equally available to this tribunal." This paper concludes with the suggestion that Congress should create a "small patent" or petite patent category for algorithms, which is of shorter duration and with fewer requirements than current § 101 patents.

V. SOFTWARE AS COMMERCE (TRADEMARK MATERIAL)

A. Protect Only Software Used In Commerce

Computer programs are not merely forms of writing, they are also products. The *Software Act* takes the position that only software used in commerce should be protected. The purpose of statutory protection for software is to protect the economic value of software already written and to provide the economic incentive to write more. Therefore, the scope of the protection should be limited to bolstering that economic market. The market itself has demonstrated that physically preventing private, nonmarket copying of software does not protect the software, since consumers will simply refuse to purchase the software.

The market has also demonstrated, through the success of

---

206. *Id.* at 72-73.
207. *Id.*
209. *Id.* at 595.
210. Bob Woods, *Shareware Award Winners*, *Newsbytes*, June 27, 1995, *available in Westlaw*, Computer File (quoting Shareware Industry Awards Foundation director Eric Robichaud) ("The shareware industry is growing very quickly because the 'try before you buy' concept of shareware is becoming accepted by not only game publishers, but from business leaders like Microsoft as well.").
“shareware,” that the market itself is only concerned about commercial (including governmental) infringement.211 Furthermore, because software is such a close variant of ideas and mathematics, any form of software protection must address the concern about consuming the whole field.

Under trademark law, only those products actually used in the market are worthy of protection.212 Likewise, courts have become concerned about competing software products that are so similar that they may appear to be the same product to the user. Judge Keeton, in Lotus v. Paperback,213 noted that “a user could easily think 1-2-3 rather than VP-Planner was the program in use.”214 Software standardization brings the threat that two outwardly different programs that operate the same way, like two different models of toasters, could confuse users. There is no conceptual reason why a software vendor should not get the eternal protection of a trademark on those features (screen appearances, menus, or even “mode of operation”) that indicate to the public that the software with these features was produced by a particular vendor. The Software Act suggests that those features that become standardized as belonging to all similarly-situated toasters should not be protected.

Software companies have discovered that copying makes for good marketing. An indigenous “shareware” market has developed outside the realm of the regular commercial market, which encourages users to upload the software onto electronic bulletin boards and the Internet, to demonstrate the software and its capabilities, and to give copies to potential users.215 Shareware depends upon the honor system for payment of nominal fees, often offering in exchange additional documentation, features, or updates.216

In fact, there may be a benefit to allowing private, noncommercial infringement, since the resulting increased exposure would expand the market for new software. Many programs have emerged from shareware into viable, full-featured commercial versions by gaining market acceptance.

211. Id.
214. Id. at 70.
215. See cases cited supra note 111.
216. For example, McAffee distributes industry standard anti-virus software on the electronic marketplace of Bulletin Board Systems (“BBSs”) and via Internet. McAffee grants five days of royalty free evaluation of the software. Then commercial and governmental users must negotiate a license, while individual, noncommercial users receive a one-year nonexclusive right to use one copy for $25 upon registration. McAffee, Inc., 2710 Walsh Avenue, Santa Clara, California 95051-0963.
B. Noncommercial Use Is Not Infringement

A distinction between commercial and noncommercial use of copyrighted works is not new. Traditionally, noncommercial use of literary works has not been wholly prohibited under copyright law. Section 110 of the Copyright Act lists ten exemptions relating to the noncommercial use of copyrighted works.217 The Audio Home Recording Act of 1992218 likewise provides that "noncommercial use by a consumer" is not an infringement.219 The Software Act is consistent with these Congressionally approved noncommercial exemptions from copyright protection.

C. Vaporware

The District Court for the District of Columbia recently expressed concern about the anticompetitive effect of Microsoft and other vendors who publicly advertise new software without intent to immediately supply that software to the market.220 The court labeled this as "vaporware."221 Vaporware is defined as "the public announcement of a computer product before it is ready for market for the sole purpose of causing consumers not to purchase a competitor's product that has been developed and is either currently available for sale or momentarily about to enter the market."222 The Software Act addresses the vaporware problem first by requiring an applicant to file a good faith, bona fide "intent to use in commerce" form, similar to that used in the Trademark Act, including the same perjury considerations as in the Trademark Act.223 Second, the Federal Trade Commission Act224 would be amended to make preannouncing a product with intent to injure a competitor an unfair or deceptive act or practice (UDAP).

217. 17 U.S.C. § 110 (1988) (including performance or display in a teaching environment, nonprofit institution, or religious service; where there is no private gain and no owner objection; transmissions received on a home-style apparatus; state fairs; music for blind or handicapped; and veterans or fraternal organizations for charity).
221. Id. at 334.
222. Id.
223. See Software Act § 224(b)(3), infra Appendix.
224. See Software Act § 233(c)(5), infra Appendix.
D. Insufficient Lead Time to Recapture Software Development Costs

Products that enter the marketplace first enjoy an inherent head start advantage, which allows them to recapture costs of development before competitors enter the market and force down the price. ²²²⁵ Combined with trade secret protection, monopoly protection on software and algorithms may even be detrimental to society by unnecessarily retarding innovation by others. However, the software marketplace is unlike that of other markets. Although the cost of software research and development is high,²²⁶ there is a short turnaround time between initial production and competitor’s release of derivative algorithms and expressions.²²⁷ The short software lead time is long enough to give competitors the advantage of seeing what does and does not work, without risk. When the short lead time has expired, free-riding competitors begin nipping at the heels with similar products.²²⁸ The cost of production and distribution is identically minimal for both parties, since computer disks and data are fungible.²²²⁹

Lead time is also reduced in the software market by the phenomenon of the “penguin effect,” which does not appear in other markets. Software users realize that new versions of any program will contain bugs, compatibility problems, and even dangers to other properly working software and data. Like penguins, potential lead time users prefer to push their peers into the shark-infested waters to determine whether it is a safe environment. In a Catch-22, software does not become profitable until it becomes an industry standard, yet it cannot become an industry standard until it develops sufficient market share despite the penguin effect. This explains the aggressive advertising, low initial pricing, and inexpensive upgrade path on new products. The Software Act attempts to alleviate this conundrum by restoring an artificial period of lead time to this industry — allowing the first comer to receive temporary compulsory royalties to recoup the considerable cost of research and development.²³⁰

²²⁶ Mark Aaron Paley, Lotus Lookalike Litigation: Landmark or Limbo?, 40 Buff. L. Rev. 283, 310 (1992) (suggesting that today’s software is years behind what today’s hardware is capable of supporting).
²²⁷ Id.
²²⁸ Id.
²²⁹ Id.
²³⁰ See Software Act § 230, infra Appendix.
E. Orphanware

Computer commentator John Dvorak coined the term "orphanware" to refer to software that has been commercially abandoned. Dvorak suggests that companies with orphaned software products should release the source code into the public domain. Many good products have been abandoned for marketing reasons, because sales were sluggish or support costs too high. Bad will is created among angry consumers who are impliedly promised, via heavy advertising, that they should risk a monetary and training investment in the product because the company will continue to maintain and improve the product.

Dvorak’s concept of abandonment is consistent with the view under trademark law that protection should cease when commercial use ceases. It would also harmonize the United States with the many countries that maintain the principle that patent protection should cease when the patent is not commercially exploited. The difficulty in applying a similar concept to software is software’s similarity to books, which do not lose protection due to nonuse. In fact, copyrighted material remains protected even fifty years after an author has died (whether or not a commercial market exists for the work). However, it is not conceptually inconsistent to carve out software from other copyrighted material, and provide it with less protection. Fifty years is an inappropriately excessive length of time to protect software. Furthermore, copyright law carves out printed material consisting solely of trademarks as nonregisterable.

F. Patent "Bracketing" and "Mapping"

The Software Act’s use in commerce requirement deliberately prevents the techniques known as patent bracketing and patent mapping experienced with regular patents. "Patent bracketing" occurs where a domestic company quickly reserves patents for itself on a wide range of applications or improvements on a foreign patent. When the foreign competitor tries to enter the domestic market with infringing products, the domestic patent owner simply offers a cross-
licensing agreement, thus avoiding a steep royalty fee in the other direction.\textsuperscript{239} The similar technique known as "patent mapping," often used by U.S. firms, involves an active search or scouring of existing patents to look for unclaimed related fields or unclaimed improvements.\textsuperscript{240} They look for the inventive high ground and then file patents reserving the broadest possible available tangential fields and improvements.\textsuperscript{241} The \textit{Software Act} defeats these two techniques by eliminating the reservation system. Only commercially used embodiments of algorithms receive protection. Patent bracketing is eliminated under a world-wide registration scheme that erases national technological boundaries. Patent mapping is made unnecessary where core algorithms are readily available for a reasonable fee under compulsory licensing.

VI. \textbf{NEW FEATURES UNDER THE \textit{SOFTWARE ACT}}

A. \textit{The Distinction Between Printware and Machineware}

In 1908 the Supreme Court in \textit{White-Smith Music Publishing Co. v. Apollo Co.} \textsuperscript{242} distinguished perforated piano rolls from ordinary literary works because they were not intended to be read by humans.\textsuperscript{243} Even people skilled in making piano rolls were unable to read them as literary or musical works.\textsuperscript{244} The perforated rolls became a part of a machine since they were adapted to operate the machine to produce copyrightable musical tones when properly applied.\textsuperscript{245} The Court expressly recognized Congress' prerogative to protect such machine-readable works by statute,\textsuperscript{246} whether via the Copyright Act or other legislation, such as, the \textit{Software Act}. Congress chose to protect machine-readable works by amending the Copyright Act of 1976 definition of "copy" to include material objects "from which the work can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device."\textsuperscript{247}

The \textit{Software Act} codifies this same distinction between human-readable software ("printware") and machine-readable software ("machineware"), providing different levels of protection for each. The distinction removes printware from the \textit{Software Act}, treating it

\textsuperscript{239} Id.
\textsuperscript{240} Id.
\textsuperscript{241} Id.
\textsuperscript{243} Id.
\textsuperscript{244} Id. at 18.
\textsuperscript{245} Id.
\textsuperscript{246} Id.
like any other copyrightable literary work, and abiding by the TRIPs mandate of protecting computer software as literary works under the Berne Convention. However, in compliance with White-Smith v. Apollo, the Software Act provides a new statute offering lesser legislative protection to machineware than it does to human-readable software. The Software Act would provide Courts with a mechanism to legitimately consider machineware a part of the machine. This distinction may seem somewhat incongruous since it is fairly easy to convert from printware to machineware and vice versa. However, this distinction is not as unusual as it may appear at first glance. Virtually the same distinction with the similar rights is afforded in the music industry. Although the sheet music for a song, as printed matter, is fully protected by copyright, the right to reproduce the composition in sound recordings is covered under the Copyright Act § 115 compulsory licensing scheme. That section allows anyone, upon payment of a statutory royalty, to record a "cover" version of the song (provided that the copyright owner of the song has explicitly authorized an initial release of the song in sound recordings). In the same way, when a software creator distributes the software in the form of a printed book, the Copyright Act should apply to prevent others from distributing copied books. However, when the material in the book is converted into machineware, the Software Act would apply, granting others a compulsory license to use the protectable elements in the software.

B. Compulsory Licensing of Algorithms

Scholars argue that software patent licenses do not work very well. John Swinson points out three common arguments: (1) searching for and understanding a patent is difficult even for the diligent inventor; (2) large programs have thousands of algorithms to search; and (3) large firms stifle competition by reserving broad markets of algorithms, then refuse to license (or demand high license fees) or force competitors to waste resources inventing less efficient noninfringing workarounds. The Software Act proposes to alleviate all of these problems. The searching role is greatly reduced by removing it as a prerequisite to a petite patent. Thus, a small inventor may file an

248. TRIPs, arts. 2, 9, 10; see also Oman, supra note 53.
249. 209 U.S. 1 (1908).
253. Id. at 168-69.
application without first searching for prior art. The search is only necessary during the enforcement phase, to determine who filed first and whether the claim was already in the public domain (i.e., an expired patent). The burden is placed primarily on those who can afford it most—the big software vendors and the software rights societies. Electronic access to the Software Registry should make their search for a new feature even easier, while also providing corresponding patent numbers. Software rights societies can handle much of the searching as they perform the administrative work of maintaining databases of thousands of algorithms, just as ASCAP and BMI do now in maintaining databases of compositions, songwriters and music publishers. Under the Software Act, large firms will simply be unable to reserve large swaths of unused algorithms. Absolute refusal to license (or refusal to license at reasonable rates) is countered by compulsory licensing for commercial users and free licensing for noncommercial users.

C. Blanket Licensing

The Supreme Court, in Broadcast Music, Inc. v. Columbia Broadcasting System, Inc., condoned large organizations acting as clearinghouses for licensing copyrighted works. The American Society of Composers, Authors and Publishers (ASCAP) and Broadcast Music, Inc. (BMI) offered music users the right for a stated term to publicly perform, as often as desired, compositions owned by its members and affiliates. The Court cited economic factors justifying blanket licensing in the music market. These economic factors are not only paralleled, but actually exacerbated, in the software industry. The factors included: the extraordinary number of users across the country; the sheer volume of copyrighted works; the enormous quantity of performances each year; the ephemeral nature of each performance; the ease of infringing (performing music in public) without detection; the impracticality of negotiating individual licenses for each composition; and the fact a music composition can be consumed simultaneously by many different people without the owner’s knowledge.

254. ASCAP (American Society of Composers Authors and Producers) and BMI (Business Music, Inc.) are performance rights societies, which collect royalties for the public performance of compositions on behalf of songwriters and publishers.
256. Id. at 19-21.
257. Id. at 20.
258. Id.
The Software Act makes economic sense under the factors that the Supreme Court applied to the music industry. The Court found that blanket licensing results in a substantial lowering of costs, compared to individual licensing, benefitting both sellers and buyers. A few large clearinghouses can achieve economies of scale in negotiating, monitoring, and administering licenses and payments. The clearinghouses can serve a market need by allowing immediate and flexible access to many compositions without the delay of noneconomic negotiations. There is a dire need for similar benefits in the software industry to protect small software creators. At the beginning of the software industry, there was actually more good software coming out of programmers' basements than in corporate computer labs. However, the burgeoning cottage software industry has diminished partially due to the increased complexity of today's software and partially by the lone inventor's inability to protect his rights against large firms. The Software Act would protect a lone genius who files a simple petite patent application, while allowing the big firm to continue incorporating new algorithms into existing applications (after paying a reasonable fee to the basement inventor).

The Software Act is also consistent with the principle that authors and artists need to be protected from hard-nosed businesspeople. The law should free them to devote their talents to creating expressive works rather than business wrangling. It is expected that unknown and struggling artists do not have the business leverage or savvy necessary to negotiate the best license possible. Likewise, the Software Act considers computer programmers to be artists rather than businesspeople. Computer programmers are basement inventors. An ASCAP-like agency, perhaps the Software Protection Association (SPA), the Association for Computing Machinery (ACM), the International Society of Electrical and Electronic Engineers Computer Society (IEEE), or similar organizations, would be appropriate clearinghouses providing a standardized set of fees, thus relieving the small programmer of the need to perform such negotiation.

259. Id. at 21.
260. Wayne Ratliff wrote dBase III, a program which is still selling well (despite the introduction of dBase IV and dBase for Windows), virtually by himself in three months. Bill Gates himself wrote MS-BASIC in a hotel room in a few weeks. The NEWDOS operating system for the TRS-80, written by one programmer literally in his basement, was very successful because it was so much more powerful than the official operating system TRS-DOS.
261. See supra part IV.E.3.
262. The old joke is that a real computer programmer's worst fear is direct sunlight.
D. Reverse Engineering/Decompilation Right

Generally, the term "reverse engineering" of a computer program refers to a variety of different activities used to reveal the design of computer software. The least intrusive form of reverse engineering is often referred to as "black-box" reverse engineering, which analyzes the input and output of a program without attempting to view its internal design. Black-box engineering is often used to try to make one program compatible or interoperable with another. More intrusive reverse engineering requires disassembling or decompiling software by using special software to translate unintelligible machine-dependent object code back into an approximation of the original human-readable machine-independent source code. The decompiled code will not be exactly the same as the original source code. However, study of the decompiled code can reveal the underlying design or engineering of a computer program.263

Disassembly, by its very nature, requires making at least one copy of a computer program, since the disassembly program generates computer files and printouts of both the original object code and the reconstructed source code. The disassembled source code itself also constitutes a derivative work of the original object code program. As users further modify, adapt, and recompile the disassembled source code, they are making additional derivative works. Unless a fair use exception is found,264 these acts violate a software owner's exclusive rights.265

Sega Enterprises Ltd. v. Accolade, Inc.,266 held that since disassembly involves copying and making derivative works, these acts are squarely prohibited by § 106 of the Copyright Act.267 Thus reverse engineering is per se unlawful.268 However, reverse engineering may be a fair use where no alternative means exists to gain an understanding of the underlying uncopyrightable ideas and functional concepts of a work.269 Functional aspects of software are unprotected and noncopyrightable under the Copyright Act.270 Thus, Accolade could disassemble Sega's video game cartridges to discover these unprotected functional components, since no other viable alternative existed to discover them and could reverse engineer as much as needed to

263. Johnston & Grogan, supra note 69, at 4.
266. Sega Enters. Ltd. v. Accolade, Inc., 977 F.2d 1510 (9th Cir. 1993).
267. Id.
268. Id. at 1519-20.
269. Id. at 1527-28.
discover how to make cartridges functionally compatible with Sega's Genesis machine. However, wholesale disassembly would still be per se unlawful where fair use is not found.

In the wake of both the Ninth Circuit's *Sega v. Accolade* and the Second Circuit's *Computer Associates v. Altai*, where reverse engineering (i.e., "copying") was permitted to discover the uncopyrightable functional elements, many software vendors are now dumping the interface specifications in their competitor's laps. Software developers are allowing competitors to create functionally compatible products, as a way to foreclose the fair use argument by creating alternative means to learn the unprotectable elements other than reverse engineering.

The *Software Act* suggests a far more equitable approach. Vendors need not give away their functional compatibility standards for free, by publishing them or by tolerating unfettered reverse engineering. Neither should competitors be locked out from industry standards. The *Software Act* achieves an even playing field by allowing anyone to reverse engineer. Competitors may freely use any nonprotectable elements, which would be filtered out under the infringement test. However, competitors must pay a reasonable compulsory licensing fee to use, in commercial software, the patented algorithms they discover through reverse engineering.

The *Software Act* also discourages software trade secret infringement suits. Software vendors have begun to argue that valuable ideas are hidden in the internal engineering of their mass distributed commercial software. Microsoft won one such trade secret counterclaim against Stac when Stac reverse engineered MS-DOS specifically to uncover MS-DOS's unique ability to start up simultaneously with a computer's power supply (discussed *infra*). Yet, legal and industry commentators argue that the concept of trade secrets is inherently inconsistent with a widely distributed product like software. The trade secret law in many states holds that readily ascertainable information is ineligible for trade secret protection because such information is treated as generally known. The mere ability to disassemble software (even where it is not practiced) renders software's secrets inherently readily ascertainable and generally known.

272. Johnston & Grogan, supra note 69, at 1.
273. Id. at 2.
274. Id. at 5.
275. Id.
276. Id.
Vendors attempt to counter these state trade secret laws by utilizing state contract law. Vendors insert shrinkwrap licenses wherein the user agrees, by contract, to keep the trade secret in confidence and not to reverse engineer. The Software Act clears any doubt about whether federal copyright law or state trade secret law can prohibit reverse engineering. The Software Act overrides them both. Furthermore, the Software Act encourages voluntary revelation of software trade secrets either by publication or by patent, because functional nonliteral elements not claimed in petite patents are fair game for unlicensed usage.

E. Compulsory License Right to Port an Algorithm into New Hardware

Generally, the term “porting” is used to mean the process of adapting a piece of software to work in different hardware environments or new computers. Such adaptation can be analogized to the recording of a musical composition. In music, we allow a second comer to record a first comer’s song in his own voice, as long as the second comer pays a compulsory license fee to the creator of the song. Software porting should operate the same way, allowing the software to sing in a new machine with a different voice.

Critics may argue that a porting right is similar to a translation right. Under the Copyright Act, only the author would have the right to translate a work from English to French. But Congress realized that a software consumer is interested primarily in computer uses of software, not admiring software’s copyrightable prose, aesthetic, and artistic qualities. Congress granted the software buyer or licensee the right to adapt that software enough to make it work on the owner’s computer. The Software Act right to port is merely §117(1) applied to a third party who does not own the underlying software, but who

277. Id.
278. IBM Dictionary of Computing 517 (George McDaniel ed.) (1994) (defining “port” as “To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer”).
279. E.g., Whitney Houston might be granted a compulsory license to reproduce her version of Dolly Parton’s “I Will Always Love You.” In actuality, Houston probably licensed the song privately from Parton without using the compulsory license scheme. Often such licenses are granted at less than the statutory rate when a new recording is likely to be a hit. However, if Parton did not wish to license the song, Houston could nonetheless record and release her version by complying with the compulsory license provisions. See generally Al Kohn & Bob Kohn, The Art of Music Licensing, ch. 5 (1994)
pays a license fee. Hence, due to the special nature of computer software, a porting right should overrule a vendor's §106(2) exclusive right to prevent others from translating his software. The royalty fee for porting should be quite high, providing second comers with a low profit, since porting is likely to make high use of the first comers' algorithms and protected elements, while adding only minor protected elements of the second comers' own.

F. Exhaustion

Most commercial software vendors do not sell their products to their customers—they license the products. The typical shrinkwrap license retains title to the software in the vendor in perpetuity. The license prohibits consumers from reselling the software. What makes shrinkwrap licenses a "really ugly field of the law" is that they involve unsigned contracts governed by a mass of state contract law, which has not kept pace with the growth of the software industry. 283 "The issue of software licensing is a national problem that requires federal legislation to clarify licensing issues that involve software sold in interstate commerce." 284 The Software Act provides exactly that legislation by determining that the petite patent right to sell off-the-shelf, canned software ceases upon first sale. A consumer should have as much right to sell software when finished using it as he does to sell a book when finished reading it. The Software Act makes economic sense by allowing consumers to sell both old hardware and software in a combined package to a new user. Many new users, frugal consumers, and developing countries find value in old software. Ultimately these novices will learn from the old software and eventually upgrade to newer software themselves. As their needs expand to fill their computer knowledge, they will eventually need to replace their hardware with better equipment as well. Meanwhile, those who sell their old software can then use the income to buy new, upgraded software, increasing the overall commerce and demand in the market.

The Exhaustion Provision also alleviates problems of delegation of software ownership rights like those in MAI Systems Corp. v. Peak Computer, Inc. 285 In that suit, MAI sold computer hardware and licensed computer software to many customers. 286 It was undisputed
that MAI owned the copyright to the operating system software.\textsuperscript{287} Several MAI employees broke away from the firm and joined Peak Computer, which performed routine maintenance and emergency repair of MAI computers.\textsuperscript{288} Maintaining a computer and diagnosing problems at the customer's site required running the MAI operating system and viewing the MAI systems error log.\textsuperscript{289} The mere act of turning on the computer caused MAI's software to be copied from a storage medium into the computer's RAM (random access memory).\textsuperscript{290} The software license agreement allowed MAI customers to use the software for their own internal information processing, but it did not allow for use or copying of the software by third parties.\textsuperscript{291} The Ninth Circuit determined that Peak violated MAI's copyright by copying the software from hard disk into RAM.\textsuperscript{292} Although a MAI customer was entitled to perform the same acts for itself, under the software license, the MAI customer had no right to delegate these rights to third parties to perform on the customer's behalf (however, if the customer could have performed this maintenance for itself, it would not need to call in a service company like Peak Computer).\textsuperscript{293}

The seemingly inequitable outcome of this case would be rectified in most cases under the Software Act. The Software Act would provide that off-the-shelf (noncustom) software is sold to customers, not licensed. The vendor's right to use the software is exhausted upon the first sale of that particular copy. The exhausted rights, which cede to the customer, may explicitly be encharged to a customer's authorized agent. However, as a compromise, the Software Act would not allow MAI-type licenses in situations where the software is custom-written (not off-the-shelf) for a particular client. In the custom-written software transaction, the customer has greater leverage to bargain for inclusion or exclusion of a third-party maintenance right via bona fide direct negotiations.

G. \textit{Cheap Easy Filing}

The current utility patent model requires an examination by technical experts of the utility, novelty, and inventive level of the invention.\textsuperscript{294} Although this serves to filter out inventions failing to meet

\textsuperscript{287} \textit{Id.} at 517.
\textsuperscript{288} \textit{Id.} at 513.
\textsuperscript{289} \textit{Id.} at 518.
\textsuperscript{290} \textit{Id.} at 511, 518.
\textsuperscript{291} \textit{Id.} at 517.
\textsuperscript{292} \textit{Id.} at 519.
\textsuperscript{293} \textit{Id.} at 517.
\textsuperscript{294} 35 U.S.C. § 131.
these statutory requirements, it can be expensive and time-consuming. The Software Act endeavors to achieve a standardized filing form much more like a simple Copyright Form TX or Form VA, which even a layman can complete and submit for $20.00.\textsuperscript{295} Under the Software Act’s requirement of disclosing the patented sequence of computer program steps, it will be necessary for an applicant to file a complete source code or object code listing. Since many programs are comprised of tens of thousands of lines of text, it would be acceptable to submit the program in digital form. Object code submission would be allowed because it discloses a sufficient quantity of information for a programmer to reverse engineer it into an functioning approximation of the original source code and algorithms.

The Copyright Office already maintains a Computer Shareware Registry, to track ownership, transfer of ownership, and security interests in computer shareware.\textsuperscript{296} The Patent and Trademark Office already requires applicants to file computer listings for long sequences of DNA.\textsuperscript{297} Therefore, it would not be unreasonable for the government to maintain a registry of software and algorithms. Nor would it be unreasonable for the government to receive this information in a digital format.

VII. WHAT CONSTITUTES INFRINGEMENT UNDER THE SOFTWARE ACT

A. What is the Purpose of an Infringement Action?

An infringement action would serve many purposes under the Software Act. The patent search, the patent examination, the patent interference, the patent opposition, and the patent transfer would all be postponed until an infringement action is brought. It is expected that the vast majority of software patents will never reach litigation. When a party brings an action, all of the above functions will be performed at once. A court would determine which party filed a valid claim first-in-time, invalidate other claims, recast claims where necessary, transfer misappropriated claims to the correct party, and impose reasonable license fees and penalties on infringers.


\textsuperscript{297} Biotechnology Invention Disclosures Application Disclosures Containing Nucleotide and/or Amino Acid Sequences, 37 C.F.R. § 1.821(e); see also 37 C.F.R. § 1.821(a) (a "long sequence" consists of an unabridged sequence of four or more).
B. Overview of the Abstraction-Filtration-Comparison Test to be Codified

The infringement test used in *Lotus v. Paperback*298 and its progeny evolved with insight from Professor Nimmer into the Abstraction-Filtration-Comparison Test.299 Following a lengthy discussion on the constitutionality of a court’s ability to draft its own nonstatutory test,300 Judge Keeton divined the infringement test by examining: (1) the language of the statute as a whole;301 (2) its object and policy;302 and (3) the markers that Congress had placed to delineate the boundary between copyrightability and noncopyrightability.303 Keeton found that copyrightability turns upon whether or not an element expresses an idea in an original way.304 To make such a determination, the court first should determine what the idea and expression are; then, filter out expressions unworthy of protection; and finally, compare the accused expressions to remaining expressions to determine whether they were copied.305

The first step is to separate the idea being expressed from the expression itself. The court must conceptualize or abstract the idea “along the scale from the most generalized conception to the most particularized.”306 The abstraction process necessarily places the expression somewhere on a flexible continuum from very specific to very abstract.307 The abstraction process itself biases its own outcome, because the more specific an expression is ultimately determined to be, the more likely it will be found protectable. Furthermore, the abstraction decision is doomed to be ad hoc in nature, based upon the facts of the case.

299. See *Mitek Holdings, Inc. v. Acre Eng’g Co., Inc.*, 864 F. Supp. 1568, 1577 (S.D. Fla. 1994) (quoting 3 Nimmer § 13.03[F at 13-102.17) (proposing the abstraction-filtration-comparison test as a way to “help a court separate ideas [and processes] from expression and eliminate from the substantial similarity analysis those portions of the work that are not eligible for copyright protection.”).
300. *Paperback*, 740 F. Supp. at 46. It is ironic that the court should question its ability to draft where Congress has been silent, since CONTU Commissioner Arthur Miller points out that Congress adopted the Commission’s position that the courts should be free to develop a doctrinal matrix to fill in the gaps of the statute. See *supra* note 39 and accompanying text.
303. Id. at 47, 52-53 (citing *Kelly v. Robinson*, 479 U.S. 36 (1986)).
304. Id. at 59-60.
306. Id. at 60 (citing *Nichols v. Universal Pictures Corp.*, 479 U.S. 36 (1986)).
307. Id. at 60-61.
The second step is to filter out expressions that cannot be copyrighted. Under the merger doctrine, where there are only a limited number of ways to express an idea, the expression cannot be copyrighted.\textsuperscript{308} The expression is said to merge with the idea because other authors are left with no other noninfringing way to express the unprotected idea. This is in effect granting a copyright on the idea itself,\textsuperscript{309} which The Copyright Act expressly forbids.\textsuperscript{310}

The third step of the test for infringement is a comparison of the remaining protectable elements against the allegedly infringing elements. The Second Circuit, in \textit{Computer Associates v. Altai},\textsuperscript{311} called these remaining elements the protectable "golden nugget" of a software program.\textsuperscript{312} If the accused software has misappropriated a substantial portion of the golden nugget, then infringement has occurred. In the comparison step, the decision maker must weigh the quantitative and qualitative substance of the questioned expression. A quantitatively small fragment of an expression may be qualitatively large.\textsuperscript{313}

Codification of the abstraction-filtration-comparison test becomes even more important today in light of the First Circuit's deliberate creation of a split between the circuits by rejecting the test. The First Circuit, in \textit{Lotus v. Borland},\textsuperscript{314} explicitly created a split between itself and the Ninth and Tenth Circuits by expressly side-stepping the doctrinal analysis developed by its sister courts.\textsuperscript{315} It also created an implicit split with the Fifth Circuit, which expressly adopted the abstraction-filtration-comparison test in \textit{Engineering Dynamics v. Structural Software}.\textsuperscript{316} The First Circuit's drawing of lines in the sand is a virtual invitation to Congress (or the Supreme Court) to settle the rift. The \textit{Software Act} does so.

The First Circuit unhelpfully created a doctrinal schism by straying from application of the abstraction-filtration-comparison test to every software infringement case. The First Circuit did so despite admitting it was navigating uncharted waters in a case of first impres-
The First Circuit set the wrong course. In *Lotus v. Borland*, the First Circuit held that the abstraction-filtration-comparison test does not apply to cases of literal copying. The court reasoned that the abstraction step is unnecessary since literal elements are already abstracted as themselves. The court suggested that confusion can be avoided in literal copying cases by completely skipping the abstraction-filtration-comparison test. However, in this author's opinion, creating a bifurcated infringement test actually creates more confusion without saving the court any real work. Furthermore, whether or not the court officially labels it as such, it must nevertheless perform an abstraction step by framing the issue to be decided.

The First Circuit, in *Lotus v. Borland*, inherently performed an abstraction step by determining the issue to be "[w]hether a computer menu command hierarchy constitutes copyrightable subject matter." The abstracted element to be tested was the menu command hierarchy. The court merely needed to proceed to the filtration step, apply the Copyright Act § 102(b) method of operation filtration, and eliminate the menu command hierarchy element as uncopyrightable. The court would then proceed to the comparison step, find no elements remaining after the filtration step and come to the same conclusion that it did where it skipped this abstraction-filtration-comparison test. There was no need for the First Circuit to depart from the established test. The First Circuit's mistake was shortsightedness in seeing only a single filter available as a filter (i.e., the merger doctrine filter).

The Tenth Circuit in *Gates Rubber v. Bando Chemical* correctly found many filters capable of removing unprotectable elements during the filtration step, including: any of the reasons in § 102(b); public domain elements; elements dictated by function under the merger doctrine; and common or banal scenes a faire. By codifying the abstraction-filtration-comparison test, the Software Act would prevent courts from roaming away from the standard test, even if they ultimately arrive at the same conclusion.

---

318. *Id.*
319. 9 F.3d 823 (10th Cir. 1993).
320. 17 U.S.C. § 102(b) (ideas, procedures, processes, systems, methods of operation, concepts).
321. *Gates Rubber*, 9 F.3d at 836-38 (describing scenes a faire as stock elements of computer programs dictated by the standards or compatibility requirements of hardware, software, industry, or programming).
C. **The Abstraction Step**

The Abstraction Step is effective in dissecting computer programs because the test breaks down the program in a way that parallels typical program development.\(^{322}\) Expert witnesses are often necessary to provide the court with guidance in applying the abstraction test. The Tenth Circuit suggests six levels of generally declining abstraction: (1) the main purpose, (2) the program structure or architecture, (3) modules, (4) algorithms and data structures, (5) source code, and (6) object code.\(^{323}\) The Second Circuit described the levels of abstraction as ascending from object code, to source code, to parameter lists, to services required, to general outline.\(^{324}\)

The *Software Act* would force the applicant to do the initial work of conceptualization by openly announcing in the claims exactly what is to be protected. Requiring the applicant to be this forthright not only creates a bright line, delineating which claims are unavailable for expropriation without royalty, but narrowing the claims also makes them more likely to survive review. Narrow claims are more protectable than broad claims (which are likely to encompass prior art). The *Software Act* does not deprive the court of its ability to determine the level of abstraction at which to cast the elements. The *Software Act* would assist the court in reaching its abstraction decision by making the applicant draft the first pass. The court will use the drafted claims as guidance, but it will be free to recast the claims as appropriate. As Judge Keeton pointed out, a computer element should not be conceptualized along the scale of abstraction as a mathematical calculation; rather, it is more akin to a metaphorical human calculation, where a balancing of the "scales of justice" regularly involved in daily human affairs is involved.\(^{325}\)

D. **The Filtration Step**

A court having identified the elements claiming protection, should then filter out those elements which are not deserving of protection. The Second Circuit, in *Computer Associates v. Altai*,\(^{326}\) recognized that expressions fall outside of copyright protection where they are: already in the public domain; incidental to the idea; dictated by functional demand or the nature of other programs; or scenes a

\(^{322}\) *Id.* at 834.

\(^{323}\) *Id.* at 835.

\(^{324}\) Computer Assocs. v. Altai, 982 F.2d 693, 714 (2d Cir. 1992).


\(^{326}\) 982 F.2d 693 (2d Cir. 1992).
farae. The Software Act would also provide a filter for nonprofit, noncommercial use, more expansive and simple, which encompasses essentially all of the Copyright Act's fair use exceptions, as well as the complicated and detailed lobbyist group exemptions.

Filtration principles were explained further by the Ninth Circuit in Sega v. Accolade where Accolade reverse engineered software controlling the popular Sega Genesis game cartridges and console. Accolade disassembled three Sega cartridges, then experimented with reconstructed source code ("copies") to discover how to make game cartridges compatible with Sega's console. The Ninth Circuit held these intermediary copies to be fair use as a matter of law since disassembly was the only method providing access to the unprotected functional requirements of console compatibility. The court allowed Accolade to use a literal copy of a twenty-byte header initialization code necessary to make their games functionally compatible. The court could also have allowed such copying on the theory that was de minimis (a small or insignificant portion of the protected work), since a complete game consisted of 500,000 to 1,500,000 bytes. On the other hand, the court was possibly better off avoiding the de minimis theory, since it is vulnerable to the argument that the portion copied is quantitatively small but qualitatively large in economic value. The irony of Sega v. Accolade lies in the fact Sega refused the compensation proposed by Accolade in a proffered license. The result of Sega's attempt to enforce its copyright was a total loss of all compensation for its code. Accolade had sought the compatibility license before it even attempted to reverse engineer Sega's code. Sega refused unless it retained exclusive right to manufacture all game cartridges produced by Accolade. The litigation resulted in Accolade receiving for free that which it had offered to pay. This seems like an economically senseless solution for all parties. The money that Accolade spent on reverse engineering could have been better spent on compensating Sega with license fees in ex-

327. Id. at 706-10.
328. 17 U.S.C. § 107 (criticism, comment, news reporting, teaching, scholarship, research).
331. Id. at 1514-15.
332. Id. at 1514.
335. Id. at 1516.
336. Id. at 1514.
337. Id.
338. Id.
change for more information from Sega than its own trial-and-error research could provide. Likewise, Sega incurred the unnecessary and distraction of litigation, without offsetting license fee income and a permanent loss of future license fees.

The *Software Act* would resolve this situation differently, providing a better result for all parties. The *Software Act* would grant Accolade a right to reverse engineer the Sega's product for compatibility and pay Sega a reasonable compulsory license fee. Alternatively, since most firms will be economically encouraged to deposit their algorithms in a blanket licensing agency, Accolade could avoid the cost of reverse engineering, receive accurate compatibility information from the clearinghouse, and pay a reasonable fee to the clearinghouse. This system provides first comers like Sega with an economic incentive to develop the hardware game console in the first place, since Sega would know that second comers would pay a fee for compatibility. The second comers would have an incentive to produce such compatible products, since they will know in advance what the fee will be to sell such products. The public will benefit by finding a wider selection of compatible products from which to choose. Those second comers who do not want to pay the fee remain free to attempt to create their own standard with their own hardware.

E. Not Filters

Protecting software under the *Software Act* differs significantly from protecting software under the Copyright Act. The *Software Act* would accept as protectable any of the reasons listed in the Copyright Act for rejecting copyright protection (i.e., idea, procedure, process, system, method of operation, concept, principle, or discovery).³³⁹ While this may appear at first blush to be a radical view, it is not. The § 102(b) exclusions primarily enumerate precisely what cannot be protected by copyright. Yet, these exclusions are generally within the domain of patent. Since the *Software Act* is a patent statute it makes sense to embrace these as protectable. It is often these aspects of software that have precisely the troublesome hurdles that courts faced when trying to protect software via copyright. As discussed above, the very nature of software is that it has utility or usefulness. The purpose of the *Software Act* is to provide a way of protecting that very functionality and usefulness without having to pretend that software is something that it is not. Many of the algorithms that the *Software Act* would protect are actually in fact named "functions" in computer ter-

minology, while others are named "procedures" (meaning a subroutine, which may or may not accept arguments as input, and which do not return an output value upon conclusion).

The *Software Act* also rejects the merger doctrine because it is an unnecessary hinderance under this scheme.\(^{340}\) The merger doctrine seeks to avoid expropriation of ideas by denying a monopoly upon expressions of those ideas where only a few means of expression exist. The *Software Act* eliminates this conundrum by recognizing that the nature of software is such that the difference between an idea and an expression of an algorithm is very small.

Therefore, by implication, the Supreme Court's prohibition against protection for pure mathematical algorithms, as in *Benson*,\(^{341}\) is also unnecessary. The distinction between a pure mathematical algorithm and some other kind of algorithm is an impossible one to make. Neither the courts nor the public should have to suffer uncertainty in trying to determine which kind of algorithm they have. Under the *Software Act*, if Einstein were to create his famous \(E=mc^2\) the day after enactment of the Act, he could acquire a patent on the pure mathematical formula encompassed in his commercial software. Teller and Oppenheimer could pay him a fee for a few years to incorporate the formula in their own commercial bomb software. More likely, an algorithm clearinghouse would contract with Einstein to act as his licensing agent along with the unexpired Newton and Kessler algorithms, so a commercial physicist could pay an annual fee to license all of the necessary algorithms to blow his comrades to smithereens. University researchers would not have to be concerned about the fees, because their use would be noncommercial, nonprofit.

For those who still have their doubts about this scheme, there is also the consideration that the *Software Act* protects only two things: (1) literal software code in a machine-readable form, and (2) algorithms which meet all of the requirements under the definition of the term. The definition states that an algorithm must contain all the following elements: (1) a sequence of precisely defined computer program steps, (2) that will work for data of a specified range and type, (3) to produce answers of a specified quality, (4) that unambiguously solve a particular one of a specified general class of problems, (5) in a reasonably efficient finite time, and (6) using a reasonably efficient finite allotment of computational resources. Even algorithms meeting this narrow definition fail protection under the *Software Act* unless

\(^{340}\) *Software Act* § 234(e), *infra* Appendix.

\(^{341}\) *Gottschalk v. Benson*, 409 U.S. 63 (1972); *see also supra* notes 112-115 and accompanying text.
they also meet the test of patentability, requiring: (1) a new, (2) non-obvious inventive step, (3) used in commerce, (4) in computer software, and (5) in a new and useful configuration.

The above requirements do create a fairly bright-line test. A software creator can identify those portions of his code that he believes meet these requirements. While the Software Act will not circumvent the regular Patent Act questions about what is novel or what is obvious, this is well-trod ground. Under the Software Act, however, these questions do not even arise until the patent is challenged, then an examination occurs.

F. The Comparison Step and Remedies for Infringement

The Software Act requires the decision maker to look at the elements that have survived the filtration step. The court then determines whether there has been literal or nonliteral copying of those remaining elements. Literal copying of those elements that have survived filtration for de minimis use is particularly disfavored. Therefore, for literal copying, the court should impose substantial penalties, costs, or actual damages. Nonliteral copying of either literal elements or nonliteral elements has a much more lenient remedy. Infringed parties are entitled to be put in the same position as they would have been in if the infringer had taken a license from the beginning of the infringement. Infringers should also pay a reasonably small penalty. Otherwise, infringers would have no incentive to voluntarily take a directly negotiated or compulsory license, as they could pay the same amount after being caught, taken to court, and losing.

The Software Act also allows a court to transfer a patent to the rightful owner, rather than requiring such owner to file an independent application as the patent act requires. Invalidating someone else’s patent is not very helpful if one can not get the patent transferred into one’s own name. The transfer right will also harmonize our law with European law. If a court finds prior use in commerce of the algorithm, the court may order registration of the petite patent, relating back to the date of first proved use, upon the filing of an application.

VIII. ENACTMENT OF THE Software Act

A. Where to Put the Software Act – Copyright Act vs. Patent Act?

1. Amending the Copyright Act

It is not a new development to append other types of law onto the Copyright Act. Unfair competition law was tacked onto the Copyright
Act, via the Audio Home Recording Act of 1992,\textsuperscript{342} which establishes a compulsory royalty scheme on digital media used to make private (noncommercial) digital copies. Likewise, petite-type patent law was tacked onto the Copyright Act via the Semiconductor Chip Protection Act of 1984.\textsuperscript{343} It would, therefore, be reasonable to append the Software Act’s petite patent law onto the Copyright Act, where software currently receives its greatest protection anyway. The Software Act, as a part of the Copyright Act, would be compatible with the Berne Convention by providing national treatment, as well as granting the same protection to both domestic and Member inventors. The Berne Convention leaves it to each country to define “literary and artistic works” for itself.\textsuperscript{344} Since Berne provides that “literary and artistic works” include (nonexclusively) “every production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression,”\textsuperscript{345} then software algorithms must certainly be included within this broad scope. Berne does not require a particular level of originality, so the Software Act could potentially include algorithms lacking originality within its definition of “literary and artistic works.” The Software Act, however, may seem at odds with GATT-TRIPs Article 10.1, which requires that Berne member countries protect computer software as “literary works,” thus granting Berne Article 7(1) lifetime-plus-50-years protection. The Software Act skirts this problem by allowing just the human-readable printed software to receive standard Berne life-plus-fifty term, while restricting machine-readable magnetic software to its short term protection. This should be acceptable, since Berne allows member countries to provide new protections outside the Convention’s scope,\textsuperscript{346} which grant less than Berne’s normal minimum rights.

Likewise, even limited protection of ideas is outside the scope of the Berne Convention, which exudes a general abhorrence of monopolies removing ideas from the public domain. The Software Act, however, does not remove formulas of nature and mathematical algorithms entirely from the public domain because it only protects specified commercial embodiments, leaving all other embodiments and non-commercial use in the public domain. Even where the Software Act does protect an idea, it allows access to everyone for a reasonable fee. Furthermore, the Software Act’s short-term duration only requires rea-

\textsuperscript{344} Berne Convention, arts. 2(2), 2(4).
\textsuperscript{345} Id. art. 2(1).
\textsuperscript{346} Id. art. 19.
reasonable fees for a period long enough to compensate the inventor for
the commercial release of the idea into the world.

2. Amending the Patent Act

It is more sensible to append the Software Act to the Patent Act, rather than the Copyright Act. The Software Act would fit in well among the other petite patents, such as, design patents\textsuperscript{347} and plant patents.\textsuperscript{348} More importantly, the Software Act begins a series of steps necessary to reform the whole Patent Act into harmony with the rest of the world. One might even argue that amending the Software Act to the Patent Act is consistent with GATT-TRIPs Agreement art. 27 (1), which requires WTO (World Trade Organization) countries to make patent protection available to inventions in "all fields of technology."\textsuperscript{349} If integrated into the Patent Act, then international obligations under the Paris Convention for the Protection of Industrial Property of 1883\textsuperscript{350} would be triggered.

The Software Act is not only consistent with, but is better than, the Paris Convention, which permits member countries to grant royalty-free nonexclusive compulsory licenses if a patent is not "worked."\textsuperscript{351} The Software Act, consistent with TRIPs Article 31, requires compensation to the patentee and judicial review of both the license grant and the amount of compensation.\textsuperscript{352} The Paris Convention obviates the need to file simultaneously in every country where one desires patent protection, even though the territorial grant requires a separate filing in each jurisdiction.\textsuperscript{353} The Paris Convention provides that these separate applications do not have to be filed simultaneously, generally allowing later filings to inherit the initial date the application was first filed in a member country.\textsuperscript{354} However, the Paris Convention does not go far enough in providing a single worldwide site where a member may file a single application and receive protection in each country. This Software Act proposes a unified Algorithm Office where filings may be made to grant protection in every country which agrees to this concept.

\textsuperscript{347} 35 U.S.C. §§ 171-173.
\textsuperscript{348} 35 U.S.C. §§ 161-164.
\textsuperscript{349} General Agreement on Tariffs and Trade, Dec. 15, 1993, 33 I.L.M. 81, 93 (emphasis added).
\textsuperscript{351} Paris Convention, supra note 350, art. 5(2).
\textsuperscript{352} See Software Act § 228, infra Appendix.
\textsuperscript{353} Paris Convention, supra note 350, art. 4(A).
\textsuperscript{354} Id.
However, new treaties with other countries would probably be needed anyway, because the international conventions do not cover this subject matter. The reluctance of GATT countries to agree to copyright protection for something as venerable as sound recordings, strongly suggests that they might shy away from agreeing to protect something as nebulous (yet valuable) as algorithms. Even a bilateral agreement with a single country, namely Japan, would relieve much of the U.S.'s international trade concerns. The Paris Convention and Berne Convention, art. 20 permit special agreements among member countries granting more extensive intellectual property rights, as long as such special agreements do not contravene the Conventions.

B. Beyond the Patent Cooperation Treaty — The Internet Convention

The Software Act should not be limited to the United States. Software today often needs success in two or three world markets to survive. An international convention similar to the Berne or Paris Conventions, including technical as well as legal experts, should be convened to implement worldwide algorithm petite patent uniform rules and enforcement. Such a software convention would be uniquely situated to draw expert commentary from every corner of the planet by the people most affected by the outcome, via the Internet. Hence, the convention might be called the “Internet Convention.” Separate Internet channels for official communication might be provided, along with Usenet discussion groups for those with interested commentary. The draft act presented in this paper might serve as a kernel for the debate. The United States might achieve international harmony by working in advance with other countries, rather than cramming a fait accompli solution down their throats. An international electronic discussion might lead to the end of the Japanese policy of rejecting software patent applications for failing to apply scientific laws or clearly defined applications.

The Software Act seeks consistency with the Patent Cooperation Treaty (PCT), yet provides additional protections not implemented in

357. Id.
358. Id.
the PCT.\textsuperscript{360} The PCT simplifies filing by allowing a single, standard-
ized, international application form to be filed in a local patent office.\textsuperscript{361} The applicant lists the countries where a patent is sought, includes translations and registration fees for each country, and the WIPO forwards the application to each country, after an international search is performed.\textsuperscript{362} The PCT does not harmonize substantive patent laws, it merely simplifies multinational filing procedures.\textsuperscript{363} The Software Act envisions true one-stop filing for uniglobal, multinational protection. A single filing in any member country’s patent office would provide the protection in all other member countries. This type of protection was considered but not implemented in the Patent Coop-
eration Treaty. Multinational markets are needed in today’s shrinking world to succeed commercially in the software business. However, PCT membership is limited to countries that are already members of the Paris Convention,\textsuperscript{364} which explains why as of January 1, 1988, there were only 40 parties to the PCT.\textsuperscript{365} The Software Act should not be limited to Paris Convention countries. It would be in the best interest of both developed and lesser developed countries for the LCD’s to join the Software Convention out of their own self interest. Developed countries receive legal access to markets where software piracy is the current standard; LCD’s get compulsory licenses that legally allow them to use foreign software algorithms, as well as automatic protection for their own works. The Software Act eliminates the lengthy and expensive search process, since the first to file automatically receives priority if there are reasons for invalidating it as prior art.

If the world is to move toward eventual true harmony of utility patents, it is sensible to start down this path with petite patents. Let petite patents break the ground and serve as a beta test to work out the problems of implementing such an infrastructure. By definition, the mistakes, or growing pains, of a petite patent would be less than they would be for a grand patent. After the path has been hewn, utility patents may follow. It is particularly sensible to begin unifiling with software patents, since the world (in the Internet Age) is rapidly be-

\begin{thebibliography}{9}
\bibitem{361} \textit{id.} art. 11.
\bibitem{362} \textit{id.}, art. 3.
\bibitem{364} \textit{id.}; see also Patent Cooperation Treaty, \textit{supra} note 360, art. 62 (on becoming a party to the treaty).
\bibitem{365} Greenwald & Levy, \textit{supra} note 363, at 831-32. There were 97 parties to the Paris Convention as of January 1, 1988. \textit{id.} at 677-78.
\end{thebibliography}
coming interconnected anyway. Software patents are particularly well suited to submission by electronic means. Generally, software already exists as text in a computer. Although software may contain algorithms or nonliteral elements which are not text, software does not generally require the "drawing" that most machine-type inventions do. The average programmer would be able to fill out a standardized form found directly in the text of the software, much like the way lay people currently fill out Copyright Form TX. The petite patent application form would then be submitted electronically (along with electronic payment) from any computer in the world connected to a telephone. While it may be somewhat ethnocentric to require filing in the English language,\textsuperscript{366} English is often considered the international language of business, and even more so, the language of computing.

IX. CONCLUSION – OPENING THE DOOR TO THE SOFTWARE ACT

The Court of Customs and Patent Appeals described the Patent Act's subject matter requirements\textsuperscript{367} as doors through which every invention must pass to be patent eligible.\textsuperscript{368} The Software Act seeks not merely to open the door to software algorithm patents, but rather it creates a new threshold. Much like the small door confronting the oversized Alice in Wonderland, the Software Act door is designed to fit only inventions as small as software algorithms. Yet, it is an important door. It is necessary to give shelter to valuable software algorithms, since such protections are not available in the standard copyright and patent houses. Copyright law is not the correct form of protection for computer programs, since software is not truly literature. Few people read software source code for its prose or its beauty as they would literature. Ten thousand lines of machine-readable code does not truly represent beautiful art work. Few people read a computer program for its plot. Rather, people buy software for its usefulness. Software is a toaster – a tool used to get a job done – and the law should treat it as such.

Moreover, patent law fails to protect algorithms well – or software at all. Software patents are limited to the terms "process" and "machine."\textsuperscript{369} Since a process generally requires transformation of a physical raw material from one state into another, and a machine requires newly configured physical hardware, cramming algorithms into these definitions is difficult and unnatural. Patent process claims

\textsuperscript{366}. See infra Appendix § 224.
\textsuperscript{368}. In re Bergy, 596 F.2d 952, 960 (C.C.P.A. 1979).
for algorithms must be loaded with the buzz words of patentability that have been approved by courts.\textsuperscript{370} Why should it be necessary to wink and nudge at the PTO, pretending that algorithms are something they are not? The open secret of recasting algorithms as processes is reminiscent of the movie \textit{Casablanca}, where Claude Raines is "shocked, just shocked, to find out that there's gambling going on here."\textsuperscript{371} The practical requirement of patent-by-incantation is not only deceptive, rather it provides a level of protection that is too uncertain. Likewise, the Federal Circuit's tortured interpretation of the term "machine" provides inherently illogical support for patentability.\textsuperscript{372} The meaning of "process" and "machine" must be expanded\textsuperscript{373} to protect algorithms. Even amending §101 to explicitly enumerate algorithms as patentable would be inappropriate because regular patent rights are too extensive, and the regular patent total exclusion is too broad for software. While some might argue that the current system is sufficient since it does protect some algorithms from infringement, the result of \textit{Stac Eletronics v. Microsoft Corp.}\textsuperscript{374} illustrates that even under the best-case scenario the current system resembles what the \textit{Software Act} would provide to the average scenario at a much lower cost. Software algorithms do not deserve the theoretical full strength patent protection they now enjoy. The current rights are theoretical because it is unlikely that most small software algorithm inventors (who the law should seek to protect the most) will be able to acquire and enforce those rights.

\textsuperscript{370} Patent lawyer Michael H. Jester describes the "Section 101 game" this way:
Over the past ten years, our patent law firm has prepared literally hundreds of U.S. patent applications on programmed digital computers and software, the vast majority of which have, or will, issue as U.S. patents. It has always been necessary to package and present these particular inventions in a manner specifically designed to avoid a rejection under 35 U.S.C. Section 101. The poor patent lawyer who originally naively claimed "a computer program comprising..." must redraft the claims to cast the invention in terms of a process for doing X, Y, Z steps in a particular computer system environment, the steps being studiously drafted to appear unalgorithm like.


\textsuperscript{371} \textit{CASABLANCA} (Metro-Goldwyn-Mayer 1942).

\textsuperscript{372} See supra part IV.D.

\textsuperscript{373} See Paula Samuelson, Benson Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions, 39 EMORY L.J. 1025, 1142 (1990) (suggesting that algorithms can be thought to be "shoehorned" into the terms "process" and "machine"); see also, Richard Stern, The Bundle of Rights Suited to New Technology, 47 U. Pitt. L. Rev. 1229, 1264-65 (1986) (proposing to eliminate the crisis resulting from each newly invented technology out dating intellectual property law, by enacting skeleton legislation which the PTO may activate to arm these new industrial properties with an off-the-shelf arsenal of protective tools).

\textsuperscript{374} CV 93-413-ER, Feb. 23, 1994 (C.D. Cal 1994); see also supra part IV.E.3.a.
Opposition to this proposal might also be found among those arguing that the establishment of petite patent for software will increase the administrative burden on small software developers who would have to file patent applications. This problem is addressed via inexpensive fees, central claims (which should be simpler to write than peripheral claims), and the first-to-file method. Compulsory license income should offset the cost of the application. Furthermore, the benefit of added certainty of protection should outweigh the additional cost of unknown litigation potential. Of course, an author could completely avoid the cost of filing, if he is willing to pay compulsory royalties to another person who files first. This proposal offers rights which developers previously did not have. While it may increase a vendor's patent application budget, the supplemental rights are designed to decrease the vendor's litigation budget. The mere addition of clarifying statutory definitions and more detailed bright lines should eliminate the need for many suits before they ever begin.

The Software Act also provides a new weaker petite patent designed to match the nature of software itself and the needs of its market. Cheap, quick, and short protection would utilize central claims that are examined only after infringement takes place. Petite patents will be available only to commercial uses of the software algorithm, thus eliminating the need for research exceptions in patent claims. Central claims (rather than peripheral claims) benefit society, while narrow claims leave room for future software to solve peripheral problems in parts of the economy that were commercially ignored by the patentee. Eliminating the patent reservation system prevents claims covering every possible use of the algorithm in a computer, unless the vendor actually commercially performs every possible use. The Software Act presumes automatic claim to the literal code of software, and provides substantial penalties for literal (letter-for-letter, bit-for-bit) copying of elements which are not filtered out in the abstraction-filtration-comparison test. Blanket licensing of software algorithms administered by large rights societies, such as, ASCAP or the Software Protection Association, would provide the same kind of benefits to the software industry that the music industry currently enjoys (and for the same reasons). Anyone would be able to use the algorithm at a fee. Yet, inventors will still be encouraged to disclose their ideas which otherwise might have been locked up in regular patent fortresses or hidden under trade secret law. This compulsory licensing alternative will be faster and cheaper than today's ultimate punishment of court injunction and seizure (which was not even effective in the Stac case, since Stac — financially — was forced to settle).
The ideas presented in this proposal merit consideration at large. As an alternative to the Second Circuit's call for a CONTU II second round of federal computer law overhaul, a more global solution needs to be sought in the form of an international convention similar to the Berne and Paris Conventions. The computer industry is perfectly situated to draw upon the genius of not only legal scholars but computer technical scholars. This paper proposes that the new convention be named the "Internet Convention," as this could provide an unparalleled opportunity for immediate substantial input from experts all over the world. The Software Act proposes a small ("petite") experiment: stepping U.S. law into closer harmony with the rest of the world before attempting to amend the regular Patent Act. Ultimately, the goal of universal global electronic filing is the only sensible solution in a world comprised of global markets.

375. See supra note 42 and accompanying text.
APPENDIX—THE MODEL SOFTWARE PETITE PATENT ACT
PATENT ACT
35 U.S.C. §§ 221–237

CHAPTER 19—SOFTWARE ALGORITHM PATENTS

§ 221. Short Title
(a) This Act shall be known and may be cited as the “Software Act.”

§ 222. Patents for Software Algorithms
(a) Whoever invents an algorithm may obtain a patent therefor, subject to the conditions and requirements of this title, provided that the algorithm contains all the following elements:
   (1) a new
   (2) non-obvious inventive step
   (3) used in commerce
   (4) in computer software
   (5) in a new and useful configuration.
(b) The provisions of this title relating to patents for inventions shall apply to patents for software algorithms, except as otherwise provided.

§ 223. Definitions - Except as otherwise provided, the following terms and their variant forms for this Chapter have the following meanings:
(a) “Abandoned” means:
   (1) express written disclaimer of claims by patent owner; or
   (2) the claimed element’s use in commerce has been discontinued with intent not to resume such use. Intent not to resume may be inferred from circumstances. Nonuse for two consecutive years shall be prima facie evidence of abandonment.
(b) “Algorithm” - The required elements of a claimed algorithm are:
   (1) a sequence of precisely defined computer program steps
   (2) that will work for data of a specified range and type
   (3) to produce answers of a specified quality
   (4) that unambiguously solve a particular one of a specified general class of problems
(5) in a reasonably efficient finite time 
(6) using a reasonably efficient finite allotment of 
computational resources.

(c) “Claimed Element” means literal or nonliteral 
algorithm element claimed. The source code and 
object code are automatically claimed as literal 
elements.

(d) “Literal element” means the machine-independent 
source code, and the machine-dependant object 
code, embodying the claimed algorithms.

(e) “Nonliteral element” means algorithms or 
expressions other than those expressed as source 
code or object code.

(f) “Canned computer software” means prewritten or 
standard software computer software designed for 
and distributed “as is” for multiple persons who 
can use it without modifying its code and that is 
not otherwise considered custom computer 
software. It is off-the-shelf software not created for 
a particular customer and sold to many different 
customers without substantial modification by the 
vendor for the customer. There is a presumption 
that software is canned software where it costs less 
than $10,000, or where sales tax is charged on the 
software.

(g) “Custom computer software” means computer 
software that is designed for a single person’s or a 
small group of persons’ specific needs. “Custom 
computer software” includes modifications to 
canned computer software and can be developed in-
house by the user, by outside developers, or by 
both. A group of four or more persons is presumed 
not to be a small group of persons for the purposes 
of this subsection unless each of the persons is 
affiliated through common control and ownership. 
The department may by rule provide a definition of 
small group and affiliates consistent with this 
subsection. “Custom computer software” means 
software which:
(1) requires the software vendor’s analysis of the 
customer’s requirements;
(2) for the vendor to substantially select, adapt, 
modify, prepare, or develop the software;
(3) pursuant to the customer's special order;
(4) exclusively for a specific user; and
(5) the software is of an original, one-of-a-kind nature.

Example 1: A software vendor offers for sale a prewritten program which can be used in several computer models. Prior to operation, instructions must be added by the vendor which specify the particular computer model in which the program will be utilized.

Example 2: An existing program is selected for modification by the vendor, and the vendor makes a substantial modification of that program so that it may be used in the customer's specific make, model, and version of hardware device or other software.

(h) "Commerce" means, where not inconsistent with this chapter, commerce as specified and interpreted under Trademark Act § 45, 15 U.S.C. § 1127.

(i) "Commercial computer software" means computer software which is used regularly for other than Government purposes and is sold, licensed, or leased in significant quantities to the general public at established market or catalog prices.

(j) "Computer" means an electronic device that:
(1) is programmable;
(2) performs logical, arithmetic, and memory functions by the electronic manipulation of data;
(3) supplies the results of this data manipulation; and
(4) the device has input, output, processing, storage, and communication facilities connected to the device or available to the device via a computer network.

(k) “Computer equipment” or “hardware” mean central processing units, microprocessors, data storage and other computer memory devices, and computer terminals or similar devices.

(l) “Computer network” means two or more computer systems connected so as to permit the exchange or sharing of data between or among them.
(m) “Computer program” and “computer software” are interchangeable terms.

(1) “Computer program” means computer code which, when executed by a computer system, causes the computer system to perform computer services to solve a specific problem.

(2) Computer programs may be either machine-independent or machine-dependent.

(i) “Computer Code” means a list of literal text instructions or statements.

(ii) “Source Code” means the human-readable machine-independent computer code written in a computer programming language.


(iv) “Compile” or “Assemble” means to convert source code into object code.

(v) “Decompile” or “Disassemble” means to convert object code to source code.

(3) Computer programs may be general-purpose in nature or be designed to satisfy the requirements of a particular user.

(4) Computer programs are hereby deemed to be “useful articles.”

(5) Contrary to any state sales tax definition, the term “computer program” in this chapter does not mean:

(i) Databases; nor

(ii) Computer software documentation.

(n) “Computer software documentation” means human-readable literature, data, or computer program code listings or printouts, which:

(1) document the design or details of the software; or

(2) explains the capabilities of the software; or

(3) provides operating instructions for using the software to obtain desired results from a computer.

(o) “Computer services” means data input, data output, data processing, or data storage by or in a computer system or computer network.
"Computer system" means computer equipment or hardware connected together and operating under the control of one or more computer programs.

"Data" means numeric, textual, graphic, or other information or records which are created, stored, or retrievable by a computer or electronic means. The term "data" does not include computer software, which is the tool used to create, store, or retrieve data.

"Database" means a collection of data organized in a strictly prescribed form and format capable of being accessed or managed with the aid of computer software, but that does not itself have the capacity to direct the operation of a computer system or other machinery or equipment. Databases are protectable, if at all, by the Copyright Act, not the Software Act.

"Embedded software" is computer software that resides permanently on some internal memory device in a computer system or other machinery or equipment, that is not removable in the ordinary course of operation, and that is of a type necessary for the routine operation of the computer system or other machinery or equipment.

"Member country" means any country which (bolt-on option) [belongs to the Paris Convention.]
(sui generis option) [joins the Software Convention, gives effect to the provisions of this Act, and offers reciprocal protection to nationals in other Member Countries.]

"Public domain computer software" means software upon which the petite patent has expired or software which has been abandoned.

"Publish electronically" means providing the right or ability to retrieve a document or data via the Internet, an Electronic Bulletin Board System, a commercial data service, or any other telecommunications or storage-and-retrieval media.
§ 224. Application

(a) Who May Be an Applicant. Only a national of Member Countries may be an applicant under this Act.

(b) Written application shall be made in the English language by the inventor, or inventor's authorized agent, to the Commissioner. Such applications shall include:

(1) A specification or description which shall disclose the invention in a manner sufficiently clear and complete for the invention to be carried out by a person skilled in the art; and

(2) One or more claims which, independent of the specification, clearly and concisely point out with particularity the distinct required elements of each algorithm.

(i) Whenever appropriate, claims shall contain:
(A) a statement indicating those technical features of the invention which are necessary for the definition of the claimed subject matter but which, in combination, are part of the prior art,

(B) a characterizing portion stating concisely the technical features which, in combination with the features stated under (A), it desires to protect.

(ii) Any claim which includes all the features of one or more other claims (claim in dependent form, hereinafter "dependent claim") shall do so by a reference, to the other claim or claims and shall then state the additional features claimed. "Multiple Dependent Claims" (dependent claim which refers to more than one other claim) are not allowed.

(iii) If a court finds that any of these requirements are not met, the court may invalidate the claim, or the court may equitably save the claim from invalidity by filling in or recasting the claim to meet the requirements; and

(3) Verification by the applicant, or applicant's authorized agent, specifying:

(i) the applicant's domicile and citizenship;

(ii) the date of applicant's first commercial use of each algorithm, or alternatively applicant's good faith bona fide intention to use each algorithm in commerce;

(iii) the goods or services in connection with which the algorithm is used; mode or manner in which the algorithm is used in connection with such goods or services; and
(4) a statement that the person making the verification believes himself, or the principal on whose behalf he makes the verification, to be the owner of the algorithm, and that no other natural or artificial person or business entity has the right to use such algorithm; and

(5) the fee prescribed by the Patent and Trademark Office.

(c) Optional Standard Application Form.

(1) The Commissioner shall prescribe and make available free of charge upon request an optional standard application form which may be used to fulfill the requirements of this section.

(2) The optional application standard form shall be published on paper and published electronically.

(d) Optional Electronic Application Method.

(1) The Commissioner shall prescribe regulations on the optional method of submitting an application and payment electronically to the Patent and Trademark Office.

(e) Unified Office. Applications may be filed from any Member Country.

§ 225. Examination, Issue, and Effect of Patent

(a) The Commissioner shall not examine the invention until challenge has been made.

(b) The Commissioner shall examine the application, and if it appears that the requirements of application have been met, the Commissioner shall issue a patent.

(c) Software patents shall have a unitary character, having equal effect throughout the territories of Member Countries and may only be granted, transferred, revoked, or allowed to lapse in respect of the whole of such territories.

§ 226. Registration and Publication

(a) Issued patents shall be published in a Software Patent Registry.
(b) Documents may be recorded in the Software Patent Registry to establish a public record of licenses, documents transferring ownership of some or all rights under this act, security interests in, and other legal documents governing the relationship between software patent owners and persons associated with the commercial dissemination or use of such software.

(c) Documents may be submitted in machine-readable format, as the Commissioner may establish by regulation.

(d) The date of recordation is the date when all of the elements required for recordation, including the prescribed fee have been received in the Patent and Trademark Office. The Commissioner shall send the applicant a certificate of record as evidence of recordation. The Commissioner may destroy the submission after preparing suitable copies.

(e) The Commissioner may publish electronically the register and application information.

(f) The Commissioner may impose a filing fee by regulation.

§ 227. Challenge of Algorithm Patent
(a) Any person who believes that he is, or will be, damaged by the patent may challenge the patent at any time.

(b) A challenge may be cited as an affirmative defense or counterclaim to an infringement action.

(c) A challenge may be to the Patent and Trademark Office upon payment of the prescribed fee.

(d) The challenge shall state the facts supporting the grounds upon which it is based.

§ 228. Effect of Challenge
(a) A court may invalidate any or all claims of a patent in its determination of an infringement suit.

(b) A court may order that:
   (1) an infringer pay compulsory license fees, plus costs, plus a penalty to the patent owner; or
   (2) the Commissioner register or transfer a patent, upon payment of registration fees, to a person who has proved that he is entitled to it; or
 invalidated claimed elements drop into the 
 public domain.

§ 229. Granted Rights
(a) In the case of software patents, the grant shall be 
the right to exclude others from making, using, 
selling, offering for sale, or importing software 
containing patentee’s claimed elements without a 
license from patentee.
(b) The right to “use” includes the act of loading 
software from a storage medium into computer 
memory and executing the software on a computer.
(c) The right to “import” excludes nonprofit, 
noncommercial importation of no more than one 
copy by one person at one time.
(d) The Secretary of the Treasury and the U.S. Postal 
Service shall separately or jointly make regulations 
for the enforcement of a patent owner’s right to 
prohibit parallel importation.

§ 230. Term of Software Algorithm Patent
(a) Patents for software algorithms shall be granted for 
the term of ten years from the date of filing of the 
patent application.

§ 231. Test of Infringement
(a) Whoever without authority exercises the granted 
rights within any member country during the term 
of the patent, infringes the patent.
(b) Infringement shall be determined using the 
abstraction-filtration-comparison test.

§ 232. Abstraction Step
(a) Abstraction Step. The court may accept the level 
of abstraction as cast in the claims as patented, or 
may recast the claims sua sponte or upon motion 
based upon its determination of the level of 
abstraction.

§ 233. Filtration Step
(a) Failure to meet any element or requirement of 
patentability under this chapter.
(b) A claimed element is invalidated where it claims 
prior art.
(1) An element becomes prior art where before the 
applicant’s filing date it has been:
(i) patented;
(ii) described in a patent application with an earlier filing date;
(iii) described in publications available to the public;
(iv) used in commerce; or
(v) is in the public domain.

(2) An element is not prior art where during the year preceding the filing date:
(i) It is claimed by the inventor or his successor in interest;
(ii) A competent national office that in contravention of the applicable norms publishes the content of the patent application filed by the inventor or his successor in interest;
(iii) A third party obtained the information directly or indirectly from the inventor or his successor in interest;
(iv) An evident abuse occurred vis-à-vis the applicant or his successor in interest; or
(v) Applicant publicly exhibited the software because it was necessary to make it public in order to continue its development. Applicant shall state at the time of filing the application that the invention has in fact been exhibited.

(c) Anti-Vaporware Provision

(1) "Vaporware" means:
(i) the public announcement of a commercial computer software product
(ii) before it is commercially developed
(iii) with the sole intent of causing consumers not to purchase
(iv) a competitor's commercially developed product.

(2) "Commercially Developed" means that the software exists, is workable, and is at a stage where it could be sold on the commercial market.
(3) "Workable" means the software has been analyzed or tested sufficiently to demonstrate to a reasonable person skilled in the applicable art that there is a high probability that it will operate as intended.

(4) Vaporware claims shall be invalidated.

(5) Vaporware claims shall be prima facie evidence of an unfair or deceptive act or practice under Federal Trade Commission Act section 45 [15 U.S.C. 45].

(d) Peripheral Claiming. A claim shall be invalidated to the extent that it exceeds the scope of its commercial embodiment.

(e) Anti-Orphanware Provision. A claim shall be invalidated, and shall drop into the public domain, when the claim has been abandoned.

(f) Prior Use. A software patent shall have no effect against any person who proves prior use, or substantial arrangements necessary for such use, in a Member Country prior to filing date of the application. This right can only be inherited or transferred with the business.

(g) Exhaustion.
   (1) The patent owner's right to use, sell, or offer for sale (but not the right to import) canned software shall be exhausted with respect to a particular copy following first sale of that copy.
   (2) Software licensing conditions limiting the right to resell (but not import) or use canned software shall be voidable.
   (3) The software customer may authorize agents to exercise the exhausted rights on his behalf, including the right to use the software.376

(h) Public Domain. Elements fully within the public domain do not infringe.

(i) Scenes a Faire. Stock elements commonly found in a general class of computer programs do not infringe.

(j) De Minimis Use. Literal copying of a small and insignificant portion of the work does not infringe.

(k) Nonprofit, Noncommercial Use.
(l) Reverse Engineering and Decompilation Right. A software patent shall not affect a user's right to learn the internal design of a program by studying and experimenting with disassembled or decompiled portions of, or all of, the program code.

(m) Equitable Defenses. Elements may be filtered out or invalidated, based upon equitable principles including but not limited to: fraud, inequitable conduct, laches, promissory estoppel, unclean hands, unfair competition, or unjust enrichment.

§ 234. Not Filters. An infringer may be liable to pay a reasonable royalty upon, and a claim shall not be invalidated solely on the grounds that it is:
(a) A method of doing business;
(b) A mathematical formula;
(c) Any reason listed in Copyright Act § 102(b) (idea, procedure, process, system, method of operation, concept, principle, or discovery),
(d) An element dictated by petite-patented compatibility requirements of computer hardware, software, industry, or programming standards; or
(e) It preempts all or most other means or expressions of its idea.

§ 235. Comparison Step
(a) The claimed elements remaining after filtration shall be compared to the accused elements.
(b) If the court determines that the remaining elements have been literally copied (letter-for-letter or bit-for-bit), then the court shall impose substantial penalties or actual damages, plus costs.
(c) If the court determines that the remaining elements have been nonliterally copied, then the court shall impose a compulsory nonexclusive license, plus a reasonable penalty. Factors of nonliteral copying, include, but are not limited to:
(1) substantial similarity in both ideas and expression;
(2) error common to both the accuser's and accusee's work; or

378. Merger doctrine does not apply.
the accused work is designed or optimized substantially in light of the accuser’s hardware or software environment rather than the accusee’s

§ 236. Compulsory Nonexclusive License
(a) Definitions — for this section:
(1) “Licensed product” means products or services embodying or made in accordance with one or more claims of a software patent.
(2) “Improvement” means any modification to a patent, which if unlicensed, would infringe one or more claims of a software patent.
(3) “Granted rights” means the right to make, use, offer for sale, sell, and import Licensed Products.
(4) “Manufactured and sold” means, for the purposes of computing royalties, when billed out, or when shipped, or when paid for, or when otherwise disposed of, whichever shall occur first.

(b) Any person may obtain a compulsory license to commercially exercise the granted rights of the patent, unless such license would be contrary to the public interest.

(c) A compulsory licensee is not required by this section to assign future improvements of the licensed patents to the licensor. Compulsory licensee is entitled to file application for patent in licensee’s own name (or licensee’s principal’s name) on the derivative species improvement of the original genus patent.

(d) Noncompete clauses in compulsory licenses shall be voidable.

(e) Notice of Intention to Obtain Compulsory License.
(1) Any person may obtain a compulsory license under this section to exercise the granted rights by serving notice to do so on the patent owner and the Patent and Trademark Office within 30 days after such exercise of granted rights. The notice shall comply in form, content, and manner of service, with requirements that the Commissioner shall prescribe by regulation.
(2) Failure to serve or file the notice required by this section forecloses the possibility of a compulsory license, and in the absence of a negotiated license, renders the making, using, or selling of the patent an act of infringement subject to the remedies provided under this title.

(f) Royalty Payable Under Compulsory License

(1) To be entitled to receive royalties under a compulsory license, the patent owner must be identified in the registration or other public records of the Patent and Trademark Office. The patent owner is entitled to royalties for use of the patent occurring only after being so identified and is not entitled to recover for the exercise of the granted rights prior to the registration date.

(2) The compulsory licensee shall pay by the royalty due date a reasonable royalty equalling the royalty base multiplied by the royalty rate multiplied by the royalty index.

(i) The Commissioner shall establish by regulation royalty bases, royalty rates, and royalty indexes.

(ii) The royalty base may differ for different classes of claims, to be the unit quantity or the net sales of licensed product manufactured and sold, metered, or any other reasonable base.

(iii) The royalty rate shall be an amount determined by the Commissioner to provide a reasonable royalty.

(iv) The royalty index shall be a factor to adjust for economic inflation or deflation.

(v) The royalty due date shall be the twentieth day of each month for royalties for the month next preceding, unless the Commissioner determines another reasonable date.
(3) The Commissioner shall prescribe regulations under which detailed cumulative annual statements of account, certified by a certified public accountant, shall be filed for every compulsory license under this section. The monthly and annual statements of account shall prescribe the form, content, and manner of certification.

(4) If the patent owner does not receive payments and statements when due, the owner may give written notice to the licensee that, unless the default is remedied within 30 days from the date of the notice, the compulsory license will be automatically terminated. Such termination renders licensee’s further exercise of the granted rights act of infringement subject to the remedies under this title.

§ 237. Blanket Licensing Alternative

(a) Software rights societies or organizations may act as clearinghouses administering nonexclusive licenses to users and paying royalties to petite patent owners. If a software rights society or organization and a putative licensee are unable to agree on a fee within 60 days, the putative licensee may apply to the District Court for a determination of a reasonable fee, with the rights society or organization having the burden of proving reasonableness.

(b) Exemption from Antitrust Laws. The antitrust laws shall not apply to software licensing agreements related to software rights societies or organizations, provided that such agreements allow the patent owner to opt out of the blanket license via a directly negotiated license with a licensee.