Computer Programming and the Automation of Invention: A Case for Software Patent Reform

Robert Plotkin, Esq.[1]
Robert Plotkin, P.C.
rplotkin@rplotkin.com

I. Introduction: Putting Software First

The appropriate form of intellectual property protection for software has long been a subject of intense debate.[2] Although every participant in the debate appears to agree that software has unique properties that are relevant to the form of intellectual property protection that should be afforded to software,[3] there is little agreement about precisely *what* is different about software or about how intellectual property law should apply to software.

---

[1] Mr. Plotkin is a solo practitioner with a practice emphasizing intellectual property protection for computer hardware and software. The author gratefully acknowledges the research assistance provided by Meleena Bowers, Esq. in the preparation of this paper. The author also wishes to thank Melissa Hoffer for the insightful and invaluable feedback she provided on the topics addressed herein. The author is also grateful for feedback provided by Professors Harold Abelson, Archon Fung, and Michael Martin, and Attorneys Peter Gordon and Larry Kolodney. Finally, the author wishes to thank Professor James Moor, the Nelson A. Rockefeller Center at Dartmouth College, and the Dartmouth Lawyers Association (DLA) for providing the author with an opportunity to present some of the ideas contained herein as part of the DLA's Law Speaker Series in the spring of 2003.

[2] The debate dates back at least to 1965, when President Johnson commissioned a comprehensive study of the United States patent system. The Commission explored a wide range of pressing issues facing the patent system and recommended, among other things, that "no patents on . . . computer programs" be issued. 1966 Report of the President's Comm'n on the Patent Sys. 1. The Supreme Court first wrestled with software patentability in Gottschalk v. Benson, 409 U.S. 63 (1972). For an overview of the history of software patents, *see*, *e.g.*, GREGORY A. STOBBS, SOFTWARE PATENTS 1-46 (2d ed. 2000). For information on some of the history of copyright protection for software, *see, e.g.,* Pamela Samuelson, *CONTU Revisited: The Case Against Copyright Protection for Computer Programs in Machine-Readable Form*, 1984 DUKE L.J. 663 (1984).

[3] *See*, *e.g.*, James R. Goodman et al., *Toward a Fact-Based Standard for Determining Whether Programmed Computers are Patentable Subject Matter: The Scientific Wisdom of* Alappat *and Ignorance of* Trovato, 77 J. PAT. & TRADEMARK OFF. SOC'Y 353 (1995) (arguing that the particular manner in which computers are programmed using software distinguishes software from other technologies in a way that is relevant to patent law); *see also* Symposium*: Toward a Third Intellectual Property Paradigm: A Manifesto Concerning the Legal Protection of Computer Programs*, 94 COLUM. L. REV. 2308, 2327 (1994) (arguing that computer "programs are machines that happen to have been constructed in the medium of text" and therefore differ from other kinds of machines and other kinds of textual works for purposes of intellectual property protection); Samuelson, *supra* note 2, at 663 (arguing that computer program code is functional and therefore should not be subject to copyright protection, but rather should be subject to a new form of intellectual property protection).

I agree that software is a new kind of creative work, and that software's novel qualities should be reflected in the rules of the intellectual property system. To discern software's novel qualities and to determine the rules that should apply to intellectual property protection for software, I propose an approach which begins by analyzing the nature of software without respect to the law. In particular, I answer the following four questions in sequence: (1) "What is software?," (2) "How does software differ from other creative works?," (3) "Which of these differences, if any, are relevant to intellectual property law, and how?," and (4) "How should intellectual property law treat software in light of such differences?"

Previous attempts to determine how intellectual property law should apply to software have tended to ask whether software fits into existing intellectual property paradigms without first developing a clear, detailed, and accurate conception of what software is, how it works, and how it is created. One common approach, for example, is to recognize that patent law protects "functional" products and processes,[4] that copyright law protects "expressive" works,[5] and to conclude that patent law is the appropriate means for protecting software if software is solely (or perhaps primarily) functional,[6] and that copyright law is the appropriate means for protecting

---

[4] *See* DONALD S. CHISUM, PATENTS, A TREATISE ON THE LAW OF PATENTABILITY, VALIDITY AND INFRINGEMENT § 4.01 (Matthew Bender ed., 2000) [hereinafter PATENTS].

[5] The subject matter copyright law protects is defined in 17 U.S.C. § 102(a) (2002): "Copyright protection subsists…in original works of authorship fixed in any tangible medium of expression . . ." 17 U.S.C. § 102(b) expressly excludes from copyright protection, *inter alia*, procedures, processes, systems, and methods of operation, which are within the purview of patent law. *See also* Symposium: *Toward a Third Intellectual Property Paradigm: Misappropriation as a Third Intellectual Property Paradigm*, 94 COLUM. L. REV. 2594, 2597 (1994) ("[P]atent law protects creative but functional invention, while copyright protects creative but nonfunctional authorship.").

[6] *See, e.g.,* Vincent Chiappetta, *Patentability of Computer Software Instruction as an "Article of Manufacture": Software as Such as the Right Stuff*, 17 J. MARSHALL J. COMPUTER & INFO. L 89, 143 (1998) (arguing that "[a] proper test for patentability of software related inventions must clearly and consistently draw a line separating claims to software as the specific means for computer system implementation of the contained algorithms/processes (which are patentable subject matter) from those using a software context merely to express and communicate those algorithms/processes (which must be tested on their own merit independently of the software context to determine if they involve patentable subject matter).").

software if software is solely (or perhaps primarily) expressive.[7] If software is both functional

and expressive, so the argument goes, then software is susceptible to protection by both patent

and copyright law or some hybrid of the two.[8] Some argue that patent and copyright protection

are or should be mutually exclusive for a particular work.[9]

Although such an approach has merit, it is incomplete because it fails to provide a

sufficiently clear mechanism for determining whether intellectual property protection should be

granted to particular instances of software or for determining the scope of such protection on a

case-by-case basis. By contrast, the approach employed herein begins by first developing a clear,

detailed, and technically accurate conception of what software is,[10] before addressing any legal

considerations.[11] This approach avoids confusion about the meaning of the term "software" and

---

[7] *See, e.g.,* Pamela Samuelson, Symposium: The Semiconductor Chip Protection Act of 1984 and its Lessons: *Creating a New Kind of Intellectual Property: Applying the Lessons of the Chip Law to Computer Programs,* 70 MINN. L. REV. 471, 516 (1984) (noting that, with one exception, patent and copyright protection were mutually exclusive prior to the advent of software); Randall Davis et al., *A New View of Intellectual Property and Software*, COMMUNICATIONS OF THE ACM, Mar. 1996, at 124-45.

[8] For discussions of dual and hybrid approaches, *see generally* Symposium: *Toward A Third Intellectual Property Paradigm*: *Legal Hybrids: Beyond Property And Monopoly?*, 94 COLUM. L. REV. 2630 (1994); *see also* Symposium*: Toward A Third Intellectual Property Paradigm*: *Legal Hybrids Between The Patent And Copyright Paradigms*, 94 COLUM. L. REV. 2432 (1994); John Swinson, *Copyright or Patent or Both: An Algorithmic Approach to Computer Software Protection*, 5 HARV. J.L. & TECH. 145 (1991); John M. Griem, Jr., Note: *Against a Sui Generis System of Intellectual Property for Computer Software*, 22 HOFSTRA L. REV. 145 (1993); Gregory J. Maier, *Software Protection – Integrated Patent, Copyright and Trade Secret Law*, 28 IDEA 13 (1987); Daniel G. Feder, *Computer Software: Hybrid Technology Demanding a Unique Combination of Copyright and Patent Protection,* 59 UMKC L REV. 1037 (1991).

[9] *See, e.g.,* Allen B. Wagner, *Patenting Computer Science: Are Computer Instruction Writings Patentable?,* 17 J. MARSHALL J. COMPUTER & INFO. L. 5 (1998) ("[P]atents and copyrights are mutually exclusive statutory interests with no overlap in 'abstract expression' subject matter."); Robert A. Kreiss , *Patent Protection for Computer Programs and Mathematical Algorithms: The Constitutional Limitations on Patentable Subject Matter*, 29 N.M.L. REV. 31, 77 (1999). For an overview of the issues related to mutual exclusivity of patent and copyright protection for software, and an argument against such mutual exclusivity, *see* David A. Einhorn, *Copyright and Patent Protection for Computer Software: Are They Mutually Exclusive?*, 30 IDEA 265 (1990); Pamela Samuelson, Benson *Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions,* 39 EMORY L.J. 1025, 1128-9 (1990).

[10] *See infra* Sections II and III.

[11] *See infra* Sections IV and V.

ensures that proposed changes in the law are firmly grounded in an accurate understanding of the nature of software.

In the following sections I answer each of the four questions posed above. In brief, I conclude that an executable[12] software program[13] may be[14] a component of an electromechanical[15] machine and that executable software programs differ from other kinds of electromechanical machine components due to the unique process by which they are designed and instantiated.[16] In particular, executable software is the first kind of electromechanical machine component that may successfully be designed solely in terms of its logical structure.[17] This unique feature of software programs is relevant to patent law because patent law's rules of patentability and claim scope generally require electromechanical machine components to be conceived of, described, and claimed in terms of their physical structure, not merely in terms of their logical structure.[18] Application of these rules to software initially resulted in confusion about whether software programs should be patentable, and more recently has resulted in legal protection for software programs which is in some aspects too broad, in others too narrow, and in all senses not in harmony with the nature of software.

---

[12] The term "executable" means "[c]apable of being run on [a] computer." WEBSTER'S NEW WORLD COMPUTER DICTIONARY 341 (9th ed. 2001).

[13] The terms "software program" and "computer program" are used interchangeably herein.

[14] I assert that an executable software program "may be" a machine component because a particular executable software program may or may not be a machine component depending on the function performed by the software. For a more detailed justification of this position, *see infra* Section V.B.1.

[15] I use the term "electromechanical" to mean "electrical, mechanical, or a combination of both."

[16] See *infra* Section II.

[17] By "successfully be designed" I mean that an executable software program may be built automatically based solely on a design that specifies the software program in terms of its logical structure. In other words, the "success" of the design process depends on the ability to construct a working embodiment of the software program based on the design specification that results from the design process without the exercise of any additional creative effort by a human being.

[18] See *infra* Section II.

Patent law does, however, contain the basic mechanisms that are needed to protect the useful aspects of software programs. These mechanisms of patent law should be modified to allow software programs to be patented solely in terms of their logical structure.[19] In particular, patent law should allow software programs to be conceived of, described, and claimed solely in terms of their logical structure. The traditional requirements for patentability (e.g., novelty, nonobviousness, and utility) should be applied to the logical structure of the claimed software program in each case, and the scope of software patent claims should be commensurate with the scope of the claimed logical structure.

II. What is Software?

A. "Software" Defined as "Executable Software Programs"

The first step in the present analysis is to answer the ontological question, "What is software?" The question itself is ambiguous because the term "software" has many meanings.[20] The term "software" is used to refer alternatively, for example, to algorithms and procedures carried out by software programs,[21] descriptions of software written in a natural language,[22]

---

[19] See *infra* Section V.

[20] For dictionary definitions of "software" *see* MERRIAM WEBSTER'S COLLEGIATE DICTIONARY 1117 (10th ed. 1993) (defining software as "something used or associated with and usu. contrasted with hardware," such as "the entire set of programs, procedures, and related documentation associated with a system and esp. a computer system"); WEBSTER'S NEW WORLD COMPUTER DICTIONARY, *supra* note 12, at 341 (defining software as a "computer program or programs, in contrast to the physical equipment on which programs run (hardware))"; MICROSOFT COMPUTER DICTIONARY 489 (5th ed. 2002) (defining software as "[c]omputer programs; instructions that make hardware work").

[21] *See, e.g.,* Griem, *supra* note 8, at 154 ("Where the mechanical engineer uses pumps and pulleys to manipulate physical objects, the software designer uses programming algorithms to manipulate inchoate data."). Statements such as this imply that software programs literally are algorithms.

[22] *See, e.g.,* Mark Aaron Paley, Article: *A Model Software Petite Patent Act*, 12 COMPUTER & HIGH TECH. L.J. 301, 319 (1996) ("[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.").

source code[23] written on paper or stored in a computer's memory,[24] and executable software stored on a persistent storage medium such as a hard disk.[25] My inquiry is limited, however, strictly to executable software programs, so that the question being answered in the first step of the present analysis may be reduced to the following question: "What are executable software programs?"

As used herein, the term "executable software program" refers to a tangibly embodied sequence of instructions which are executable automatically by a physical computer.[26] An instance of the Microsoft® Word word processor residing in the memory of a personal computer in the form of stored electrical signals is an example of an executable software program. The instance of Microsoft Word is said to be "running" or "executing"[27] when a user is using the program to write a letter or perform other tasks.[28] Even if the program is simply residing in the

---

[23] For definitions of "source code," *see, e.g.,* WEBSTER'S NEW WORLD COMPUTER DICTIONARY, *supra* note 12, at 343 ("In a high-level programming language, the typed program instructions that programmers write before the program is compiled or interpreted into machine language instructions the computer can execute."); *see also* MICROSOFT COMPUTER DICTIONARY, *supra* note 20, at 491 ("Human-readable program statements written by a programmer or developer in a high-level or assembly language that are not directly readable by a computer.").

[24] *See, e.g.,* Symposium: *Toward a Third Intellectual Property Paradigm: A Manifesto Concerning the Legal Protection of Computer Programs*, *supra* note 3, at 2322 ("Thinking of software as a machine makes it clear that source code is the medium in which a program is created, even though the value of the program, as with other machines, lies in its behavior.").

[25] *See, e.g.,* Symposium: *The Future of Software Protection: Common Law, Uncommon Software*, 47 U. PITT. L. REV. 1037, 1064 (1986). The USPTO has conceded that computer programs embodied in a tangible medium, such as a floppy diskette, are patentable subject matter. Software patent claims which describe a program in terms of a set of instructions embodied in a tangible medium are common. *See* Richard H. Stern, *An Attempt to Rationalize Floppy Disk Claims*, 17 J. MARSHALL J. COMPUTER & INFO. L. 183 (1998).

[26] The term "physical computer" refers herein to a physical machine implementing the "general-purpose computer architecture" described in *supra* Section III.C. The term "physical computer" therefore contrasts with terms such as "logical computer," "abstract computer," and "virtual machine," all of which refer to an ideal or abstract machine that has the logical structure (i.e., architecture) defined by the general-purpose computer architecture.

[27] The terms "run" and "execute" are synonymous with respect to computer programs. *See, e.g.*, WEBSTER'S NEW WORLD COMPUTER DICTIONARY, *supra* note 12, at 341(defining "run" as "[t]o execute a program").

[28] The term "execute" means "[t]o perform an instruction. In programming, execution implies loading the machine code of the program into memory and then performing the instructions." MICROSOFT COMPUTER DICTIONARY, *supra* note 20, at 200.

computer's memory or in a file on a hard disk and therefore not presently *executing*, the program

is *executable* because it is in a form that may be executed automatically[29] by the computer.[30]

I equate "software" with "executable software programs" because an executable software

program is a physical instantiation of a program that actually causes a computer to perform the

program's intended function. If we are to inquire whether "software" is susceptible to protection

by patent law in the same manner as traditional machines and processes that perform useful

functions, we should be sure to compare such machines and processes to the form of software

(i.e., its executable form) that is also capable of performing useful functions. To compare a

---

[29] Providing a rigorous definition of automation is beyond the scope of this article. At least two plausible meanings, however, are sufficient for purposes of the present discussion. In one sense, an action is "automatic" if its execution does not involve the performance of substantial physical acts by a human being. This sense of the term "automatic" focuses on the lack of all but the most trivial *physical* involvement by a human being in performance of the action, and is reflected in certain dictionary definitions of "automatic" (e.g., "done or produced as if by machine" and "having a self-acting or self-regulating mechanism") and "automaton" ("a machine or control mechanism designed to follow automatically a predetermined sequence of operations or respond to encoded instructions"). MERRIAM WEBSTER'S COLLEGIATE DICTIONARY, *supra* note 20, at 78. A dishwasher, for example, washes dishes "automatically" in the sense that it performs the necessary sequence of actions without the physical assistance of a human being, other than the physically trivial acts of selecting a setting and pressing the power button. In another sense, the term "automatic" refers to an action that is performed without all but the most trivial *mental* involvement by a human being. In particular, the term "automatic" in this sense refers to an action that is performed without the substantial exercise of conscious human thought or discretion. According to this sense, an action may be performed automatically even if physically performed substantially or entirely by a human so long as the action is performed unconsciously or without the need to exercise substantial discretion. Workers on an industrial assembly line perform their actions "automatically" in this sense despite the significant amount of physical labor they perform. This sense of the term "automatically" is also reflected in conventional dictionary definitions (e.g., "largely or wholly involuntary" and "acting or done spontaneously or unconsciously"). *Id.* In the context of software, the instance of Microsoft Word described above may be executed in response to a user double-clicking on an icon representing the program. The Microsoft Word program is executable "automatically" in both senses just described: execution of the software requires the human user to engage in neither substantial *physical* activity nor substantial *mental* activity. Executing the program is akin to pressing the "power" button on a dishwasher. Interesting questions are raised by various meanings of the term "automatic," such as whether computer source code written on paper qualifies as an "executable software program" if it may be executed "automatically" by a computer in either or both of the two senses described above. The answers to such questions and the technological, social, and legal implications of this observation are beyond the scope of this paper. Also beyond the scope of this paper is the question of whether all tangibly-embodied instructions that are executable by a computer qualify as software, or whether such instructions must be stored in the memory of a computer to qualify as software.

[30] Drawing a bright line between executable source code and non-executable software designs is particularly difficult. As a result, it is particularly difficult to identify the point at which the process of "designing" software ends and the process of "implementing" or "coding" software begins. Despite these difficulties, *executability* provides a reasonable basis for distinguishing between mere software designs and executable source code. *See, e.g.,* DICK HAMLET & JOE MAYBEE, THE ENGINEERING OF SOFTWARE: TECHNICAL FOUNDATIONS FOR THE INDIVIDUAL 211-12

physical machine, for example, to a programmer's idea of a program, rather than to the

executable form of the program, would be to compare apples to oranges. I therefore use the term

"software" to refer to executable software programs, the term "design" to refer to the Platonic

universal[31] for a software program,[32] the term "conception" to refer to a programmer's mental

idea of the software program, and the term "specification" to refer to the programmer's written

description of the software program[33]. Although other terms could be ascribed the same

meanings, what is most important is that the meanings that are chosen be applied consistently.[34]

B. Executable Software Programs are Physical Components of Computers

Executable software programs literally are physical components of computers. This

assertion has been and remains controversial.[35] In particular, software programs are routinely

---

(Addison-Wesley 2001) ("Despite the similarities between the processes of designing and coding, there is one tremendous distinction between the end results: code can be executed on hardware – a design cannot.").

[31] *See* PLATO, THE REPUBLIC, Book VII (Viking Press 1955). For a discussion of the relationship between the universal/particular distinction and the idea/expression distinction in copyright law, *see* Thomas M. Powers, *Ideas, Expressions, Universals, and Particulars: Metaphysics in the Realm of Software Copyright Law*, Proceedings of the Sixth Annual Ethics and Technology Conference 195 (2003).

[32] A design is one example of a "logical entity" as that term is defined below in Section III.D.1.

[33] Although these terms are commonly used with the meanings ascribed to them herein, the terms "conception" and "specification" must be carefully used in context because each may be used to refer either to a process or to the product of that process. For example, a programmer may form *a conception* of a software program as a result of engaging in the *process of conception*. The context in which such terms are used should make their intended meaning clear in particular cases.

[34] The term "design," for example, has other meanings, which differ from the meaning used herein. According to U.S. patent law, for example, a design patent may be granted for any "new, original and ornamental design for an article of manufacture." 35 U.S.C. § 173 (2002). A design patent protects the *nonfunctional* aspects of an ornamental design as shown in a patent. *See* Elmer v. ICC Fabricating, Inc., 67 F.3d 1571 (Fed. Cir. 1995); *see also* Keystone Ret. Wall Sys., Inc. v. Westrock, Inc., 997 F.2d 1444 (Fed. Cir. 1993). I use the term "design" to refer to universal (ideal) representing the logical structure of an invention, despite the potential for confusion, because its colloquial meaning is closest to that which is intended herein, and because other potential terms (such as "invention") carry with them even more problematic theoretical baggage.

[35] *See, e.g.,* Wagner, *supra* note 9, at 34-36 (arguing that "[s]oftware is disembodied symbolism" even when fixed in a tangible medium). Although I claim that a software program literally is a component of a machine, computer scientists sometimes use term "software component" in a different, figurative, sense to draw an analogy between physical components of an electromechanical device and (typically object-oriented) software program "components" that may be interconnected with each other logically to form larger programs. *See, e.g.,* BERTRAND MEYER, OBJECT-ORIENTED SOFTWARE CONSTRUCTION (Prentice Hall 2d ed. 1997).

characterized as "intangible," "abstract ideas," and mere "mental processes," to distinguish them from physical hardware.[36]

Such characterizations of software make one or more of three mistakes. The first mistake is to conclude that computer programs *are* mental processes merely because they *mimic* or *replace* mental processes. The fact that software mimics or replaces mental processes does not mean that software itself is a "mental process" any more than the fact that a pocket calculator mimics or replaces mental processes means that the calculator *is* a mental process.[37]

The second mistake involved in the denial that executable software programs are physical is to use terms such as "intangible" and "abstract" in one sense when referring to software and in

---

[36] *See, e.g.* James Gleick, *Patently Absurd*, N.Y. TIMES, Mar. 12, 2000 (Magazine), at 44 *available at* http://www.nytimes.com/library/magazine/home/20000312mag-patents.html ("Patents began in a world of machines and chemical processes – a substantial, tangible, nuts-and-bolts world – but now they have spread across a crucial boundary, into the realm of thought and abstraction."). Although Gleick later acknowledges that software programs "*are* machines, in a way," he asserts that "they are machines without substance – incorporeal machines, machines made of imagination and 'logic' and 'bits.'" *Id.* Jim Warren of Autodesk, Inc., testified before Congress that "[s]oftware is not a gadget. . . . Software is what occurs between stimulus and response, with no physical incarnation other than as representations of binary logic." *Prepared Testimony and Statement for the Record of Jim Warren Before the Patent and Trademark Office* (Jan. 26-27, 1994), *available at* http://www.bustpatents.com/autodesk.htm (September 14, 2003). Warren also recommended that the Patent Office "[i]ssue a finding that software . . . implements intellectual processes that have no physical incarnation; processes that are exclusively analytical, intellectual, logical and algorithmic in nature." *Id.* Professor Donald Knuth, the grandfather of algorithms, stated his belief in a letter to the Patent Office that "[t]he Patent Office has fulfilled this mission [of serving society by formulating patent law] with respect to aspects of technology that involve concrete laws of physics rather than abstract laws of thought," such as software. Letter from Professor Donald Knuth, the Patent Office, *available at* http://lpf.ai.mit.edu/Patents/knuth-to-pto.txt (September 14, 2003). *See also* John Perry Barlow, *The Economy of Ideas*, WIRED (Mar. 1994) (arguing that "all the goods of the Information Age [including software] . . . will exist either as pure thought or something very much like thought: voltage conditions darting around the Net at the speed of light, in conditions that one might behold in effect, as glowing pixels or transmitted sounds, but never touch or claim to 'own' in the old sense of the word.").

[37] *See, e.g.*, GEORGES IFRAH, THE UNIVERSAL HISTORY OF COMPUTING: FROM THE ABACUS TO THE QUANTUM COMPUTER 120, 123-124 (John Wiley & Sons 2001) (claiming that Pascal's invention of the Pascaline mechanical calculating device in 1624 "mark[e]d the final break with the long age of ignorance, superstition and mysticism which above all had stopped the human race from contemplating that certain mental operations could be consigned to material structures made up of mechanical elements, designed to obtain the same results."). Similarly, Charles Babbage, the 19th century mathematician and designer of early computation devices referred to as the Inference Engine and the Analytical Engine, was recognized as having effectively embodied "mental processes" in a physical form. *See id.* at 204 (quoting W.S. Jevons, *On the Mechanical Performance of Logical Inference*, 160 PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY, 497-518 (1870) ("The mind seems thus able to imbue matter with some of its highest attributes, and to create its own rival in the wheels and levers of an insensible machine.").

another sense when referring to other kinds of creative works.[38] For example, the second mistake

is made when the term "tangible" is used to refer to *physical embodiments* of inventions such as

an automobile engine, while the term "intangible" is used to refer: (1) to the *processes* carried out

by executable software programs, (2) to the best explanation of such programs, or (3) to the

information conveyed by such programs.[39] This is to compare apples to oranges. All processes,

explanations, and information are intangible, not just those that pertain to software. Although an

automobile engine is tangible, for example, the processes that it carries out are intangible because

all processes are intangible. The intangibility of the processes performed by the engine does not

render the engine itself intangible. An executable software program, like an automobile engine,

[38] The term "intangible" is used with several meanings in the law and the literature. For example, the term sometimes refers to entities, such as electrical signals and microscopic particles, that literally are not "touchable" or directly perceptible by the unaided senses. *See, e.g.,* Richard H. Stern & Edward P. Heller, *In Re* Alappat: *The Gordian Knot Retwisted*, 2 U. BALT. INTELL. PROP. J. 187 (1994) (characterizing electrical signals as "intangibles") (quoting *In re* Schrader, 22 F.3d 290, 294 (Fed. Cir. 1994)); *see also* Barlow, *supra* note 36. Sometimes the term "intangible" refers to energy in contrast to matter, so that even microscopic particles are "tangible" despite the fact that they are not directly perceptible by the senses, while electricity is "intangible" because it is energy, even if it takes the form of a lightning bolt and therefore is perceptible directly by the senses. *See, e.g.,* Eric J. Sinrod, *Court Rules Insurance Doesn't Cover Damage to Computer Systems*, *available at* http://www.duanemorris.com/publications/pub899.html (July 16, 2002) (describing case of America Online, Inc. v. St. Paul Mercury Insurance Co. (E.D. Va. 2002), in which the court held that damage to computer data did not constitute damage to "tangible" property under an insurance policy). Finally, the term "intangible" sometimes is used to refer to Platonic universals (ideals), while the term "tangible" is used to refer to particulars, so that all matter and energy are tangible, while ideas, opinions, laws, and mathematical formulae are intangible. *See, e.g.,* Greenwalt v. Stanley Co., 54 F.2d 195, 196 (3d Cir. 1931) (examples of "intangible, illusory, and nonmaterial things" include "emotional or aesthetic reactions"); *In re* Schrader, 22 F.3d 290, 296 n.12 (Fed. Cir. 1994) (implicitly distinguishing "intangible subject matter" from "physical activity or objects"). Similar confusion arises with the term "physical," which is sometimes used to refer only to matter in contrast to energy and other times to refer both to matter and energy (particulars) in contrast to universals. I use the terms "tangible" and "physical" herein synonymously to refer to any entity that is composed of matter and/or energy, i.e., to particulars in the physical world.

[39] This is exemplified by the distinction often drawn between "bits and atoms," as popularized by Nicholas Negroponte in BEING DIGITAL. *See* NICHOLAS NEGROPONTE, BEING DIGITAL (Vintage Books 1995). This distinction, although useful in some ways, conflates the logical and the physical in a way that can be as likely to confuse as to illuminate. In particular, comparing bits, which are units of information (like decimal numbers) that can be embodied in many different physical forms, to atoms, which are physical entities, is to compare apples to oranges and to conflate the logical properties of bits with the physical properties of electrons. Negroponte claims, for example, that "[a] bit has no color, size, or weight, and it can travel at the speed of light." *Id.* at 14. In reality, these are properties of electrons, whether such electrons carry digital information (i.e., bits) or analog information (such as an analog television signal). For purposes of clarity it would be more useful to define a bit as a logical unit of information having certain logical properties (in the same way that numbers in general are defined), distinct from any particular physical form in which a bit may be embodied, and to speak of whether bits are *embodied*,

is not itself "intangible" merely because the processes it performs are intangible or because the information it conveys is intangible.

The third mistake involved in the denial that executable software programs are physical is to point out that software is "intangible" merely because it is embodied in electrical signals, which are a form of energy and therefore "intangible" in contrast to matter. This is a mistake for at least two reasons. The first reason is that it is not necessary for executable software programs to be embodied in electrical signals; they may be embodied in matter.[40] The second reason is that the distinction between energy and matter is irrelevant to the question of whether executable software programs embodied in energy are components of machines is because there is no reason that electrical signals may not be components of machines. A machine is "an assemblage of parts that transmit forces, motion and energy one to another in a predetermined manner" or "an instrument . . . designed to transmit or modify the application of power, force, or motion."[41] With the advent of programmable computers, electrical signals are now capable of being configured into particular arrangements to perform all of these functions.

Electrical signals are tangible (physical) in the sense that they are subject to the laws of physics and, like matter, can act upon and be acted upon by other energy and matter. In this

---

instantiated, or *encoded* in a particular physical entity. Using the term "bit" in this way would help to avoid confusion in this already confusing area.

[40] Punch cards, for example, in common use until the 1970's, stored computer programs on physical cards rather than in electronic memories. Such cards were a form of executable software program. *See, e.g.*, IFRAH, *supra* note 37, at 240-41. Breakthroughs in biological computing may enable tomorrow's software to be constructed of indisputably tangible biological materials such as strands of DNA. *See, e.g.,* Thomas F. Knight, Jr. & Gerald Jay Sussman, *Cellular Gate Technology*, MIT ARTIFICIAL INTELLIGENCE LABORATORY, (1998); *see also* Leonard Adleman, *Molecular Computation of Solutions to Combinatorial Problems*, 266 SCIENCE 1021, 1021-23 (1994); Dan Boneh et. al., *On the Computational Power of DNA*, 71 DISCRETE APPLIED MATHEMATICS, SPECIAL ISSUE ON COMPUTATIONAL MOLECULAR BIOLOGY, 79-94 (1996); Sandeep Junnarkar, *In Just a Few Drops, a Breakthrough in Computing*, N.Y. TIMES, May 21, 1997. Biocomputing is still in its embryonic stage.
[41] MERRIAM WEBSTER'S COLLEGIATE DICTIONARY, *supra* note 20, at 697.

critical sense electrical signals are not "intangible" in the same sense as ideas, mental

processes,[42] or mathematical formulae, which are not subject to the laws of physics.

An executable software program stored in the memory of a computer is an example of a

machine component embodied in electrical signals. An executable software program, whether

embodied in energy or matter, has a particular physical structure. Consider again, for example,

the instance of Microsoft Word residing in the memory of a computer in the form of stored

electrical signals. These signals, as a whole, have a particular physical structure. Although this

physical structure is invisible, it exists nonetheless in the form of a particular configuration of

electromagnetic fields, which are physical in the sense just described. It is this particular

physical structure, stored physically in the computer's memory, that interacts with the computer's

hardware to cause the computer to perform functions such as displaying text on the screen.[43]

Different executable software programs have different physical structures which interact with the

computer hardware in unique ways to cause the computer to perform particular functions, just as

different drill bits perform different functions when connected to a drill. Computer hardware is

designed so that the stored electrical signals being processed by the processor at a particular time

activate particular circuitry in the processor to perform the desired function. If an executable

software program stored in the memory of a computer had no physical structure and were

"intangible" in the same sense as an abstract idea, the program would not be capable of causing

the computer to do anything.

---

[42] This assumes that, in accordance with the Cartesian mind-body duality, the mind is a non-physical entity distinct from the body.

[43] *See generally* STEPHEN A. WARD & ROBERT H. HALSTEAD, JR., COMPUTATION STRUCTURES (MIT Press 1990), especially pages 281-512 for an introduction to the interaction between stored electrical software and computer hardware.

C. Software is the Variable Part of a Computer

A computer includes both hardware and software. What distinguishes hardware from software? Although one might be tempted to define hardware as the "physical" part of a computer and software as the electrical or "non-physical" (intangible) part of a computer,[44] such a distinction is untenable because, as described above, software is tangible.

Software is defined better by reference to the unique *architecture* that is shared by all modern digital computers. In general, the "architecture" of a machine refers to a combination of the functions performed by the machine's components and the organization and arrangement of such components.[45] In general, the architecture of modern digital computers includes both hardware and software components.[46] The hardware components are fixed and immutable. It is in this sense that they are "hard." The software components may be modified and in this sense are "soft."

---

[44] For examples of such definitions, *see, e.g.,* John C. Phillips, Note: *Sui Generis Intellectual Property Protection For Computer Software*, 60 GEO. WASH. L. REV. 997, 1002 (1992) ("In the broadest sense, a computer consists of two elements: hardware and software. Hardware includes the physical embodiment and structures associated with a computer system."); Andrew G. Isztwan, *Computer Associates International v. Altai, Inc.: Protecting The Structure Of Computer Software In The Second Circuit*, 59 BROOK. L. REV. 423, 425 (1993) ("Hardware consists of the physical electronic circuits in which the processing ordered by the instructions occurs.").

[45] *See, e.g.,* WEBSTER'S NEW WORLD COMPUTER DICTIONARY, *supra* note 12, at 24 (defining "architecture" as "[t]he overall conceptual design and design philosophy of a hardware device or computer system or network"). In the context of software development, "[i]t has become common practice to use the term *architecture* to characterize the internal structure of a software system . . . ." JAMES F. PETERS & WITOLD PEDRYCZ, SOFTWARE ENGINEERING: AN ENGINEERING APPROACH 206 (John Wiley & Sons, Inc. 2000). *See also* HAFEDH MILI ET AL., REUSE-BASED SOFTWARE ENGINEERING: TECHNIQUES, ORGANIZATION, AND CONTROLS 382-393 (John Wiley & Sons, Inc. 2001). As will become clear from the discussion below in Section III.D, the architecture of a computer is an example of a logical structure. The general-purpose computer architecture requires only that the components of a computer perform particular functions and exhibit particular logical properties both individually and collectively, without requiring that any particular physical structures be used to implement such features. In fact, one might say that all modern computers have the same basic logical structure despite having widely varying physical structures.

[46] Although the term "computer architecture" is sometimes used to refer solely to computer hardware, a computer is not complete in an important sense until a particular software program is stored in its memory. I therefore use the term "computer architecture" to refer to a combination of hardware and software and, in particular, to refer to the particular relationship between hardware and software that defines modern digital computers.

More specifically, the modern computer architecture includes a processor, a memory, and a set of instructions stored in the memory. The processor retrieves instructions from the memory and executes them. The processor and memory are fixed and therefore are hardware. The set of instructions is also referred to as a "program" or more generally as "software."[47] Different programs may be stored in the same memory and executed on the same processor.

The hardware of a computer does not do anything particularly useful by itself. Rather, it is intended to serve as a *platform* on which software may executed.[48] A computer without software cannot automatically balance a checkbook, transmit electronic mail, or even add a pair of numbers.

In this sense, computer software is to computer hardware as a drill bit is to a drill. A drill is a machine and drill bits are components of that machine. The drill provides generic hardware and each of the bits provides specific hardware for performing a particular function when connected to the drill to form a complete machine. To make a drill that performs a new function, it is not necessary to design and build an entire new drill. Rather, one need only design and build new bits, so long as such bits are shaped to fit into the designated physical connector on the drill.

---

[47] *See, e.g.*, WEBSTER'S NEW WORLD COMPUTER DICTIONARY, *supra* note 12, at 341 (software is "[a] computer program or programs, in contrast to the physical equipment on which programs run (hardware)."); MICROSOFT COMPUTER DICTIONARY, *supra* note 20 (software is "[c]omputer programs; instructions that make hardware work").
[48] *See, e.g.,* MICROSOFT COMPUTER DICTIONARY, *supra* note 20, at 407-8 (defining "platform" as "[t]he foundation technology of a computer system. Because computers are layered devices composed of a chip-level hardware layer, a firmware and operating-system layer, and an applications program layer, the bottommost layer of a machine is often called a platform."). The term "platform" is sometimes used alternatively to refer to a combination of particular hardware and a particular operating system, in contrast to application programs (such as word processors and web browsers), which require the particular combination of hardware and operation system to execute. *See, e.g.,* WEBSTER'S NEW WORLD COMPUTER DICTIONARY, *supra* note 12, at 286 (defining "platform" as "[a] type of computer system defined by the type of hardware and operating system used. An example of a platform is the category of computers that have an Intel or Intel-compatible processor and are equipped with [the] Microsoft Windows [operating system.]").

The basic architecture of modern computers is analogous to that of a drill and bit. A computer is a machine and software programs are components of that machine.[49] The computer's hardware is generic; it is capable of performing functions that are common to all of the software that is capable of being executed on the computer. For this reason, the computer architecture described herein is commonly referred to as the "general-purpose computer architecture."[50]

Storing a particular software program in the memory of the computer forms a complete machine that is capable of performing the particular functions defined by the program.[51] To make a computer that performs a new function, it is not necessary to design and build an entire new computer. Rather, one need only design and build new executable software for performing the new function.[52]

---

[49] *See, e.g.,* Symposium; *Toward a Third Intellectual Property Paradigm: Article: A Manifesto Concerning the Legal Protection of Computer Programs*, *supra* note 3, at 2316 ("[P]rograms are, in fact, machines (entities that bring about useful results, i.e. behavior) that have been constructed in the medium of text (source and object code)."). Programs need not, however, be "constructed in the medium of text," as evidenced by the existence of "visual" (i.e., graphical) programming languages, such as Microsoft Visual C++®, which allow programs to be specified at least in part using graphical rather than textual elements. Programmers design programs in such programming languages by using a mouse and/or keyboard to add icons to the program, modify existing icons, and interconnect icons in various ways. Any program written in a textual programming language may alternatively be represented graphically and *vice versa*. *See, e.g.,* Joseph G. Arsenault, Comment: *Software without Source Code: Can Software Produced by a Computer Aided Engineering Tool be Protected?*, 5 ALB. L.J. SCI. & TECH. 131 (1994).

[50] The "general-purpose computer architecture" is described in more detail in Section III.C, *infra*. For reasons which will become clear in Section III.C, *infra*, modern digital computers are also referred to alternatively as "universal Turing machines" and, by abbreviation, as "Turing machines" and "universal machines."

[51] *See In re* Alappat, 33 F.3d 1526, 1545 (Fed. Cir. 1994) ("[S]uch programming creates a new machine, because a general purpose computer in effect becomes a special purpose computer once it is programmed to perform particular functions pursuant to instructions from program software."); *In re* Bernhart, 417 F.2d 1395, 1400 (C.C.P.A. 1969) ("[I]f a machine is programmed in a certain new and unobvious way, it is physically different from the machine without that program; its memory elements are differently arranged. The fact that these physical changes are invisible to the eye should not tempt us to conclude that the machine has not been changed.").

[52] In fact, as described in Section III.G below, one need only design the function to be performed by the software, because the computer itself performs the equivalent of structural design and construction of the software.

III. How does Software Differ from Other Creative Works?

A. Introduction

The second step in the present analysis is to answer the question, "How does software differ from other creative works?" Based on the discussion above, this reduces to the question: "How do executable software programs differ from hardware components of computers and other electromechanical machines?" I conclude that the fundamental[53] distinction between software and conventional electromechanical devices is the unique process by which software is designed and instantiated. To justify this conclusion, I first describe some salient features of the processes that are used to design conventional electromagnetic machine components.

B. Conventional Electromechanical Design

Various models of the process of engineering design have been proposed.[54] I will use a simplified model of engineering design, referred to herein as the "ideal design model," as a tool for describing how electromechanical devices historically have been invented.[55]

---

[53] Software differs from conventional electromechanical components in other ways, at least as software is embodied using current technology. For example, as described above, software may be embodied in electrical signals rather than matter, and therefore may be copied, stored, and transported more quickly than matter-based components. This is not, however, the fundamental difference between software and hardware for the reasons described herein.

[54] *See generally* GERARD VOLAND, ENGINEERING BY DESIGN (Denise Penrose ed., Addison-Wesley Longman, Inc. 1999); NAM P. SUH, THE PRINCIPLES OF DESIGN (J.R. Crookall et al. eds., Oxford University Press, Inc 1990); ATILA ERTAS & JESSE C. JONES, THE ENGINEERING DESIGN PROCESS (Cliff Robichaud ed., John Wiley & Sons, Inc. 2d ed. 1993). In particular, increasing attention is being paid to the application of engineering design techniques to software development. *See generally* PETERS & PEDRYCZ, *supra* note 45; MILI, *supra* note 45; SHARI LAWRENCE PFLEEGER, SOFTWARE ENGINEERING: THEORY AND PRACTICE, (Prentice Hall 2d ed. 2001); HAMLET & MAYBEE, *supra* note 30.

[55] I use the term "ideal" not to imply that the model described herein is superior to others, but merely to indicate that it is "idealized" in the sense that it is an abstraction incorporating features of various other models. In particular, the "ideal design model" described herein is an example of a "waterfall" design model because the stages of the model are performed in sequence, each stage producing results that form the basis for the next stage, calling to mind a waterfall cascading down a terrace. *See, e.g.,* PFLEEGER, *supra* note 54, at 48-50. Other kinds of engineering design models include life-cycle models, such as the evolutionary, prototyping, incremental, and spiral life-cycle models. *See, e.g., generally* PETERS & PEDRYCZ, *supra* note 45, at 46-48. The analysis herein does not, however, require the process of engineering design to proceed according to the stages of the ideal design model described herein. Rather, the ideal design model that I present is provided merely to highlight particular characteristics of engineering design that are relevant to computer programming and to patent law. In practice, engineers employ an extremely wide

The ideal design model includes five stages: (1) Problem Definition; (2) Requirements Analysis; (3) Logical Structural Design; (4) Physical Structural Design; and (5) Construction.[56] In the simplest case, a single engineer engages in each of these stages in sequence. To clarify explication of the ideal design model, I will use Samuel Morse's invention of the telegraph as an example.[57]

In the Problem Definition stage, the inventor defines the problem to be solved.[58] Prior to Morse's invention of the telegraph, for example, both Morse and other engineers realized that existing modes of communication, such as horse-drawn carriage, were inefficient and unreliable, and that electricity likely could be harnessed to solve this problem. The problem these engineers set out to solve was how to design and build a device that could transmit messages quickly and reliably over long distances using electricity.[59]

The second stage, Requirements Analysis, involves analyzing and defining the requirements that any solution to the problem defined in the Problem Definition stage must satisfy.[60] In the case of the telegraph, for example, the requirements may have included the ability to transmit messages over a particular distance using no more than a particular amount of electricity.

---

variety of techniques to design new products and processes, some of which are highly rigid and methodical and others that are largely spontaneous and based on intuition. All such techniques are consistent with the analysis presented herein.

[56] Although for completeness the model may include additional stages, such as testing and maintenance, such stages are omitted because they are not relevant to the discussion herein. For a description of a somewhat more complex design model, *see* STEVE MCCONNELL, CODE COMPLETE: A PRACTICAL HANDBOOK OF SOFTWARE CONSTRUCTION 1-6 (Microsoft Press 1993).

[57] I have used facts described in the Supreme Court case of O'Reilly v. Morse, 56 U.S. 62 (1853), whenever possible. In cases where relevant facts are not available in *O'Reilly*, I use hypothetical facts and indicate as such in the text.

[58] *See, e.g., generally* PETERS & PEDRYCZ, *supra* note 45, at 119-20; SUH, *supra* note 54, at 30-35.

The third stage, Logical Structural Design, involves designing the logical structure of the solution to the problem in a manner consistent with the requirements.[61] Although I defer a definition of the term "logical structure" to a later section,[62] the stage of Logical Structural Design may be explained at this point in terms of Functional Design, which is a subset of Logical Structural Design. The process of Functional Design involves designing subsystems of a design in terms of the functions the subsystems perform.[63] In the case of the telegraph, for example, the Functional Design stage may have involved determining that solving the defined problem and satisfying the identified requirements would require a device including: (1) a transmission subsystem (transmitter) for transmitting a message over a communications channel and (2) a reception subsystem (receiver) for receiving the transmitted message over the communications channel.[64] Note that the two subsystems of the proposed solution are defined at this stage solely in terms of the functions they perform: sending and receiving. The proposed solution does not, at

---

[59] *See* STOBBS, *supra* note 2, at 129. ("During the decade that followed [Oersted's discovery of electromagnetism], several prominent inventors saw Oersted's miraculous new form of energy as a possible way to communicate at a distance. Many tried, but Samuel Morse is credited as the one who first made it happen.").

[60] *See, e.g., generally* PETERS & PEDRYCZ, *supra* note 45, at 117-122; SUH, *supra* note 54, at 12-13.

[61] I use the term "Logical Structural Design" to refer to a stage of the design process that is often referred to by other names. For example, it is sometimes referred to as "architectural design". *See, e.g.,* MCCONNELL, *supra* note 56, at 3; PETERS & PEDRYCZ, *supra* note 45, at 205-209. The term "structural design," by itself, is ambiguous, because it may refer either to the design of a logical structure or to the design of a physical structure (which I call "physical structural design"). Similarly, the term "structure" is ambiguous because it may refer either to a logical structure or a physical structure, depending on the context in which it is used. For example, the term "structure" in the title of the classic introductory text on computer programming, STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS, refers to the *logical* structure of computer programs. *See* HAROLD ABELSON & GERALD JAY SUSSMAN, STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS (MIT Press 1985). More generally, references in the software engineering literature to the "structure" of software refer to the *logical* structure of software. I make liberal use of the terms "logical" and "physical" as qualifiers for the terms "structure" and "structural" to avoid any such ambiguities.

[62] *See,* Section III.D.1, *infra.*

[63] *See, e.g.,* HAMLET & MAYBEE, *supra* note 30, at 89-90; PETERS & PEDRYCZ, *supra* note 45, at 132-137.

[64] The process of Functional Design may further be performed for each subsystem. For example, Morse may have decided that the transmitter should include a means for receiving a message from a human, means for converting the message into electrical signals, and means for transmitting the message over the communications channel. This process of subdividing an element of a design into lower-level elements is referred to as "decomposition." For descriptions of functional decomposition, *see, e.g.,* MEYER, *supra* note 35, at 103-14; MCCONNELL, *supra* note 56, at 145-48.

this stage, specify which particular physical structure should or can be used to perform these functions.

In the Physical Structural Design stage, the engineer designs the physical structure of a device that implements the logical structure designed in the Logical Structural Design stage.[65] In the case of the telegraph, Physical Structural Design involved selecting and/or designing the particular physical components of the telegraph (such as the main circuit, key with signal lever, local circuit, and receiver with electromagnet)[66] and the physical interconnections among them.[67]

Note that the result of the Physical Structural Design stage is not a tangible physical product, such as a telegraph, but rather a *design* for such a product.[68] The term "design" as used herein refers to a Platonic universal (ideal), although it may also refer to the engineer's mental idea of the design without any loss of clarity in the present analysis. A design for a physical product may be tangibly embodied in a *physical design specification*, such as a schematic drawing of a telegraph.

The next and final stage in the ideal design model is Construction.[69] In this stage, a *builder*, sometimes referred to as a *mechanic*, uses the physical design specification provided by

---

[65] *See, e.g.,* SUH, *supra* note 54, at 18-25.

[66] *See* O'Reilly v. Morse, 56 U.S. 62, 55 (1853).

[67] Morse conceived of the physical structure of the telegraph while on a trans-Atlantic voyage from Havre, France to New York. *See id.* at 12-13.

[68] Note that the term "design" is used to refer to the end product of the *process* of physical structural *design*. *See, e.g.,* PFLEEGER, *supra* note 54, at 195 ("Design is the creative process of transforming the problem into a solution; the description of a solution is also called [a] design."). It is important to keep these two meanings of the term "design" distinct. *See supra* note 34.

[69] *See, e.g.,* ERTAS & JONES, *supra* note 54, at 28-30. The Construction Stage often is referred to by other names, such as "manufacturing" or "production." Further confusing matters, the term "construction" in the context of computer programming is often used to refer to the process of "constructing" the *logical* structure of a program by writing source code, rather than to the process of constructing the *physical* structure of the program. For example, the term "construction" in the titles of the books *Code Complete: A Practical Handbook of Software Construction* and *Object-Oriented Software Construction* refer to the detailed design of computer source code, not to the kind of physical construction associated with conventional electromechanical devices. *See* MCCONNELL, *supra* note 56; *see also* MEYER, *supra* note 35. This usage of the term "construction" is correct by analogy in that it refers to the final

the engineer-inventor to build a working physical embodiment of the designed device, such as a physical and operational telegraph.[70] The resulting device solves the problem defined in the Problem Definition stage, satisfies the requirements specified in the Requirements Analysis stage, has the logical structure designed in the Logical Structural Design stage, and has the physical structure designed in the Physical Structural Design stage.

A physical device built in the Construction stage is an example of a "physical entity" as that term is used herein. A physical entity is composed of lower-level physical entities that are physically interconnected in a particular way. The "physical structure" of a physical entity refers to the combination of the physical entity's components and their physical interconnections. The process of Physical Structural Design involves identifying existing physical entities, possibly designing modifications to their physical structure, and designing physical interconnections among them. Existing physical entities include both raw natural materials and man-made physical entities.

This idealized model of engineering design just described illustrates at least two relevant points about electromechanical design. The first is that historically it has been necessary for a human being to engage in Physical Structural Design, if nothing else, to invent a new electromechanical device. Even if, for example, Morse had dispensed with Problem Definition, Requirements Analysis, and Logical Structural Design, and had conceived of the telegraph in a single flash of genius, his conception would have had to include a conception of the physical

---

and most detailed stage carried out by a human being in both electromechanical and software design. The use of the term "construction" to refer to the design of computer source code, however, may be confusing in the context of the present discussion. I therefore only use the term "construction" to refer to the physical construction of a physical entity (such as an executable software program), not to any aspect of the process of *designing* a software program or other product.

structure of the telegraph. If Morse's conception fell short of this, his design would not have

qualified as an invention.[71] A corollary to this conclusion is that problem definitions,

requirements, and logical structural designs historically have been insufficient to enable the

construction of new electromechanical machines in the absence of physical structural designs.

Imagine, for example, that Morse had merely engaged in Problem Definition, thereby

identifying that devices which could transmit messages more quickly and reliably than a horse

and carriage would be useful, if they existed. Such an accomplishment would not have been

considered to be an act of invention, because it would not have been possible for Morse to make

and use a working device that solved the problem based merely on his identification and

description of the problem. Morse's specification of requirements in the Requirements Analysis

stage would fail to qualify as an invention for the same reason.

Now imagine that Morse had completed the next stage, Functional Design.[72] Even if

Morse had determined that a system including a sending device, a communications channel, and

a receiving device could solve the defined problem and satisfy the specified requirements, such a

functional design alone would not have enabled Morse to make and use a working telegraph

absent a conception of particular physical structures for performing the functions of sending,

communicating, and receiving.

The inability of functional designs by themselves to enable the construction and use of

working machines that solve previously unsolved problems and/or satisfy previously unsatisfied

---

[70] "The true date of invention is at the point where the work of the inventor ceases and the work of the mechanic begins." Cameron & Everett v. Brick, 1871 C.D. 89, 90 (Comm'r Pat. 1871). The builder may, of course, be the same person who performed the Physical Structural Design stage.

[71] I use the term "invention" in this portion of the analysis despite potential confusion with the legal meaning of the term "invention" because the use of the term "invention" in this context closely tracks its legal meaning. The legal requirements for invention are described below in Section IV.C.

requirements historically has been a near-universal feature of electromechanical design.

Deriving a physical structural design from a functional design historically has been a "hard problem," i.e., a problem for which no generally applicable techniques have been available to apply routinely in particular cases. As a result, the only way to obtain novel physical structural electromechanical designs has been to rely on the creativity of individual engineers and whatever idiosyncratic design techniques such creativity might entail. In the absence of a physical structural design produced through the creativity of a human engineer, a particular problem might go unsolved for a long period of time, even if the problem, requirements, and functional design are well known and expressible in simple terms.

Only upon designing a particular physical entity having a particular physical structure has it been possible to make and use working embodiments of a new electromechanical device, which is why we have reserved the term "invention" to refer to the design of such physical structures. Prior to performance of this critical step, electromechanical engineers historically have remained in the realm of abstract thought, not concrete problem-solving. Although many engineers in Morse's time had identified the functional characteristics of a hypothetical device that could transmit messages quickly and reliably over long distances using electricity, Morse was the first to design a device having a physical structure capable of performing these functions and therefore to qualify for the title of "inventor."[73]

Similarly, it typically has been necessary for inventors to *describe* their inventions *in terms of* their physical structure to enable others to make and use them. Even once Morse had

---

[72] As described above, Functional Design is a specific case of Logical Structural Design.
[73] *O'Reilly*, 56 U.S. 62, 107 (1853).

designed the physical structure of the telegraph, it was not possible for anyone other than Morse to make and use a telegraph unless he described it in terms of its physical structure.

The second point the ideal design model illustrates is that electromechanical engineers typically design the physical structure of a particular device to perform a particular function. A hammer drives nails and a toaster toasts bread, but a toaster may not hammer nails nor a hammer toast bread. Even in the case of a device, such as a Swiss army knife, which is designed to perform many functions, the functions performed by such a device typically are dictated by and correspond to particular physical structures in the device. As a result, conventional electromechanical devices may be considered to be *special-purpose machines*. To perform a function that an existing machine does not already perform, it typically has been necessary to design a new special-purpose machine for performing that function.

C. Enter the Universal Machine

Designing the physical structure of an electromechanical machine can be tedious, time-consuming, costly, and prone to error. The founders of modern computing realized this, and set about to create another way to design machines. Alan Turing and other early innovators during World War II were tasked with cracking the encrypted messages used by the Axis.[74] Time was short. Turing helped to design and build code-breaking machines consisting of large amounts of complex and interconnected circuitry. Each particular code-breaking machine, like virtually all calculating devices prior to the advent of the modern computer, was designed to perform particular operations on a code in an attempt to break it.[75] If the machine was not successful, it

---

[74] The definitive biography of Alan Turing is ANDREW HODGES, ALAN TURING: THE ENIGMA (Walker & Company 2000); see also MARTIN DAVIS, ENGINES OF LOGIC: MATHEMATICIANS AND THE ORIGIN OF THE COMPUTER (W.W. Norton 2000).
[75] *See, e.g.,* IFRAH, *supra* note 37, at 158.

was necessary to redesign and build a new machine, or at least to redesign and re-wire parts of the existing machine, and then to try again.[76] This was extremely tedious and time-consuming. There had to be a better way.[77]

Turing's solution to these problems was to design a "universal machine" that would be capable of performing functions so numerous and varied that for most practical purposes they may be considered to be infinite. The architecture of all of today's modern computers is based on Turing's universal machine.[78]

A universal machine is capable of performing such a wide variety of functions because of its unique architecture, which comprises, in its essence, a memory for storing instructions (software), and a processing unit for retrieving instructions from the memory, carrying out the instructions, and storing the results of the instructions back into the memory.[79] The architecture may also include an input unit for receiving data from a human user or an external device, and an output unit for providing data to a human user or external device.

A universal machine can be made to perform different functions merely by storing appropriate instructions in the machine's memory and activating the machine. In response, the

---

[76] *See, e.g.,* JOHN AGAR, TURING AND THE UNIVERSAL MACHINE: THE MAKING OF THE MODERN COMPUTER 60-61 (Totem 2001) ("The ENIAC . . . could also be 'reprogrammed,' switching from calculating trajectories to the motion of a shockwave. But this reprogramming meant extensive rewiring: two days spent completely rearranging a nest of plugboard wires. It was not the same machine doing different calculations, but a subtly different machine for each job. . . . [T]he frustration of rebuilding the machine each time the mathematical problem was changed forced the ENIAC team to devise the crucial idea of this book: the concept of the stored program.").

[77] For a general overview of Turing's codebreaking work during World War II, *see* HODGES, *supra* note 74, at 160; DAVIS, *supra* note 74, at 170-75. The quest to eliminate the tedium of electromechanical design is related to the general quest of scientists to eliminate the tedium involved in performing the calculations that are necessary for all scientific work. *See, e.g.,* IFRAH, *supra* note 37, at 101 (quoting L.F. Menabrea, *Sur la Machine Analytique de Charles Babbage*, COMPTES RENDUS DES SÉANCES DE L'ACADÉMIE DES SCIENCES, 179-82 (July 28, 1884); AGAR, *supra* note 76, at 39-62, 101-12.

[78] The particular architecture implemented by modern digital computers is sometimes referred to as the "von Neumann" architecture, named after John von Neumann, another computing pioneer. In 1945 von Neumann wrote the now-famous *First Draft of a Report on the EDVAC*, which proposed that the proposed EDVAC computer be designed and constructed as a physical implementation of a universal Turing machine. DAVIS, *supra* note 74, at 182.

machine executes the instructions, thereby performing the function specified by the instructions. In so doing, the universal machine mimics a special-purpose machine having a physical structure specially designed to perform the specified function. The universal machine is "universal" in the sense that its single set of hardware may perform a near-infinite number of functions and thereby mimic nearly any other special-purpose machine.

This extreme degree of flexibility, or malleability,[80] is indeed the primary feature that distinguishes computers from other kinds of electromechanical devices.[81] Computers are sometimes referred to as "general-purpose computers" to indicate their generality of function. Although it would have been difficult to believe 100 years ago that a single machine could balance checkbooks, play games, perform mathematical calculations, display artwork and movies, and transmit messages across the world, a single modern computer can perform all of these different functions and more.[82]

Turing first conceived of and described the universal machine solely in abstract, logical terms.[83] The term "abstract universal machine" refers solely to the architecture of the universal

---

[79] *See, e.g.,* DAVIS, *supra* note 74, at 151; IFRAH, *supra* note 37, at 272-80, 313.

[80] James Moor coined the term "logically malleable" to refer to this feature of computers. *See* James H. Moor, *The Future of Computer Ethics: You Ain't Seen Nothin' Yet!*, 3 ETHICS & INFO. TECH. 89, 89-91 (2001).

[81] References to the "computer revolution" sometimes focus on the incredible speed at which computers operate and the degree of miniaturization that is realized in computer technology. Although these technological advances may qualify as revolutionary in their own right, they are distinct from the theoretical advance represented by the unique architecture of the general-purpose computer.

[82] Even Howard Aiken, one of the founders of modern computing, expressed his surprise at the ability of a computer to mimic a wide variety of machines when he said: "If it should turn out the basic logics of a machine designed for the numerical solution of differential equations coincide with the logics of a machine intended to make bills for a department store, I would regard this as the most amazing coincidence I have ever encountered." PAUL CERUZZI, RECKONERS: THE PREHISTORY OF THE DIGITAL COMPUTER, FROM RELAYS TO THE STORED PROGRAM CONCEPT 43 (Greenwood Press 1983) (quoted in DAVIS, *supra* note 74).

[83] *See* Alan Turing, *On Computable Numbers, with an Application to the Entscheidungsproblem, Proceedings of the London Mathematical Society*, ser. 2, vol. 42 at 230-67 (1936-37), correction in ser. 2, vol. 43 at 544-46 (1937). For a description of the substance of the paper and the work behind it, *see, e.g.,* AGAR, *supra* note 76, at 139-76. The "universal machine" is also referred to as the "universal Turing machine," which is capable of mimicking any of a near-infinite number of special-purpose machines referred to as "Turing machines."

machine, devoid of any particular physical structure for implementing that architecture. Turing

and others did more, however, than merely design an abstract universal machine as an intellectual

curiosity; they also designed and built physical computers that implemented the architecture of

the abstract universal Turing machine.[84] This required both additional theoretical

breakthroughs[85] and a variety of technological breakthroughs, such as the development of fast

and reliable electronic memories.[86] The universal Turing machine is implemented in modern

computers using physical hardware components such as silicon-based central processing units

(CPUs), random access memories (RAMs), display monitors, and keyboards.

D. Software Design is Logical Structural Design

The process of computer programming enables a programmer to create a machine that has

a particular novel physical structure for performing a particular function without requiring the

programmer to design the novel features of the machine in physical terms. This differs from

conventional electromechanical design, in which it is necessary to engage in physical structural

design to produce a working machine having a novel physical structure. In the following

sections I expand upon and justify these assertions about the process of software design.

1. Logical Structure Defined

Before describing in more detail how programmers design programs in terms of their

logical structure, the meaning of the term "logical structure" will be clarified. The term

"structure" in colloquial usage often refers to logical structure. For example, the "structure" of

an essay or poem refers to its *logical* structure, not its physical structure. The logical structure of

---

[84] DAVIS, *supra* note 74, at 169-97.

[85] John von Neumann, for example, is typically credited with proving the equivalence between an abstract universal Turing machine and actual digital computers having particular physical structures. *See* IFRAH, *supra* note 37, at 291; *id.* at 178.

this article is represented by the particular hierarchy of the article's sections, sub-sections, paragraphs, sentences, and words. Although an embodiment of this article on a physical page or a computer monitor screen may have a particular physical structure consisting of blobs of ink or illuminated phosphors arranged in a particular configuration, the "structure" of the article presumptively refers to the article's *logical* structure. The same is true when we speak of the "structure" of a law, a system of beliefs, or an argument.

More generally, the term "logical structure" refers herein to the structure of a *logical entity*. A "logical entity" is defined herein recursively as an entity which is either a primitive logical entity or which is composed of logical entities that are related to each other solely by logical relationships.[87] Although I do not provide a strict definition of "primitive logical entity," familiar examples include numbers, such as the numbers two, zero, and negative four. Such numbers are logical (abstract) entities that do not have any physical structure, although they may be embodied in or represented by physical structures, such as ink on a page. Another example of a primitive logical entity is a unit of currency, such as a dollar. A dollar *bill* is a physical entity (composed of paper and ink) which *embodies* or *instantiates* a single dollar.

A "logical relationship" is a relationship between two logical entities. For example, arithmetic operations, such as addition and subtraction, are examples of logical relationships. The logical operation of addition relates the numbers two and four in the arithmetic expression

---

[86] *See, e.g., id.* at 177-97, 280-95.

[87] The term "software architecture" is often defined in a similar way, as in the following:

A description of a software architecture consists of three basic elements: processing, data, and connecting elements…

- A *processing element* is a software structure that transforms its inputs into required outputs.
- A *data element* consists of information needed for processing or information to be processed by a processing element.
- *Connecting elements* are the "glue" that holds different pieces of an architecture together.

"2 + 4." The expression "2 + 4" is therefore an example of a logical entity, which consists of primitive logical entities (the numbers 2 and 4) which are related to each other by a logical relationship (addition). Mathematical formulae more generally (such as $z^2 = x^2 + y^2$, better known as the Pythagorean Theorem) are among the best-known examples of logical entities. Mathematicians design, understand, manipulate, and describe mathematical formulae directly in terms of their logical structure.[88] Other examples of logical relationships include the relationship among money in different sub-accounts of a bank account, the relationships among sections and sub-sections in written works such as this one, and the hierarchical organizational relationship among employees in a corporation.

In addition to the fact that mathematical formulae are themselves not physical, the terms in such formulae need not represent physical entities, and a person who performs mathematical operations need not know anything about what the operations mean.[89] Consider, for example, the fact that a school child can successfully be taught to add two twenty-digit numbers on paper using purely formal operations without knowing what the process of addition means and without being able to conceive of any physical quantity that might be represented by either of the two numbers or their sum.

PETERS & PEDRYCZ, *supra* note 45, at 208. In this description of software architecture, the processing and data elements are examples of logical entities and the connecting elements are an example of logical relationships.

[88] Although one might attempt to distinguish between mathematics and logic, modern mathematics and logic are inextricably intertwined. *See, e.g.,* BERTRAND RUSSELL, INTRODUCTION TO MATHEMATICAL PHILOSOPHY, 194 (1996) (quoted in IFRAH, *supra* note 37, at 268) ("Logic has become more mathematical and mathematics more logical. The consequence is that it is now wholly impossible to draw a line between the two; in fact, the two are one. They differ as the child differs from the man; logic is the youth of mathematics and mathematics is the manhood of logic.").

[89] "Thus it will be readily understood that in order to demonstrate a theorem, it is not necessary or even useful to know what it means… we might imagine a machine where we should put in axioms at one end and take out theorems at the other, like that legendary machine in Chicago where pigs go in alive and come out transformed into hams and sausages. It is no more necessary for the mathematician than it is for these machines to know what he is doing." H. POINCARÉ, SCIENCE AND METHOD ch. 3 (Dover Press 1952) (quoted in DAVIS, *supra* note 74, at 93).

The fact that a particular logical entity may *represent* a physical quantity or physical entity does not mean that the logical entity is *itself* physical. For example, the fact that the variable *x* or the number two in a particular formula represents the amount of (physical) water in a container does not mean that the variable *x* or the number two are *themselves* physical entities. Although this observation may appear obvious, failure to adhere to it strictly has caused much confusion in patent law, as described below in Section IV.E.

2. Putting the Pieces Together

The mere fact that a computer is capable of performing a near-infinite number of functions in theory would be of little practical value if it were prohibitively difficult in fact to determine how to make the computer perform a particular desired function. Computer programming is the art of determining how to make computers perform particular functions by designing and implementing appropriate computer program instructions. One of the goals of computer science is to make it increasingly easy to do so.

One way in which this goal is achieved is through the design and implementation of increasingly powerful *programming languages*[90] in which computer programmers may write programs. The process of computer programming involves designing computer programs[91] and expressing them in instructions defined according to an appropriate programming language.[92]

---

[90] A "programming language" is defined as "[a]n artificial language consisting of a fixed vocabulary and a set of rules (called syntax) that one can use to create instructions for a computer to follow." WEBSTER'S NEW WORLD COMPUTER DICTIONARY, *supra* note 12, at 299.

[91] *See supra* n. 54.

[92] Certain aspects of computer programming involve mental and physical processes that are the same as or very similar to those involved in writing literary works. As a result, computer programming is often referred to as a process of "writing" code. *See, e.g.,* Randall M. Whitmeyer, Comment: *A Plea for Due Processes: Defining the Proper Scope of Patent Protection for Computer Software*, 85 NW. U.L. REV 1103, 1107 (1991) ("Developing software is most definitely a literary exercise, like writing an essay or a book."). All too often, however, computer programming is conceptualized *solely* as an act of "writing," thereby obscuring aspects of programming that are dissimilar from writing. *See, e.g.,* Alan L. Durham, *"Useful Arts" in the Information Age*, 1999 B.Y.U.L. REV 1419,

For purposes of simplicity, assume that a computer to be programmed only recognizes programs written in a single programming language. Such a programming language, like a natural language, has a particular syntax which defines words in the language (typically referred to as "keywords" or "commands") and the ways in which they may be combined into phrases (referred to as "expressions"), sentences (referred to as "statements"), and larger syntactic structures. The programming language also has a particular semantics which defines the commands, statements, and other syntactic structures in terms of the functions they perform.[93]

Instructions in programming languages typically include instructions for performing arithmetic operations and Boolean logic and instructions for making decisions based on the outcome of other instructions. Such instructions define logical entities in at least two senses. The first is that each instruction is defined independently of the hardware on which the instruction is to execute. Consider, for example, the instruction "Add(2,4)," which is an instruction to add the numbers two and four. This instruction does not specify the physical features of the memory in which the instruction is to be stored or the physical features of the processor which is to execute the instruction. The second sense in which programming language instructions define logical entities is that they define instructions in terms that are independent of the physical structure in which the instructions are to be embodied when stored in memory or

---

1457-8 (1999) ("[P]rogramming does not begin with coding any more than house building begins with nailing boards together. In either case, the construction phase is preceded by a planning stage."). Increasing attention, however, is being paid to the process of *designing* programs prior to and while writing source code. *See generally* MCCONNELL, *supra* note 56, at 53-170. Furthermore, the choice of language may influence the design process in important ways. *See* ABELSON & SUSSMAN, *supra* note 61, at 4 ("A powerful programming language is more than just a means for instructing a computer to perform tasks. The language also serves as a framework within which we organize our ideas about processes.").

[93] More generally, commands, statements, and other syntactic structures are defined in terms of their logical structure, as described in more detail below.

executed by a computer. For example, the instruction "Add(2,4)" does not specify a particular configuration of electrical signals for embodying the instruction in the memory of a computer.

The fact that computer program instructions do not refer to the physical structure of either hardware or software follows from the fact that such instructions are defined solely in terms of logical entities. The instruction "Add(2,4)," for example is defined solely in terms of the logical operation of addition performed upon the numbers two and four, which are logical entities. The instruction is a logical entity because it is defined solely in terms of a logical operation performed on logical entities.

To program a computer to perform a particular function, such as adding two numbers or drawing a circle on a display screen, a programmer must either identify an existing command in the programming language for performing the function or determine how to combine existing commands in the language to perform the function. Consider, for example, a programmer who desires to write a simple program for adding the numbers two and four. All modern programming languages include commands that are capable of performing such a function. The programmer might identify this command and use it to write a one-line program such as "Add(2,4)".

Writing a program that performs a more complex function, such as calculating the square root of the number four, might require combining together several instructions from the programming language, thereby defining an *algorithm* that calculates a square root. If the programmer cannot identify an existing algorithm for calculating a square root, the programmer must exercise his or her creativity to figure out how to combine instructions in the selected

programming language to define an algorithm that calculates a square root. Such a creative process lies at the heart of computer programming.

The process of computer programming is analogous to the conventional process of electromechanical engineering design in the sense that both involve selecting, designing modifications to, and designing interconnections among existing components to form new, higher-level components to perform desired functions.[94] The difference between the two is that in electromechanical design the existing components are physical entities, while in computer programming the existing components are logical entities. In other words, the creative process that occurs during the electromechanical design stage of Physical Structural Design is analogous to the creative process that occurs during the software design stage of Logical Structural Design.

It is as if a computer programming language represents a warehouse of available logical components that may be selected, modified, and combined with each other to produce new logical components.[95] The programmer indicates, using source code written in the selected programming language, which low-level logical components to select and how to modify and combine them into higher-level logical components having logical structures defined by the source code instructions. For this reason, I refer to the process of computer programming as a process of designing the logical structure of a computer program.

---

[94] *See* Symposium*, Toward a Third Intellectual Property Paradigm: Article: A Manifesto Concerning the Legal Protection of Computer Programs*, *supra* note 3, at 2327-8; *see also* Durham, *supra* note 92, at 1463-5.
[95] *See* Paley, *supra* note 22, at 325-6 ("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 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.").

3.   Not All Logical Entities are Created Equal

Having described generally the concept of a "logical entity," I will now describe three

classes of logical entity that commonly arise in computer science: algorithms (procedures),[96] data

structures, and objects. An "algorithm" typically is defined as a sequence or combination of

steps for performing a particular function.[97] The logical structure of an algorithm consists of the

steps of the algorithm and their interrelationships, such as the particular sequence in which they

are arranged.

A particular algorithm typically is specified in terms of steps that may be carried out by a

particular actual or hypothetical machine.[98] The Java programming language, for example, is

designed for programming a "virtual machine" that is defined not in terms of its physical

structure, but rather solely in terms of its logical properties.[99] The virtual machine is defined in

part in terms of the instructions it is capable of executing, namely those instructions capable of

being expressed in valid statements written in the Java programming language. Given such a

---

[96] It is common to equate the terms "computer program" and "algorithm." *See, e.g.,* Christopher M. Mislow, *Computer Microcode: Testing the Limits of Software Copyrightability*, 65 B.U.L. Rev. 733, 775 (1985); Wagner, *supra* note 9, at 14; Swinson, *supra* note 8, at 147. It is not true, however, that all computer programs are algorithms. Object-oriented software programs, for example, define systems of interrelated logical objects. Software which defines graphical user interfaces (GUIs) defines such interfaces not only in terms of the steps that such interfaces perform but also in terms of their logical content. Although neither object-oriented software programs nor GUI software programs are properly classified as "algorithms," they are still examples of software and are properly classified as defining logical entities.

[97] *See, e.g.,* MICROSOFT COMPUTER DICTIONARY *supra* note 20, at 23 (defining an algorithm as "[a] finite sequence of steps for solving a logical or mathematical problem or performing a task");DONALD E. KNUTH, THE ART OF COMPUTER PROGRAMMING: FUNDAMENTALS OF ALGORITHMS 1-4 (1973); Allen Newell, *Response: The Models are Broken, the Models are Broken*, 47 U. PITT. L. REV. 1023, 1024 (1986) ("A standard definition is: An algorithm is an unambiguous specification of a conditional sequence of steps or operations for solving a class of problems.").

[98] *See* Mitchell P. Novick & Helene Wallenstein, *The Algorithm and Computer Software Patentability: A Scientific View of a Legal Problem*, 7 J. COMPUTERS, TECHNOLOGY, AND LAW 313 (1980) ("A more scientific definition [of algorithm] would be as follows: 'Given both the problem [to be solved] and the device [to be used in solving the problem], an *algorithm* is the precise characterization of a method of solving the problem, presented in a language comprehensible to the device.'"); Swinson, *supra* note 8, at 147 ("[A]ll algorithms by definition have at least one defined or implied processor [on which to execute].").

[99] *See, e.g.,* TIM LINDHOLM AND FRANK YELLIN, THE JAVA™ VIRTUAL MACHINE SPECIFICATION 1-4 (Addison-Wesley 1996).

virtual machine, a sequence of steps only qualifies as an algorithm if it is defined in terms of valid Java programming language instructions.

A *data structure* is another class of logical entity found in computer science. The term "data structure" refers broadly to any data defined by or accessible to a program logically organized in a particular way.[100] The simplest example of a data structure is a single program variable for storing a single number or other value. An array is a somewhat more complicated data structure that is capable of storing multiple values at the same time. Programming languages typically define simple *data types*, such as numbers, text strings, and Boolean logic values (e.g., true and false) that may be stored in data structures. Programmers may combine simple data types to form more complex ones. The term *database* refers to a data structure whose logical structure may be very complex and which may store very large numbers of values of varying types. Although a data structure in isolation is not typically considered to be a "program," the design of useful data structures is as much an art as other kinds of programming.

A third class of logical entity found in computer science is a software "object," as that term is used within the object-oriented programming paradigm. The "object-oriented" programming paradigm gained force in the 1980s and is used widely today. Certain programming languages, aptly referred to as "object-oriented programming languages," are particularly well-suited for programming according to the object-oriented programming paradigm. An "object" in an object-oriented program is a logical entity which may include both procedures (also referred to as methods or routines) and data (also referred to as properties or

---

[100] Alternatively, a data structure is "[a]n organizational scheme, such as a record or array, that can be applied to data to facilitate interpreting the data or performing operations on it." MICROSOFT COMPUTER DICTIONARY, *supra* note 20, at 145.

attributes).[101] Object-oriented programs are best conceptualized and explained as collections of

interrelated objects which perform actions and communicate with each other, rather than merely

as sequences of procedural instructions (algorithms) or collections of inert data (data

structures).[102]

4. Ignorance is Bliss

Programming is objectively different than conventional electromechanical design in the

sense that programmers create programs merely by designing logical entities and embodying

those logical entities in descriptions written in source code intended to be executed by a

computer. This objective feature of programming is intertwined with a distinct subjective feature

of programming: programmers *conceive of* and *understand* programs solely in terms of their

logical structure. Furthermore, when a programmer writes source code or describes a program in

any other language, the programmer *describes* the program solely in terms of its logical structure.

Providing source code to a computer thereby creates an executable software program in

the memory of the computer.[103] Even though the computer hardware and the executable software

program have particular physical structures, the programmer need not, and in essentially all cases

does not, know what these physical structures are. In this sense, computer programmers are quite

---

[101] *See generally* MEYER, *supra* note 35, at 217-278.

[102] Object-oriented programming is a "programming paradigm in which a program is viewed as a collection of discrete objects that are self-contained collections of data structures and routines that interact with other objects." *Id.* at 373. It is further defined as "an approach to programming language design in which program elements are conceptualized as objects that can pass messages to each other by following established rules." WEBSTER'S NEW WORLD COMPUTER DICTIONARY, *supra* note 12, at 260.

[103] Although the source code may first need to be compiled into object code to be executed, whether the source code needs to be compiled is irrelevant to the present discussion. Even in cases in which the source code needs to be compiled into object code to be executed, such compilation may be performed automatically (as that term is used herein) by a computer. *See, supra* note 29. Therefore, one may provide source code to a computer and thereby automatically create an executable software program in the memory of the computer, whether or not the source code needs to be compiled to create the executable software program.

different from electromechanical engineers, who know and understand better than anyone else the physical structure of the machines they invent.

The programmer's ignorance of the physical structure of his inventions and the tools of his trade is no accident. Rather, one of computer science's express goals is to ensure that programmers can do their work in complete ignorance of the physical structure of both hardware and software. Such ignorance is desirable because it greatly simplifies and thereby facilitates the process of programming. The ability of computers to translate instructions defined solely in logical terms into physical entities for performing those instructions frees programmers from having to know or think about physical constraints while programming. Instead, programmers are free to focus solely on logical constraints, such as the logical functions performed by individual instructions and the ways in which different instructions interact logically.

In addition to keeping programmers ignorant of the physical structure of the hardware they use and the software they create, the process of programming typically keeps programmers ignorant of the internal *logical* structure of the instructions they use to write source code. For example, a programmer who uses a "square_root" instruction in a program need not, and in many cases *does* not, know what algorithm (combination of lower-level instructions) is used to implement the square root function. The programmer may then use the "square_root" instruction as one of many instructions in a yet higher-level algorithm (such as an algorithm for using the Pythagorean theorem to calculate the length of the hypotenuse of a right triangle) and provide the higher-level algorithm with a name (such as "hypotenuse_calculate"). Other programmers may then use the higher-level algorithm in their programs merely by referring to it by name, while

36

remaining completely ignorant of the internal logical structure (i.e., combination of instructions) of the higher-level algorithm.

This ability to define and re-use lower-level programs (also referred to as "procedures" or "functions") without re-writing their source code from scratch is more than just a way to eliminate unnecessarily redundant typing. It is a powerful tool for *building abstractions*. An introductory computer science text notes:

> Every powerful language has three mechanisms for [combining simple entities to form more complex ones]:
>
> *primitive expressions*, which represent the simplest entities with which the language is concerned,
>
> *means of combination*, by which compound expressions are built from simpler ones, and
>
> *means of abstraction*, by which compound objects can be named and manipulated as units.[104]

These mechanisms are powerful because they enable a programmer to focus on the task at hand at a particular level of abstraction, without needing to know how entities at lower levels of abstraction are implemented. This both facilitates division of labor among programmers and enables individual programmers to focus on problems that have not yet been solved rather than on reinventing the wheel, thereby increasing the speed and ease with which large and complex software programs may be designed. This is one example of "information hiding," a technique which is widespread throughout computer science.[105] The advantages of information hiding are so great that most programming languages do not merely allow it to occur, they encourage and even require it to occur.[106] Furthermore, it is considered extremely poor programming practice

---

[104] *See* ABELSON & SUSSMAN, *supra* note 61, at 4.

[105] For an overview of information hiding, *see, e.g.,* MCCONNELL, *supra* note 56, 118-30.

[106] A program written in the Java programming language, for example, is in theory capable of executing on any computer platform, regardless of its physical hardware or operating system. As a result, a programmer may write a single Java program without regard for the physical properties – or even the lower-level logical properties – of the

for a programmer to make any assumptions about or rely on the internal logical structure of any instructions that he uses in a program.[107]

The benefits of information hiding were recognized as long ago as 1642, when the French mathematician and philosopher Blaise Pascal constructed a mechanical calculating device called the "Pascaline."[108] Gilbert Pascal, brother of Blaise, said of the invention:

> My brother has invented this arithmetical machine by which you
> can not only do calculations without the aid of counters of any
> kind, but even without knowing anything about the rules of
> arithmetic, and with infallible reliability. This instrument was
> considered a marvel, for having reduced a science – whose source
> is the human mind – to the level of a machine, by discovering how
> to accomplish every operation with absolute reliability and without
> any need for reasoning.[109]

Much software development being performed today involves design using logical constructs that are removed from the "native" machine language of the computer's hardware by many layers of abstraction built on top of each other. Computer scientists have even developed programming techniques that enable programmers to remain ignorant of the *logical* structure of the software they produce, a topic that is beyond the scope of this paper but that will need to be addressed by intellectual property law as such techniques become more widespread.[110]

Although the process of programming hides the computer's physical structure from the programmer, this should not be mistaken for the absence or irrelevance of the computer's

---

platform(s) on which the program will execute. The Java programming language was designed not to include any instructions which allow the programmer to refer to specific physical features of the underlying computer hardware or operating system, although in practice the language does not strictly adhere to this high standard.

[107] *Id.*

[108] *See* IFRAH, *supra* note 37, at 122-24.

[109] *Id.* at 123 (quoting Gilbert Pascal, brother of Blaise Pascal).

[110] *See, e.g.,* Bob Frankston, *Beyond Limits*, in BEYOND CALCULATION: THE NEXT FIFTY YEARS OF COMPUTING 47 (Peter J. Denning & Robert M. Metcalfe eds., Copernicus 1997); *see also* Richard Green, *Train Once, Write Anywhere*, 6 JAVA DEVELOPER'S J. 8 (2001). The development of "genetic programming" techniques enables programmers to write programs that "evolve" other programs, thereby enabling programmers to remain ignorant

physical structure; rather, the physical structure of the computer is critical if software is to

execute and thereby perform its intended functions. As R.W. Hamming, a recipient of the

Association for Computing Machinery's (ACM) prestigious Turing Award, said:

> At the heart of computer science lies a technological device, the
> computing machine. Without the machine almost all of what we
> do would become idle speculation, hardly different from that of the
> notorious Scholastics of the Middle Ages. The founders of the
> ACM clearly recognized that most of what we did, or were going
> to do, rested on this technological device, and they deliberately
> included the word "machinery" in the title [of the ACM]. There
> are those who would like to eliminate the word, in a sense to
> symbolically free the field from reality, but so far these efforts have
> failed. I do not regret the initial choice. I still believe that it is
> important for us to recognize that the computer, the information
> processing machine, is the foundation of our field.[111]

5. Twice the Ignorance is Twice the Bliss

In addition to being ignorant of the physical hardware on which their programs execute,

the physical structure of the programs they write, and the internal logical structure of such

programs, programmers typically are also kept ignorant of the physical structure of the physical

objects that their software *controls* or *represents*.

Consider, for example, software for: (1) controlling a physical device, such as a printer,

monitor, or robot, connected to a computer on which the software executes, or (2) simulating

physical phenomena such as weather or traffic jams.[112] In both cases, the software's *purpose* or

function is in some way related to physical activities or objects, such as the printer controlled by

---

even of the *logical* structure of the programs they bring into existence. *See, e.g.,* STEVEN JOHNSON, EMERGENCE:
THE CONNECTED LIVES OF ANTS, BRAINS, CITIES, AND SOFTWARE (Scribner 2001).

[111] R.W. Hamming, *One Man's View of Computer Science,* 16 J. OF THE ASS'N FOR COMPUTING MACH. 3, 5
(1969).

[112] Some software, however, does not comfortably fit into either of these two categories. The proper status of such
software within patent law is a difficult and interesting question. For a discussion of this topic, *see* Edward Brown, ,
Patenting Architectural Features of Software, 2002 PROC. OF THE THIRD INT'L CONF. ON LAW AND TECH., at 67-
72.

the software or the tornado simulated by the program. Furthermore, both the source code of such programs and English-language descriptions of such programs may *refer* to physical objects. For example, the source code for a program that controls a robot arm include variables having physical names, such as "robot" and "arm," and may include instructions, such as "move_arm," that refer to physical objects.

Even in such cases, however, source code often refers to physical objects using highly abstract terms. For example, modern operating systems and programming languages allow programmers to write instructions to read information from and write information to memory as if it were an essentially infinite store of uniform memory locations, despite the size and/or physical properties of the actual physical memory installed in the computer.[113] As a result, source code instructions for accessing memory may use the term "memory" but often do not refer to particular physical properties of the memory being accessed.

This simple example is representative of a more general technique for divorcing the thought processes of programmers from the physical structure of the objects *to which their programs refer*. Such techniques enable programmers to design programs without having to take physical constraints into account, even in many cases in which the program is intended to control or interact with a physical device.[114] Electromechanical engineers routinely take physical

---

[113] This is accomplished using "virtual memory." For a discussion of techniques for implementing virtual memory *see* WARD & HALSTEAD, *supra* note 43, at 486-97.

[114] This idea is sometimes stated by asserting that mechanical engineers solve "physical problems," while computer scientists solve "intellectual" or "abstract" problems. *See, e.g.,* Allen B. Wagner, *supra* note 9, at 16 ("A natural scientist applies physical science to physical phenomena; that is, s/he conceives solutions to concrete (objective) problems. The computer scientist assigns meanings to symbols (the abstract model) and develops the steps (algorithm) of a symbol manipulating process; that is, s/he conceives solutions to abstract problems."). Assertions, often made by critics of software patents, that software is "intangible" or otherwise is not subject to the laws of physics may be interpreted in their best light as assertions that computer programmers do not take physical constraints into account when designing software, rather than as assertions that software itself is not physical. For example, although Jim Warren in his testimony to Congress claimed that software has no physical form, he also

constraints, such as size, weight, and friction, into account when designing electromechanical

devices. In contrast, the increasing trend is for software developers to take only logical

constraints into account when designing programs. The act of programming a computer is

divorced from the laws of nature, not in the sense that the resulting executable software and the

computer on which it executes do not operate according to the laws of nature, but rather in the

sense that programmers need not take the laws of nature into account when designing

software.[115]

    Certain programs having source code that refers to physical activity or objects may, when

executing on a computer, manipulate electrical signals that represent the physical activity or

---

stated more accurately that "[w]hat programmers do is figure out how to solve intellectual problems, rather than physical problems." Testimony of Jim Warren, *supra* note 36. Similarly, this is presumably what Richard Stallman, the most vocal proponent of the free software movement, means when he says: "We [software developers] can build a castle and rest it on a thin line, and it will stay up. In other fields you have to deal with matter, and make physical objects that work. But if I want to put an 'if' statement inside a 'while' statement I don't have to worry whether the 'if' statement will oscillate against the 'while' statement and eventually fracture. I don't have to worry about whether the 'if' statement will dissipate heat well enough, or whether a voltage drop across the 'if' statement will stop the 'while' statement from working or, if it is working under water, whether salt water will get between the two statements and corrode them. I don't have to worry about how I will physically assemble each copy, and whether I can physically get access during construction to put one statement inside the other, and I don't have to worry about how, if one breaks, I will replace it." Matt Loney, *Stallman: Patents victimize developers*, ZDNET (UK), *available at* http://zdnet.com.com/2100-1104-870390.html (Mar. 28, 2002); *see also* Richard Stallman & Simson Garfinkel, *Viewpoint: Against Software Patents,* 35 COMM. OF THE ACM 17, 19 (1992) ("When an if-statement follows a while-statement [in source code], there is no need to study whether the if-statement will draw power from the while-statement and thereby distort its output, or whether it could overstress the while-statement and make it fail."). Stallman's assertion that he does not "have to worry" about the physical properties of computer program instructions or "need to study" the physical interactions of computer program instructions may be interpreted as assertions that programmers need not take physical constraints into account when designing software, rather than as assertions that the software itself is not physical. Id. In other words, Stallman is best interpreted as making assertions about the thought processes of programmers rather than the physical properties of software. What Stallman omits, however, is the equally important fact that programmers need to take logical constraints, imposed by the virtual machine that they are programming, into account. *See also* references cited *supra* in note 36.

[115] *See* HAMLET & MAYBEE, *supra* note 30, at 50-51; J. DAVID BOLTER, TURING'S MAN: WESTERN CULTURE IN THE COMPUTER AGE 40 (1984) (quoted in Alan L. Durham, *supra* note 92, at 1463):

> [W]hat especially characterizes the programmer is his withdrawal from nature into the private intellectual world of the program he is writing. Normally, he thinks neither of the keyboard at which he is typing nor of the electrons that are performing the calculations. He concentrates his full attention on the abstract problem, its representations in the programming language, and the logical design of the machine he is using. In this respect, he resembles the mathematician, the philosopher, the theologian, or indeed the chess master, all of whom live more or less completely in intellectual worlds of their own making.

objects to which the source code refers. For example, the source code for a weather simulation program may refer to wind, rain, and heat, and the program, when executing, may manipulate electrical signals that represent wind, rain, and heat.

It is critical, however, to distinguish between whether a particular executable software program itself *is* physical and whether such a program refers to, controls, or represents physical activity or objects.[116] All executable software programs are physical in the sense that they are embodied in a physical form, such as a particular configuration of electrical signals residing in the memory of a computer. Not all executable software programs, however, refer to, control, or represent physical activity or objects. A program that merely performs arithmetic calculations, for example, may neither refer to nor control physical objects.

Although this distinction may appear elementary, it is critical to the legal analysis that follows,[117] because the courts have conflated these two senses of physicality repeatedly, resulting in much confusion. In particular, the courts have conflated two ways in which a process may be "physical," by alternatively using the term "physical process" to refer to: (1) processes (such as the process of baking a cake or smelting iron ore) that are performed using physical objects and/or that operate upon physical objects; and (2) processes (such as a weather-simulation program) that manipulate electrical signals that *represent* physical objects. Some cases have held claimed processes to constitute statutory subject-matter based on findings that the processes are physical in the first sense noted above.[118] Other cases have held or implied that a process may

---

[116] *See* JOHN SEARLE, MINDS, BRAINS, AND SCIENCE 37-38 (Harvard University Press 1984) (arguing that whether a computer program *simulates* physical activity (such as a rain storm) is distinct from whether the computer performs that activity ("We can do computer simulation of rain storms in the home counties . . . [yet] no one supposes that a computer simulation of a storm will leave us all wet.")

[117] *See infra* Sections IV-V.

[118] *See, e.g., In re* Walter, 618 F.2d 758, 767-68 (C.C.P.A. 1980) ("[I]f the end product of a claimed invention is a pure number . . . the invention is nonstatutory . . . . If, however, the claimed invention produces a physical thing . . . .

42

constitute statutory subject matter if it is physical in either or both senses, sometimes conflating

the two senses with each other and failing to state clearly whether the statutory subject matter

requirement requires that a process be physical in both senses or merely in either of the two.[119]

In some cases courts have pointed to the fact that elements in a process claim *represent* physical

entities or quantities seemingly as evidence that the process is itself physical in the first sense.[120]

In one case it was held that it is not necessary for a process to be physical in *either* sense to

qualify as statutory subject matter.[121]

---

the fact that it is represented in numerical form does not render the claim nonstatutory."); Diamond v. Diehr, 450 U.S. 175, 185 (1981) (process held to be statutory the claims "involve[d] the transformation of . . . raw, uncured synthetic rubber, into a different state or thing"); *In re* Pardo, 684 F.2d 912, 916 (C.C.P.A. 1982) (holding that a process claim directed to controlling the internal operations of a programmed computer constituted statutory subject matter because the claim was directed to "executing programs in a computer," which the court viewed as indistinct from a strictly mechanical adding machine); *In re* Grams, 888 F.2d 835, 840 (Fed. Cir 1989) (holding that an algorithm that failed to perform "physical steps" did not qualify as statutory subject matter); State Street Bank & Trust Co. v. Signature Financial Group, Inc., 149 F.3d 1368, 1373 (Fed. Cir. 1998) (holding that "the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price," qualified as statutory subject matter because it produced "a useful, concrete and tangible result," even though the transformed data did not represent physical entities). Although the court in *State Street* did not expressly refer to the *physical* transformations performed by the machine, the referenced "transformation of data" must be a physical transformation because it is performed by a machine, and the only transformations that machines are capable of performing are physical transformations. Id. The court's holding may therefore be interpreted, in its best light, to stand for the proposition that a machine which performs a *physical* transformation (e.g., of electrical signals from one form into another) qualifies as statutory subject matter if the result of the transformation is useful, concrete, and tangible (i.e., physical).

[119] *See, e.g., In re* Taner, 681 F.2d 787, 790 (C.C.P.A. 1982) (holding that claimed processes which both *performed* and *simulated* physical activity satisfied the statutory subject matter requirement, without distinguishing between these two sense of physicality or explaining the relevance of either to the subject-matter patentability determination); *In re* Abele, 684 F.2d 902 (C.C.P.A. 1982) (basing statutory subject matter determinations on whether the data used by the claimed processes *represented* physical entities); *In re* Schrader, 22 F.3d 290, 294 (Fed. Cir. 1994) (holding disputed process claims not to be directed to statutory subject matter because they "do not reflect any transformation or conversion of subject matter representative of or constituting *physical activity or objects*").

[120] *See* Arrhythmia Research Technology v. Corazonix Corp., 958 F.2d 1053 (Fed. Cir. 1992) repeatedly conflates the two senses of physicality noted herein.

[121] AT&T Corp. v. Excel Communications, Inc., 172 F.3d 1352 (Fed. Cir. 1999). The reasoning in the *AT&T* decision is seriously flawed. For example, the court in *AT&T* cited the *State Street* decision in support of the proposition that a process need not effect a "physical transformation" to qualify as statutory subject-matter, even though the holding in the *State Street* case applies only to apparatus – not process – claims, and even though the holding in the *State Street* case does not imply that a statutory process need not be *embodied* in a physical form, only that the data manipulated by a statutory process need not *represent* physical activity or objects. Id. at 1359. The *AT&T* court, in other words, conflated the two senses of physicality described herein. Although a full analysis of the flaws in the *AT&T* court's analysis is beyond the scope of this article, the recommendations provide in Section V.B, *infra*, are intended in part to address the problems in the *AT&T* court's reasoning by providing rules for patentability that analyze the two senses of physicality separately.

To avoid these confusions, I use the term "physical process" in the discussion that follows to refer to processes that actually manipulate physical objects. In this sense, the processes carried out by all computer programs when executing on computers are "physical" processes, because all such processes manipulate electrical signals. Although the question whether a particular program refers to, controls, or represents physical activity or objects may be relevant to considerations such as patent law's utility requirement, it is not relevant to the question whether such a program itself is physical, any more than the physicality of a book depends on whether the book describes physical objects or abstract ideas.

E. Bridging the Gap

Computers are machines that are capable of transforming source code that describes a program in purely logical terms into a physical executable computer program that is capable of physically performing the functions described in the source code. In this way, computers bridge the gap between logical structure and physical structure. This seemingly magical transformation is at the heart of the computer revolution.

The process of modifying the physical structure of a computer by providing a program to the computer[122] differs significantly from the traditional process of modifying the physical structure of an electromechanical machine. A mechanic who modifies the physical structure of a conventional electromechanical machine, such as a telegraph, automobile engine, or clock, physically interacts with and understands the machine in terms of the physical structure she modifies. By contrast, a programmer who modifies the physical structure of a computer by

---

[122] I use the phrase "providing a computer program to a computer" rather than "computer programming" here to distinguish between the mere physical act of providing a program to a computer so that the program is stored in the computer's memory and the creative act of designing a computer program, which may be done in whole or in part without physically providing the program to a computer.

providing source code to the computer[123] need not even know that the computer's memory is being physically modified at all, much less understand or appreciate the nature of those physical modifications.[124] The programmer's ignorance of the physical structural changes that are made to the computer results from the computer's ability to automatically and invisibly transform program instructions expressed in logical form into a physical executable program that is capable of performing those instructions.

F. Drills and Drill Bits Revisited

Although an executable software program (such as Microsoft Word) is analogous to a drill bit in certain way,[125] software and drill bits also differ in ways that are relevant to patent law. First, a computer program and a drill bit are designed using very different processes. Mechanical engineers design drill bits in terms of their physical structure, while programmers design programs solely in terms of their logical structure. Second, once a program and a drill bit have been designed and constructed, their inventors continue to think about and explain them in very different terms. The drill bit inventor continues to conceive of the drill bit in terms of its physical structure and to instruct others how to make and use the drill bit by describing it in terms of its physical structure, while the programmer continues to conceive of and describe the program solely in terms of its logical structure.

---

[123] *See, e.g.,* John A. Kidwell, *Software and Semiconductors: Why are we Confused?,* 70 MINN. L. REV. 533, 542 (1985):

> If one conceives of a computer as an extraordinarily complicated set of electrical switches and relays, the computer program is nothing more than the list of instructions for the setting of those switches to facilitate some particular electrical manipulation. The entry of the program into the computer is nothing more than the translation of the description of the switch settings into the setting of the switches themselves. Thus, as noted earlier, the instructions have been transformed into the thing itself. That is, the instructions as to switch settings have at a certain point become the switch settings.

[124] Furthermore, the same program may produce different physical structures in different computers, or even in the same computer under different conditions. *See, e.g.,* Paley, *supra* note 22, at 326-27.

This is no accident. Rather, for all practical purposes the programmer and others who think about and describe the program have no practical choice but to conceive of and describe it in terms of its logical structure. Although it is possible that the program could be conceptualized and explained in terms of its physical structure (e.g., its particular configuration of electrical signals), the best explanation of the program for essentially all purposes (such as understanding how to modify and debug the program) is an explanation in terms of the program's logical structure.[126]

In addition, the program's source code, which was once merely a blueprint for the logical structure of a program to be, is now a description of the logical structure of an existing physical executable software program.[127] In contrast, a schematic for the drill bit now describes the existing drill bit in terms of its *physical* structure.

G. Conclusion: Software and Electromechanical Design Processes Compared

The ability to create new electromechanical machine components merely by designing and describing such components solely in terms of their logical structure is a significant departure from the conventional electromechanical design process, in which historically it has

---

[125] Although the program is embodied in electrical signals and the drill bit is embodied in matter, this is an irrelevant distinction for the reasons described in Section II.C supra.

[126] It is far from clear that it would even be possible for the human mind to appreciate the physical structure of all but the simplest programs or to explain them in terms of their physical structures.

[127] Computer program instructions are often described as instructions directed to a computer, which instruct the computer to perform particular functions. This view of computer programs, while accurate, is incomplete because it best explains only some features of computer programs. Computer programs, for example, include not only instructions that define actions to be performed, but also instructions defining data. A computer program, furthermore, need not be viewed as a set of instructions at all. Rather, a computer program may be viewed as a description of a machine, akin to a conventional architectural blueprint or electrical schematic diagram. Unlike such conventional descriptions, however, a computer program describes its corresponding machine in terms of its logical structure, rather than in terms of its physical structure.

been necessary to design novel physical structures to create new machines.[128] For

electromechanical engineers, physical structural design historically has been a difficult problem

whose solution has typically required human ingenuity in particular cases. To manufacture a new

chair, a new steam engine, or a new device for transmitting messages, it has been necessary for a

human being to engage in the creative and idiosyncratic act of physical structural design. In fact,

it is the act of designing physical structure, engaged in by Morse in the case of the telegraph, that

we typically refer to when we speak of the act of "inventing."

Computers have eliminated the need for inventors to engage in Physical Structural

Design,[129] at least for that class of inventions capable of being described in a computer

programming language, by automating the process of transforming a logical structural design into

a physical structural design and then into a working physical embodiment.[130] To the extent that

the term "invention" refers to the process of designing the physical structure of a new and useful

machine, computers have automated the process of invention.

Computers have not, of course, automated the entire process of creating new and useful

computer programs.[131] By eliminating the need for human engineers to engage in Physical

---

[128] This is not to say that electromechanical engineers do not also engage in logical structural design in the process of designing electromechanical devices, only that electromechanical engineers further need to engage in physical structural design to enable the construction of working embodiments of their designs.

[129] The automation of Physical Structural Design effectively provides software developers with a shortcut for designing and building electromechanical device components. *See* James R. Goodman et al., *supra* note 3, at 354 ("[P]rogramming a computer is just another way of making circuitry, and the exact same circuitry or equivalent circuitry can be hard wired.").

[130] *See, e.g.,* James R. Goodman, et al., The Alappat Standard for Determining That Programmed Computers are Patentable Subject Matter, 76 JPTOS 771, 780 ("A computer program . . . operates in a computer to set switches, to make electrical connections, and to form circuitry."); "[P]rogramming a computer inherently makes the circuitry [to carry out the program] – this is how a computer operates." Goodman, *supra* note 3. at 357; *Cf.* Wagner, *supra* note 9, at 17 ("[C]omputer science ingenuity does not cause change to physical phenomena, since no causal relationship crosses the Cartesian divide.").

[131] Perhaps in the future the process of logical structural design will be automated, so that we will be able to construct new machines for solving particular problems merely by specifying their requirements or the problem to be

Structural Design, however, computers have pushed the final phase of engineering design[132] that requires creativity back one stage in the ideal design model, to the stage of Logical Structural Design.

IV.   How are Software's Unique Qualities Relevant to Intellectual Property Law?

A. Patent Law Requires "Physicality"

The third step in the present analysis is to answer the following question, now reformulated in light of the discussion above: "Which of the differences between executable software programs and conventional electromechanical devices, if any, are relevant to intellectual property law, and how?" The fact that executable software programs that perform useful functions are electromechanical machine components suggests that software programs should be susceptible to protection by patent law, because patent law is the branch of intellectual property law that protects machines and machine components. The fact that programmers design executable software programs solely in terms of their logical structure, however, raises problems for patent law.[133] Patent law traditionally has limited patent protection to physical products and

_____

solved and providing the specification to a computer. Perhaps someone will write a program that merely says "cure cancer" and let the computer do the rest of the work.

[132] Software developers are engineers in the sense that they design machine components to perform useful functions in the physical world. This is not to be confused with "software engineering," a term which refers to the application of particular engineering techniques to software development. *See, e.g.,* HAMLET & MAYBEE, *supra* note 30, at 49-50. Software developers are "engineers" in the sense in which I use that term regardless of the techniques that they use to develop software. Not everyone agrees that software development is a process of "engineering" in any sense of the term. *See, e.g.*, Wei-Lung Wang, *Beware the Engineering Metaphor*, 45 COMMUNICATIONS OF THE ACM, 27-29 (2002). The view of software development as a form of engineering, however, is consistent with the view that "technology may be characterized as knowledge that is applied towards material enterprise, guided by an orientation to the external environment and the necessity of design." Symposium *The Post-Industrial Patent System*, 10 Fordham Intell. Prop. Media & Ent. L.J. 346 (1999). To the extent that engineering is characterized by the design and implementation of technology, and to the extent that software is a kind of technology, software development is a kind of engineering.

[133] The logical structure of computer programs may also be relevant to other areas of the law, such as copyright law. For an overview and analysis of cases addressing the applicability of copyright law to the logical structure of computer programs, *see, e.g.,* Dennis S. Karjala, *Copyright Law: Copyright Protection Of Computer Program*

to physical processes[134] that have practical utility.[135] Furthermore, patent law traditionally has required that the inventor of an electromechanical product conceive of, describe, and claim the product in terms of its physical structure, and that the inventor of an electromechanical process claim it in terms of the physical entities used by and/or operated upon by the process.[136] As a result, the scope of a product patent claim generally is limited to the physical structures that the patent enables the public to make and use, and the scope of a process patent claim generally is limited to the particular steps of the process and any recited physical structure upon which such steps operate.[137]

B. Origins of the Physicality Requirement

Patent law grants inventors exclusive rights in their inventions for a limited period of time.[138] In exchange for such exclusive rights, inventors are required to teach the public how to make and use their inventions.[139] The *quid pro quo* of public disclosure in exchange for a grant of exclusive rights for a limited period of time is the fundamental mechanism by which patent law attempts to promote the progress of science and the useful arts.[140] Patent law's public

---

Structure, 64 BROOK. L. REV. 519 (1998); Isztwan, *supra* note 44, at 423; Steven R. Englund, Note *Idea, Process, or Protected Expression?: Determining the Scope of Copyright Protection of the Structure of Computer Programs*, 88 MICH. L. REV. 866 (1990); Thomas M. Gage, Note Whelan Associates v. Jaslow Dental Laboratories*: Copyright Protection For Computer Software Structure -- What's The Purpose?,* 1987 WIS. L. REV. 859 (1987). This article focuses on patent rather than copyright law because patent law is particularly well-suited, and copyright law particularly poorly suited, to protecting the logical structure of useful artifacts.

[134] *See* definition *supra* Section III.D.5.

[135] *See* infra Section IV.C. *See also* Richard H. Stern, *Solving the Algorithm Conundrum: After 1994 in the Federal Circuit Patent Law Needs a Radical Algorithmectomy*, 22 AIPLA Q. J. 167, 170 (1994). Although the opinions of the Federal Circuit in the recent *State Street Bank* and *Excel* opinions reflect a movement away from an absolute physicality requirement for process claims, I argue in Section V.B.1 *infra* that such movement is both misguided and unnecessary for the protection of software by patent law.

[136] *See infra* Section IV.C.2.

[137] *See infra* Section IV.C.3.

[138] U.S. CONST. art. I, § 8, cl. 8. The term of a patent is twenty years from its effective filing date. 35 U.S.C. § 154 (a)(2) (2002).

[139] This "public disclosure" requirement is embodied in 35 U.S.C. § 112 (2002).

[140] *See*, *e.g.*, Bonito Boats Inc. v. Thunder Craft Boats Inc., 489 U.S. 141, 150-51, 9 USPQ2d 1847, 1852 (1989); *see also* PATENTS, *supra* note 4, at § 7.01.

disclosure requirement serves as least two purposes: (1) to enable the public to make and use the invention;[141] and (2) to put the public on notice of the exclusive rights granted to the patentee for the duration of the patent term.[142]

Whether patent law's *quid pro quo* strikes an appropriate balance between the exclusive rights granted to inventors and the interest in allowing the public to freely use the fruits of creative effort depends not only on the nature of the rights granted but also on the *scope* of the rights granted by patent law.[143] In general, progress will not be promoted optimally if the scope of rights granted is either too broad or too narrow. If the scope of exclusive rights granted is too broad, the inventor may obtain additional market power to an extent that is not necessary to incent additional innovation, and at the same time deny the public the right to use[144] the invention. If, on the other hand, the scope of exclusive rights is too narrow, inventors will not have sufficient incentive to innovate and/or to disclose their innovations to the public, thereby depriving the public of potential innovations. To promote the progress of the useful arts, patent law must therefore strike a delicate balance between rights that are too broad and rights that are too narrow.

---

[141] *See* 35 U.S.C. § 112 (2002) (requiring that a patent specification enable the public to make and use the claimed invention).

[142] *See* Permutit Co. v. Graver Corp., 284 U.S. 52, 60 (1931); General Electric Co. v. Wabash Appliance Corp., 304 U.S. 364, 369 (1938). Putting the public on notice of the exclusive rights granted by a patent enables the public to avoid infringing the patent, to design around the patent, and to begin using the invention after the patent expires. *See* 35 U.S.C. § 112 (requiring that patent claims particularly point out and distinctly claim the invention); Hilton Davis Chem. Co. v. Warner-Jenkinson Co., 62 F.3d 1512, 1520 (Fed. Cir. 1995) ("The ability of the public to successfully design around-to use the patent disclosure to design a product or process that does not infringe**,** but like the claimed invention, is an improvement over the prior art-is one of the important public benefits that justify awarding the patent owner exclusive rights to his invention."); State Industries, Inc. v. A.O. Smith Corp., 751 F.2d 1226, 1236 (Fed. Cir. 1985).

[143] *See generally* Julie E. Cohen & Mark A. Lemley, *Patent Scope and Innovation in the Software Industry*, 89 CALIF. L. REV. 1 (2001) (discussing the scope of software patent claims); *see also* Robert P. Merges & Richard R. Nelson, *On the Complex Economics of Patent Scope*, 90 COLUM. L. REV. 839 (1990) (discussing the impact of scope on the economic significance of patents).

The "scope" of a patent right has several dimensions. The most apparent is the term (duration) of the patent, which is fixed by statute.[145] Another dimension of scope is the extent to which the rights granted protect works other than those specifically created or envisioned by the inventor. The scope of a copyright, for example, is fairly narrow, covering essentially the particular work created by the copyright holder.[146] The scope of patent claims, however, may vary widely from claim to claim, and ascertaining the scope of a particular patent claim can be a difficult task.[147]

C. Physicality and Patentability

Patent law has developed a strong emphasis on the *physicality* of inventions[148] in an attempt to implement the policies underlying patent law's fundamental *quid pro quo*. In particular, patent law generally requires that: (1) an invention be capable of being embodied either in a physical product or a physical process;[149] (2) an inventor conceive of, describe, and claim his invention in terms of its physical structure[150] and/or the physical entities upon which it acts;[151] and (3) the scope of rights in an invention be limited by the physical structure of the invention and/or the physical entities upon which it acts.[152] I address each of these requirements in turn.

---

[144] "Use" of the invention includes both "non-creative" uses of the invention – such as building, selling, and using embodiments of the invention – as well as "creative" uses of the invention, such as designing improvements to the invention and making, selling, and using such improvements.

[145] *See* 35 U.S.C. § 154(a)(2) (2002).

[146] *See* 2 MELVILLE B. NIMMER & DAVID NIMMER, NIMMER ON COPYRIGHT §§2.01-2.20 (1995).

[147] *See generally* PATENTS, *supra* note 4, at ch. 18. The manner in which the scope of a patent claim is ascertained will be discussed in more detail in Section IV.C, *infra*.

[148] The discussion in this section refers most accurately to electromechanical inventions.

[149] *See infra* Section IV.C.1.

[150] By "physical structure" I refer not only to shape, size, and other features that are typically considered "structural," but also to color, mass, chemical composition, and other physical *features*.

[151] *See infra* Section IV.C.2.

[152] *See infra* Section IV.C.2(c).

1. Inventions Must Be Physical

The patent statute limits the availability of patent protection to two categories of inventions: (1) physical products,[153] and (2) processes that produce concrete and tangible results.[154] The choice of these categories of "statutory subject matter" is closely intertwined with patent law's *utility* requirement, which requires that a design be useful to merit patent protection.[155] More specifically, patent law's utility requirement requires *practical* utility.[156] A

---

[153] 35 U.S.C. § 101 defines four categories of statutory subject matter: machines, manufactures (also referred to as "articles of manufacture"), compositions of matter, and processes. The term "product" herein refers to machines, manufactures, and compositions of matter.

[154] Courts historically have had significant conceptual difficulty defining the characteristics that a process must have to qualify as statutory subject matter. In general, a process "is not a structural entity but rather an operation or series of steps leading to a useful result." PATENTS, *supra* note 4, § 1.03. Early decisions adopted the definition of "process" provided in the case of Cochrane v. Deener, 94 U.S. 780, 788 (1876) where the Court defined a process as "a mode of treatment of certain materials to produce a given result. It is an act, or a series of acts, performed upon the subject-matter to be transformed and reduced to a different state or thing." *Cochrane*, 94 U.S. at 787-88. Similarly, Professor Robinson defined a process as "an act or a series of acts performed by some physical agent upon some physical object, and producing in such object some change either of character or of condition." 1 WILLIAM C. ROBINSON, THE LAW OF PATENTS FOR USEFUL INVENTIONS § 159 (1890). Transformation of matter from one state to another was long the lynchpin of process subject-matter patentability. Under this view a process itself is not a physical entity but rather acts upon a physical entity to transform it into "different state or thing." Later courts defined patentable processes as those processes which fall within the "technological arts." *See, e.g., In re* Musgrave, 431 F.2d 882, 883 (C.C.P.A. 1970). The U.S. Supreme Court then announced that "[t]ransformation and reduction of an article 'to a different state or thing' is the clue to the patentability of a process claim that does not include particular machines." Gottschalk v. Benson, 409 U.S. 63, 70 (1972). Presently, a process constitutes statutory subject matter if it produces "a useful, concrete and tangible result." State St. Bank & Trust Co. v. Signature Fin. Group, Inc., 149 F.3d 1368, at 1373 (Fed. Cir. 1998). The best interpretation of this standard is that a process must at least act upon physical entities – whether they consist of matter, energy, or some combination of both – to constitute statutory subject matter. For an overview of the history of the physicality requirement as it relates to process inventions, see Christopher L. Ogden, *Patentability of Algorithms After* State Street Bank*: The Death of the Physicality Requirement*, 83 JPTOS J. PAT. & TRADEMARK OFF. SOC'Y 491, 497-506 (2001).

Courts in at least some software patent cases have declined to define the term "process" solely in terms of actions performed upon physical entities. For example, in *In re Schrader*, the Federal Circuit stated in dictum that the "subject matter transformation or reduction requirement [for processes] is not limited to physical activity or objects but rather encompasses "'changes to *intangible* subject matter representative of or constituting physical activity or objects….'" *In re* Schrader, 22. F.3d 290, 295 n.12 (Fed. Cir. 1994). Furthermore, some have argued that the Federal Circuit, through the *State Street* and *Excel* opinions, has eliminated the physicality requirement for process inventions. *See, e.g.,* Symposium: *The Post-Industrial Patent System, supra* note 132, at 4. To the extent that *State Street* and *Excel* eliminate or weaken the physicality requirement, this author agrees that such a step is a move in the wrong direction. *See supra* notes 118, 121 and *infra* Section V.B.1.

[155] *See generally* PATENTS, *supra* note 4, at ch. 4.

[156] *See, e.g.,* State St. Bank & Trust Co. v. Signature Financial Group, Inc., 149 F.3d 1368,at 1375 (Fed. Cir. 1998) ("The question of whether a claim encompasses statutory subject matter should not focus on which of the four categories of subject matter a claim is directed to…but rather on the essential characteristics of the subject matter, in particular, its practical utility"); *see also In re* Ziegler, 992 F.2d 1197, 1201 (Fed. Cir. 1993) (citing Cross v. Iizuka, 753 F.2d 1040 1044); Nelson v. Bowler, 626 F.2d 853, 856 (C.C.P.A. 1980) ("'Practical utility' is a shorthand way

design has practical utility only if embodiments[157] of the design perform a useful function.[158]

Limiting statutory subject matter to include only physical products and processes that perform concrete and tangible results makes sense in light of the utility requirement's practical focus; an invention cannot have practical utility if it neither has a physical structure nor produces physical results.

Designs for mechanical devices, consisting of interconnected physical parts that interoperate to perform a useful function by acting on physical entities, exemplify the kind of designs that patent law was originally designed to protect. Devices such as the cotton gin[159] and steam engine,[160] in which useful functions were embodied in novel physical structures, are quintessential examples of devices having practical utility. Processes may also have the practical utility patent law requires if they act upon physical entities to produce concrete and tangible results.[161] Industrial-era examples of such processes include "[t]he arts of tanning, dyeing,

---

of attributing "'real-world'" value to claimed subject matter. In other words, one skilled in the art can use a claimed discovery in a manner which provides some immediate benefit to the public."); Malta v. Schulmerich Carillons, Inc., 952 F.2d 1320, 1341 n.5 (Fed. Cir. 1991) ("The patent system is directed to practical utility, not to basic research.").

[157] Patent law uses the term "embodiment" to refer to a physical product or process that has the features defined by a patent claim. A particular physical chair, for example, may be an embodiment of a claimed chair design. Thomas Edison's light bulb invention, for example, has been implemented in millions of particular embodiments having a wide variety of shapes, sizes, and other physical properties, although all of them share the basic physical structure of the invention. In general, the claims of a patent describe the invention and the specification of the patent describes embodiments of the claimed invention.

[158] See, e.g., Examination Guidelines for Computer-Related Inventions, U.S. Patent & Trademark Office, 61 Fed. Reg. 7478 (U.S. Pat. & Trademark Off. Feb. 28, 1996) (requiring that the subject matter sought to be patent "must have a practical application," thereby "limit[ing] patent protection to inventions that possess a certain level of 'real world' value, as opposed to subject matter that represents nothing more than an idea or concept"). The utility requirement and the statutory subject matter requirement are closely intertwined and often analyzed together. For example, a process constitutes statutory subject matter only if it produces a "useful, concrete, and tangible result." *State St. Bank & Trust Co.*, 149 F.3d at 1373.

[159] U.S. Patent No. X72X (issued Mar. 14, 1794).

[160] One example of which may be found in U.S. Patent No. 403,335 (issued May 14, 1889).

[161] Because a process is, by definition, a sequence of steps, a process claim is not necessarily limited in scope to any particular machinery for carrying out the process. *See, e.g., Cochrane*, 94 U.S. at 780, 787-88 (1877). As a practical matter, however, process claims historically have been required to recite physical structure either for carrying out the claimed process or for being acted upon by the claimed process in order for such claims to satisfy the statutory subject matter and utility requirements. As described in more detail above in Section III.G, however,

making water-proof cloth, vulcanizing India rubber, [and] smelting ores…."[162] Mere

mathematical formulae, scientific theories and principles,[163] laws of nature and natural

phenomena,[164] abstract ideas, problem definitions, and functional designs are not patentable

because they lack practical utility, among other reasons.[165] It is only upon applying a

mathematical formulae, scientific principles, law of nature, or abstract idea to a practical end that

patent protection becomes available.[166]

2. Inventors Must Conceive of, Describe, and Claim Their Inventions in Terms of Their Physical Structure

(a) Conception

Patent law's conception requirement[167] requires that there be a "formation, in the mind of

the inventor, *of a definite and permanent idea of the complete and operative invention, as it is*

thereafter to be applied in practice…."[168] Conception has been said to be the "touchstone of

---

the advent of software has made it possible to adequately describe processes that have practical utility without using physical terms.

[162] Corning v. Burden, 56 U.S. (15 How.) 252, 267-68 (1854).

[163] *See, e.g.,* Le Roy v. Tatham, 55. U.S. (14 How.) 156, 174-5 (1853).

[164] *See, e.g.,* Diamond v. Chakrabarty, 447 U.S. 303, 309 (1980) ("The laws of nature, physical phenomena, and abstract ideas have been held not patentable."); Funk Bros. Seed Co. v. Kalo Inoculant Co., 333 U.S. 127, 130 (1948).

[165] *See, e.g.,* Gottschalk v. Benson, 409 U.S. 63, 67 (1972); Tol-O-Matic v. Proma Produkt-Und Mktg. Gesellschaft m.b.H**.**, 945 F.2d 1546, 1552 (Fed. Cir. 1991); *see also* Diamond v. Diehr, 450 U.S. 175, 185 (1981).

[166] *See, e.g., Le Roy*, 55. U.S. at 175 ("[T]he processes used to extract, modify, and concentrate natural agencies, constitute the invention. The elements of the power exist; the invention is not in discovering them, but in applying them to useful objects."); *Funk Bros. Seed Co.*, 333 U.S. at 130; Mackay Radio & Tel. Co. v. Radio Corp. of Am., 306 U.S. 86, 94 (1939).

[167] Conception is typically considered to be a "standard" that is applied in certain circumstances rather than a "requirement" for patentability. The Patent Office, for example, does not require any proof of conception during the prosecution of a patent application, but rather relies on the inventor's oath or declaration attesting to the fact that the inventor invented the claimed invention. *See* 35 U.S.C. § 115, 37 C.F.R. 1.63, MANUAL OF PAT. EXAMINING PROC. § 602. As a procedural matter, conception is only at issue when a third party disputes the inventorship of a patent or patent application, such as in an interference or in an invalidity defense in an infringement suit. I treat conception as a requirement, however, because there is no invention absent conception. Conception is therefore effectively a substantive legal requirement for patentability despite the fact that there currently is no procedural mechanism in place to enforce it during *ex parte* patent prosecution.

[168] Mergenthaler v. Scudder, 11 App. D.C. 264, 276 (1897). Conception is a mental act. Burroughs Wellcome Co. v. Barr Lab., 40 F.3d 1223, 1232 (Fed. Cir. 1994) ("For conception, we look not to whether one skilled in the art

inventorship, the completion of the mental part of invention."[169] In particular, to satisfy the

conception requirement an inventor must form a "mental picture" of the invention's physical

structure (in the case of a product invention)[170] or of the physical entities used by or acted upon

by the invention (in the case of a process invention).[171]

The requirement that conception include the formation of a "mental picture" of the

physical structure of an invention stems from the premise that it is only upon forming such a

mental picture that the inventor becomes able to make and use the invention and to enable others

to do so.[172] The mere identification of a problem to be solved, a result to be achieved, or a

function to be performed does not satisfy the conception requirement absent the conception of

particular physical means for solving the problem, achieving the result, or performing the

function.[173] Furthermore, the mere mental formation of an idea of a mathematical formula or

---

could have though of the invention, but whether the alleged inventors actually had in their minds the required definite and permanent idea.").

[169] Burroughs Wellcome Co. v. Barr Lab., 40 F.3d at 1227-28.

[170] *See id.* at 1228; *see also* Fiers v. Revel, 984 F.2d 1164, 1169 (Fed. Cir. 1993) ("Conception of a substance claimed per se without reference to a process requires conception of its structure, name, formula, or definitive chemical or physical properties."); Amgen, Inc. v. Chugai Pharmaceutical Co., 927 F.2d 1200, 1206 (Fed. Cir. 1991).

[171] *See* Alpert v. Slatin, 49 C.C.P.A. 1343, 1347 (C.C.P.A. 1962) ("Conception of an inventive process involves proof of mental possession of the steps of an operative process and, if necessary, of means to carry it out to such a degree that nothing remains but routine skill for effectuation thereof."); *accord* Rey-Bellet v. Engelhardt, 493 F.2d 1380 (C.C.P.A. 1974); *see also* Amax Fly Ash Corp. v. United States, 206 Ct. Cl. 756, 770 (1975).

[172] *See Burroughs Wellcome Co.*, 40 F.3d at 1228; *see also* Technitrol, Inc. v. United States, 194 Ct. Cl. 596, 609 (1971). In his treatise on patent law, WALKER ON PATENTS, Walker notes that "[a] mere idea is not conception" and that "[u]ntil the entire conception is complete and is ready to be incorporated in a practical embodiment, there is no available and complete conception of the invention within the meaning of the patent law." 1 A.W. DELLER, DELLER'S WALKER ON PATENTS § 75 356-57 (Anthony William Deller ed., 2d ed. 1964).

[173] *See* Markman v. Westview Instruments, 517 U.S. 370, 373 (1996) ("A claim covers and secures a process, a machine, a manufacture, a composition of matter, or a design, but never the function or result of either, nor the scientific explanation of their operation.") (quoting 6 E. LIPSCOMB , LIPSCOMB'S WALKER ON PATENTS 21:17 at 315-16 (E. Lipscomb, ed., 3d ed. 1985); *Fiers*, 984 F.2d at 1169); Hitzeman v. Rutter, 243 F.3d 1345, 1356 (Fed. Cir. 2001) ("An idea is definite and permanent when the inventor has a specific, settled idea, a particular solution to the problem at hand, not just a general goal or research plan he hopes to pursue.") (quoting *Burroughs Wellcome Co.*, 40 F.3d at 1228); *Amgen, Inc.*, 927 F.2d at 1206; *Fiers*, 984 F.2d at 1169; Rex Chainbelt Inc. v. Borg-Warner Corp., 477 F.2d 481, 491-92 (7th Cir. 1973); *Amax Fly Ash Corp.*, 206 Ct. Cl. at 768; Field v. Knowles, 183 F.2d 593, 611 (C.C.P.A. 1950) ("It is not sufficient that the *result* to be obtained be conceived, but it is required that there be conceived and disclosed the *means* provided to accomplish that result...."); Land v. Dreyer, 33 C.C.P.A. 1108,

scientific theory, absent the mental formulation of a particular practical application of the formula or theory, does not satisfy the conception requirement. Patent protection only becomes available when the inventor engages in physical structural design, thereby conceiving of and specifying the particular physical means for applying the mathematical formula or scientific theory, solving the identified problem, achieving the identified result, or performing the designed function.[174] Even when the inventor has invented a particular machine or process for achieving a particular result or performing the particular function, it is only the machine or process that may be patented, not the result achieved or the function performed.[175]

Upon completion of the formation of a mental picture of the physical structure of an electromechanical invention, "[a]ll that remains to be accomplished, in order to perfect the act or instrument, belongs to the department of construction, not invention,"[176] because the formation of such a mental picture represents "the point where the work of the inventor ceases and the work of the mechanic begins."[177] As a result, the "mental picture" view of the conception requirement serves well as a test for invention in the context of electromechanical inventions.

(b) Written Description

Patent law's "written description" requirement requires that the inventor provide a "written description of the invention" in the patent specification.[178] The "essential goal" of the

---

1113 (C.C.P.A. 1946) ("It is not sufficient [to satisfy the conception requirement], therefore, to show that a party claiming an invention has conceived a result to be obtained; the patentable thing is the means provided and disclosed by him to accomplish that results."); Knapp v. Morss, 150 U.S. 221, 227 (1893); Morton v. N.Y. Eye Infirmary, 17 F. Cas. 879, 881 (No. 9865) (S.D.N.Y. 1862) (No. 9865).

[174] *See* Mackay Radio & Tel. Co. v. Radio Corp. of Am., 306 U.S. 86, 94 (1939) ("While a scientific truth, or the mathematical expression of it, is not a patentable invention, a novel and useful structure created with the aid of knowledge of scientific truth may be."); *Morton*, 17 F. Cas. at 884.

[175] *Markman*., 517 U.S. at 373.

[176] Mergenthaler v. Scudder, 11 App. D.C. 264, 276 (1897).

[177] Cameron & Everett v. Brick, 1871 Dec. Comm'r Pat. 89, 90 (Apr. 1, 1871).

[178] 35 U.S.C. § 112 ¶ 1 (2000).

written description requirement "is to clearly convey the information that an applicant has

invented the subject matter which is claimed."[179] The Federal Circuit has held that the written

description and enablement requirements are distinct requirements[180] even though they derive

from a single sentence in the patent statute.[181]

In particular, the Federal Circuit has interpreted the written description requirement to

require that, as a general rule, inventors describe product inventions in terms of their physical

features.[182] In most cases it is insufficient merely to describe a product invention in terms of how

to make and use it[183] or the functions it performs,[184] devoid of any reference to the physical

features of the invention itself. The specification for a conventional electromechanical design,

for example, typically includes both a textual description and drawings that describe and

illustrate the invention's physical features.

---

[179] *In re* Barker, 559 F.2d 588, 592 n.4 (C.C.P.A. 1977).

[180] *See* Vas-Cath, Inc. v. Mahurkar, 935 F.2d 1555, 1563 (Fed. Cir. 1991). The Federal Circuit is increasingly emphasizing the importance of the written description requirement. *See*, *e.g.*, Enzo Biochem, Inc. v. Gen-Probe Inc., 296 F.3d 1316, 1324 (Fed. Cir. 2002); Regents of Univ. of Cal. v. Eli Lilly & Co., 119 F.3d 1559 (Fed. Cir. 1997); Fiers v. Revel, 984 F.2d 1164, 1170-71 (Fed. Cir. 1993); University of Cal. v. Eli Lilly & Co., 119 F.3d 1559 (Fed. Cir. 1997). The Federal Circuit's trend towards enforcing the written description requirement distinctly and vigorously has drawn sharp criticism. *See*, *e.g.*, Harris A. Pitlick, *The Mutation on the Description Requirement Gene*, 80 J. PAT. & TRADEMARK OFF. SOC'Y 209 (1998).

[181] 35 U.S.C. § 112 ¶ 1.

[182] *See Fiers*, 984 F.2d at 1170-71 (holding that a written description of a DNA sequence requires "precise definition, such as by structure, formula, chemical name, or physical properties," because conception of a DNA sequence requires the DNA sequence to be conceived in physical terms, and "one cannot describe what one has not conceived"); *accord Eli Lilly & Co.*, 119 F.3d at 1568 (affirming invalidity of claims which defined genetic material merely in terms of its function rather than its physical structure because such a definition "is only an indication of what the gene does, rather than what it is"). Although essentially all written description cases have involved product claims, it appears that the written description requirement is satisfied for a process claim by describing the particular steps that comprise the process. *See In re* Kaslow, 707 F.2d 1366 (Fed. Cir. 1983). An important exception to the general rule that product inventions be described in terms of their physical structure is described in Section I *infra*.

[183] *See Vas-Cath, Inc.*, 935 F.2d at 1563-64 ("The purpose of the 'written description' requirement is broader than to merely explain how to 'make and use'; the applicant must also convey with reasonable clarity to those skilled in the art that, as of the filing date sought, he or she was in possession *of the invention*.").

[184] *See Fiers*, 984 F.2d at 1169. Furthermore, it is not sufficient merely to describe an invention in terms of its objectives or goals. *See In re* Wilder, 736 F.2d 1516, 1521 (Fed. Cir. 1984). A purely functional description may, however, be sufficient if the function described is known to correlate to a specific structure. *See* Guidelines for the Examination of Patent Applications Under the 35 U.S.C. § 112, ¶ 1 'Written Description' Requirement, § II.A.1.3.a, 66 Fed. Reg. 1099 (U.S. Pat. & Trademark Off. Jan. 5, 2001).

Failure to describe the physical structure of an electromechanical invention would almost certainly result in failure to enable the public to make and use the invention. Imagine, for example, that Morse's patent on the telegraph had described the telegraph solely in functional terms, stating merely that the telegraph was "a device for sending and receiving messages over long distances reliably using electricity." Such a description, lacking any mention of the particular physical structure of the telegraph, would have been insufficient to enable the public to make and use a telegraph. The written description requirement therefore is a good, although not perfect, proxy for the enablement requirement in the context of conventional electromechanical inventions, and therefore serves at least some of the same valid purposes as the enablement requirement.[185]

(c) Claims

Every patent must include "claims" that particularly point out and distinctly claim that which the inventor regards as his invention.[186] A patent's claims define the invention that is protected by the patent.[187] One might view the patent specification as a description of embodiments of the invention and the claims as a description of the invention itself. Different claims in a single patent may provide different scopes of protection to an invention.

A product claim may claim a product invention in essentially[188] one of two ways: (1) in terms of its physical structure, or (2) in terms of the functions it performs using the "means plus

---

[185] *See, e.g.,* Reiffin v. Microsoft Corp., 214 F.3d 1342, 1345 (Fed. Cir. 2000) ("The purpose of [the written description requirement] is to ensure that the scope of the right to exclude, as set forth in the claims, does not overreach the scope of the inventor's contribution to the field of art as described in the patent specification.").

[186] 35 U.S.C. §112 ¶ 2.

[187] *See* Bell Communications Research v. Vitalink Communications Corp., 55 F.3d 615, 619 (Fed. Cir. 1995); Constant v. Advanced Micro-Devices, Inc., 848 F.2d 1560, 1571 (Fed. Cir. 1988).

[188] Other claims formats, such as the product-by-process claim format, are used relatively infrequently and are not relevant to the present discussion. A product-by-process claim characterizes a product in terms of the process by which it is made. The proper interpretation of product-by-process claims is currently unclear due to conflicting

function" claim format.[189] The general rule is for a product to be claimed in terms of its physical structure.[190] Although a product may be claimed in other ways, such as in terms of its function or the process by which it is made,[191] the scope of a product claim is limited to the physical structure of the invention for purposes of determining novelty, nonobviousness, and infringement, regardless of the claim format that is chosen to claim the invention.

In other words, a product claim covers what a product *is*, not what the product *does*.[192] Consider, for example, the interpretation of claims written using structural language and those using functional language. If a claim to a product claims the product in terms of its physical structure, the literal scope of the claim is limited to products having the claimed physical structure, because the scope of the invention is defined by the scope of the claims.[193] To infringe

decisions from the Federal Circuit in Scripps Clinic & Research Found. v. Genentech, Inc., 927 F.2d 1565, 1583 (Fed. Cir. 1991) (holding that product-by-process claims are not limited in scope to products produced by the process recited in the claims) and Atl. Thermoplastics Co. v. Faytex Corp., 970 F.2d 834 (Fed. Cir. 1992) (holding that product-by-process claims are limited in scope to products produced by the process recited in the claims). For an overview of the law of product by process claims, *see* PATENTS, *supra* note 4 , at § 8.05, at 172; Brian S. Tomko, Comment, *Scripps Or Atlantic: The Federal Circuit Squares Off Over The Scope Of Product-By-Process Patents*, 60 BROOK. L. REV. 1693 (1995).

[189] 35 U.S.C. § 112 ¶ 6 ("An element in a claim for a combination may be expressed as a means or step for performing a specified function without the recital of structure, material, or acts in support thereof, and such claim shall be construed to cover the corresponding structure, material, or acts described in the specification and equivalents thereof.").

[190] *See, e.g., Atl. Thermoplastics Co., Inc.*, 970 F.2d at 845 (noting that "the PTO and the CCPA [have] acknowledged product-by-process claims as an exception to the general rule requiring claims to define products in terms of structural characteristics."). *See also* ROBERT C. FABER, LANDIS ON MECHANICS OF PATENT CLAIM DRAFTING (4th ed. 2002), at X-45 ("[W]hile an apparatus claim is not *merely* a catalog of parts, it *absolutely must include a catalog of* parts. It is the cataloging of the parts that gives rise to what we call the positive recitation of the elements. That is, each structural element in the claim must be set forth directly and independently of every other element."); Examination Guidelines for Computer-Related Inventions, *supra* note 158 ("For [claims to computer software] products, the claim limitations will define discrete physical structures. The discrete physical structures may be comprised of hardware or a combination of hardware and software.").

[191] *See supra* note 188.

[192] Hewlett-Packard Co. v. Bausch & Lomb Inc., 909 F.2d 1464, 1468 (Fed. Cir. 1990). *See also* FABER, *supra* note 190, at X-47 ("In apparatus claims, we are concerned solely and exclusively with a definition of structure. By structure I mean hardware, something you can touch, something you can feel, something you can manipulate. We are not interested in function, and the function of an element cannot be used as a substitute for the definition of the structure of the element itself.").

[193] See PATENTS, *supra* note 4, at 8-79 ("A 'true' product claim is one in which the product is defined in terms of structural characteristics only.").

such a claim, an accused product must include all of the elements of the claim.[194] Since the

elements of such a claim are defined in terms of the invention's physical structure, the accused

product would only infringe the claim if it had the physical structure recited by the claim.[195]

An inventor may write a claim for a physical product in terms of the functions the product

performs rather than in terms of its physical structure. Such a functionally-worded "means-plus-

function" claim to a pencil might describe the pencil as "means for writing coupled to means for

erasing." As described in more detail below,[196] even when a product claim is written in terms of

the functions performed by the invention, the literal scope of the claim is limited to the particular

physical structures employed by the invention to perform such functions. The claim is invalid if

the specification fails to disclose any such particular physical structures.[197] The literal scope of a

product claim, therefore, is limited by the physical structure of the product regardless of the form

in which the product is claimed.

An accused product, as a general rule, therefore does not infringe a product claim if the

physical structure of the accused product differs from the physical structure of the claimed

product, even if the accused product and the claimed product perform the same function, achieve

the same result, or solve the same problem.[198] The primary reason for limiting the literal scope

of a product claim to the physical structure of the invention is that in the context of

---

[194] *See*, *e.g.*, Johnston v. IVAC Corp., 885 F.2d 1574, 1578 n.3 (Fed. Cir. 1989); Hybritech Inc. v. Monoclonal Antibodies, Inc., 802 F.2d 1367, 1379 (Fed. Cir. 1986) ("It is axiomatic that for prior art to anticipate under § 102 it has to meet every element of the claimed invention . . .").

[195] *See, e.g.,* SRI Int'l v. Matsushita Elec. Corp., 775 F.2d 1107, 1118 (Fed. Cir. 1985). One exception to this general rule is the "reverse doctrine of equivalents," according to which a product that has the literal structure recited in a claim may avoid infringement if the product works in a substantially different way than the patented invention. *Id.*

[196] *See supra* Section IV.C.

[197] *See In re* Donaldson Co., 16 F.3d 1189, 1195 (Fed. Cir. 1994) (en banc).

electromechanical inventions the disclosure of a product having a particular physical structure typically only enables the public to make and use products having the same or similar physical structures. To allow an inventor, therefore, to obtain patent protection for any and all products that perform a particular function based merely on the disclosure of a single product for performing that function would be to grant the inventor a scope of protection that is broader than the scope of the invention.[199]

Consider, for example, the *Morse* case, in which the Supreme Court invalidated Morse's claim 8, which stated:

> I do not propose to limit myself to the specific machinery or parts of machinery described in the foregoing specification and claims; the essence of my invention being the use of the motive power of the electric or galvanic current, which I call electro-magnetism, however developed for marking or printing intelligible characters, signs, or letters, at any distances, being a new application of that power of which I claim to be the first inventor or discoverer.[200]

Claim 8 essentially claimed any device, regardless of its physical structure, that performed the function of transmitting messages using electricity. The Supreme Court invalidated claim 8 in part because the facial scope[201] of the claim encompassed *any* device that performed the function of electrically transmitting messages, even though Morse had only invented and disclosed a

---

[198] The primary exception to this rule is that the accused product may infringe the claim under the "doctrine of equivalents." *See generally* Festo Corp. v. Shoketsu Kinzoku Kogyo Kabushiki Co., 532 U.S. 722 (2002); PATENTS, *supra* note 4, at § 18.04.

[199] In general, the scope of a patent scope should be no greater than the scope of enablement. *See, e.g.*, *In re* Moore, 58 C.C.P.A. 1042, 1047 (C.C.P.A. 1971) ("The relevant inquiry may be summed up as being whether the scope of enablement provided to one of ordinary skill in the art by the disclosure is such as to be commensurate with the scope of protection sought by the claims."); *accord In re* Hogan, 559 F.2d 595 (C.C.P.A. 1977).

[200] O'Reilly v. Morse, 56 U.S. 62, 112 (1854).

[201] As used herein, the "facial scope" of a claim refers to the scope of a claim that is defined by the plain meaning of the claim as worded. For example, the facial scope of a pure means-plus-function claim encompasses any and all devices that perform the functions recited in the claim, because the plain meaning of the claim as worded encompasses any and all such devices. As used herein, the "actual scope" of a claim refers to the scope of the claim as correctly interpreted under all applicable patent law principles. For example, although the facial scope of a pure means-plus-function claim encompasses any and all devices that performed the functions recited in the claim, the

particular device for performing this function.[202] In other words, the basis for invalidating the

claim was that the facial scope of the claim was broader than the scope of enablement.[203]

If *Morse* were decided today, the functionally-worded claim 8 would not have been

invalidated. Rather, it would be held valid because Morse did in fact invent a product that

performed the claimed function and since the specification disclosed particular physical

structures for performing that function. The scope of the claim, however, would be limited to the

particular physical structures disclosed in the patent specification and equivalents thereof

pursuant to 35 U.S.C. § 112 ¶ 6.[204] The purpose of 35 U.S.C. § 112 ¶ 6, in fact, is to "cut back"

on the facial breadth of functionally-worded claims, so that they provide a scope of protection

that is limited to the physical structure of the invention and therefore commensurate with the

scope of enablement.[205]

Functional claim language is particularly likely to result in broad facial claim scope in the

case of pioneer inventions, such as Morse's telegraph, and in new fields of technology, because

in such cases there is little or no prior art to limit the scope of the claims. In the absence of prior

art, only the physical structure of the invention itself and the limitations of structural terms in the

English language are available to limit claim scope. If functionally-worded claims to pioneer

---

actual scope of such a claim is limited (by 35 U.S.C. § 112 ¶ 6) to the particular physical structure recited in the specification for performing the recited functions and equivalents thereof.

[202] See *O'Reilly*, 56 U.S. at 113.

[203] *See id.* at 119-20.

[204] *See* Valmont Indus., Inc., v. Reinke Mfg. Co., 983 F.2d 1039, 1042 (Fed. Cir. 1993).

[205] *See* Johnston v. IVAC Corp., 885 F.2d 1574, 1580 (Fed. Cir. 1989); O.I. Corp. v. Tekmar, 115 F.3d 1576, 1583 (Fed. Cir. 1997). Courts historically have been skeptical of functional claim language due, *inter alia,* to its facial breadth. In Halliburton Oil Well Cementing Co. v. Walker, 329 U.S. 1 (1946), for example, the U.S. Supreme Court invalidated a patent claim to an apparatus for measuring the depth of oil wells because the invention was claimed using functional language "at the point of novelty." *Id.* at 8. The available evidence indicates that 35 U.S.C. § 112 ¶ 6 was enacted to overrule Halliburton's prohibition against the use of functional language in patent claims, while ensuring that functional claim language is limited in scope to the particular physical structures recited in the patent specification and equivalents thereof. *See* P.J. Federico, *Commentary on the New Patent Act*, 35 U.S.C.A. 1, 25-26 (1954), *reprinted at* 75 J. Pat. & Trademark Off. Soc'y 161 (1993).

inventions and to inventions in new fields were afforded their facial scope, such claims would

potentially be broad enough to encompass an extremely wide range of future innovations in the

same field, even though such innovations had not been enabled by the initial inventor. Although

the inventor of a pioneer invention may attempt to claim his invention using *structural* terms that

are as broad as possible, there are only so many existing structural terms in the English language

from which to choose that are well-understood by those of ordinary skill in the art, and existing

structural terms that are well-understood by those of ordinary skill in the art are inherently

limited in scope. Therefore, although an inventor may use broad structural terms to obtain

relatively broad patent protection for a pioneer invention, the prohibition against using functional

terms to obtain protection for *all* products that perform the claimed function imposes an upper

limit on the scope of functionally-worded product claims.

D. Conclusion: Patent Law's Emphasis on Physicality is Consistent with the Conventional Electromechanical Design Process

It makes sense for patent law to emphasize physicality in the conception, description, and

claiming of electromechanical inventions because historically it has been necessary for

electromechanical inventors to: (1) conceive of the physical structure of a product design to

manufacture a working physical embodiment of the product; (2) describe the product in terms of

its physical structure to enable others to make and use working physical embodiments of the

product; and (3) claim the product in terms of its physical structure to limit the scope of the

claims to the scope of enablement. The current manifestations of the rules of patentability and

claim scope therefore assume that the inventions to which they apply are invented using a process

in which physical structure plays a central role.[206] Although this assumption is valid in the

context of conventional electromechanical inventions, it is not valid in the context of software

inventions. Therefore, problems arise when the traditional rules of patentability and claim scope

are applied to software.

E. Problems with Current Software Patent Jurisprudence

1. The Formal Physicality Requirement

Most of the debate over software patentability has focused on whether software qualifies

as statutory subject matter.[207] The courts,[208] for example, have attempted to apply patent law's

physicality requirements directly to software, particularly by premising subject-matter

patentability on the mere presence or absence of physical structural terms in software patent

claims and specifications.[209] Despite occasional statements by judges that it is necessary to look

---

[206] Patent law expressly disavows any requirement that an invention be invented in a particular way to merit patent protection. *See, e.g.,* 35 U.S.C. § 103(a) ("Patentability shall not be negatived by the manner in which the invention was made."). Despite this fact, the rules of patentability and claim scope reflect implicit assumptions about requirements of real-world inventive processes.

[207] *See generally* references cited in *supra* note 2.

[208] Statements made herein regarding the courts are also applicable to the United States Patent and Trademark Office (USPTO) insofar as the USPTO interprets and influences the direction of substantive patent law.

[209] *See, e.g., In re* Bernhart, 417 F.2d 1395 (C.C.P.A. 1969) (means-plus-function claim held to be patentable subject matter due to recitation of physical terms such as "computer"); *In re* Noll, 545 F.2d 141 (C.C.P.A. 1976) (means-plus-function claims held to be directed to statutory subject matter when the specification recited conventional computer hardware elements in conjunction with a computer program for performing the functions recited in the claims); *In re* Taner, 681 F.2d 787 (C.C.P.A. 1982) (method claims held to be directed to more than a mere mathematical algorithm and therefore constitute statutory subject matter where the claims recited application of the method to seismic energy waves and other physical entities); *State Street Bank & Trust Co.*, 149 F.3d at 1368 (means-plus-function claim held to be patentable subject matter due to physical structure imported into claim by reference to specification); *Diehr*, 450 U.S. at 188 (mathematical equation applied to a process for curing rubber held to be patentable subject matter); *In re* Alappat, 33 F.3d 1526, 1577 (means-plus-function claim held to be patentable subject matter due to physical structure imported into claim by reference to specification); *In re* Warmerdam, 33 F.3d 1354 (Fed. Cir. 1994) (affirming rejection of method claims which recited no physical structure and reversing rejection of product claim directed to a "machine"); *In re* Lowry, 32 F.3d 1579 (Fed. Cir. 1994) (holding claim directed to a "memory" to be statutory subject matter even though claim defined contents of memory in terms of logical entities); *In re* Trovato, 42 F.3d 1376 (Fed. Cir. 1994), *vacated by* 60 F.3d 807 (Fed. Cir. 1995) (means-plus-function claims which recited invention in terms of functional elements held to be nonstatutory subject matter due to specification's failure to recite any corresponding structure); *State St. Bank & Trust Co.*, 149 F.3d at 1373 (means-plus-function claim held to be statutory subject matter due to definition of invention in terms of physical components in specification). *Cf.*, Parker v. Flook, 437 U.S. 584, 594 (1978) (holding method for updating

---

beyond the formal features of the claims,[210] the holdings generally indicate that claims that

include "physical" terms (such as "machine" or "memory") will satisfy the statutory subject

matter requirement, while claims lacking such magic words will fail to satisfy the requirement.[211]

This is what I refer to as the "formal physicality requirement," because it looks merely to the

form of the claim rather than to the underlying nature of the claimed invention.

>   2. The Formal Physicality Requirement is Not a Good Test for Software
>   Patentability

Prior to the advent of software, the absence of any "physical" terms from a claim – such

as "machine," "gear," "leg," "engine," or even "computer" – was a good indicator that the

inventor either had not invented a patentable product or process or was attempting to claim more

than he had invented. In the former case, the lack of physical terms would indicate that the

inventor had not yet invented a particular physical product or process, but perhaps had only

devised a mere "wish" or "plan" for a product or process. In the latter case, in which the inventor

had truly invented an electromechanical device, but chose to describe it solely in terms of its

---

an alarm limit on a catalytic chemical conversion of hydrocarbons not to be statutory subject matter despite reference in claim to "the catalytic chemical conversion of hydrocarbons"); *In re* Christensen, 478 F.2d 1392 (C.C.P.A. 1973) (holding method for determining the porosity of a subsurface formation in situ to be non-statutory subject matter because the claimed invention's sole point of novelty was a mathematical formula); *In re* Walter, 618 F.2d 758 (C.C.P.A. 1980) (holding method for cross-correlating electrical signals representing seismic waves not to be directed to statutory subject matter).

[210] The courts have recognized the need to look beyond the language of the claims, asserting that "semantogenic considerations preclude a determination based solely on words appearing in the claims. In the final analysis under § 101, the claimed invention, as a whole, must be evaluated for what it is." *In re* Sarkar, 588 F.2d 1330, 1333 (C.C.P.A. 1978). This recognition has not stopped the courts, however, from applying a formal analysis based solely or primarily on the language of the claims in many cases.

[211] *See, e.g.,* Goodman et al., *supra* note 3, at 357-78 (arguing that the holding in *In re* Trovato implies that a software invention may satisfy the statutory subject matter requirement merely by claiming the invention in terms of a functionally equivalent hardware implementation); Durham, *supra* note 92, at 1485 (arguing that, according to the *Alappat* decision, "[a]ll the applicant must do . . . is draft the claims in terms of the physical hardware that performs the steps of the algorithm, and the result will be a 'new machine.'"); Shawn McDonald, *Patenting Floppy Disks, or How the Federal Circuit's Acquiescence has Filled the Void Left by Legislative Inaction*, 3 VA. J.L. & TECH. 9, 38 (1998).

function or other logical properties, such a description would tend to encompass a range of embodiments that had not been enabled by the inventor.

The absence of physical terms in a software claim, however, does not by itself indicate that the inventor has not invented a software program capable of being embodied in a physical form or that the inventor is attempting to claim more than he has invented. For example, as described above, computer programmers may conceive of programs in purely logical terms and enable others to make and use working physical embodiments of the program by describing such programs in purely logical terms. Similarly, a software claim may use purely logical terms to particularly point out the useful, novel, and nonobvious features of the claimed program.

Requiring software inventors to include physical structural terms in software patents may result in false negatives, i.e., denial of patent protection to software that should receive such protection. As described above,[212] in most cases it is not even *possible* for a programmer to conceive of or describe a program in terms of its physical structure and thereby satisfy the formal physicality requirement. Furthermore, in the case of all but the most simple programs, describing a program in terms of its physical structure would not enhance the public's ability to make and use, or even to understand, the invention.

Programmers, in other words, have no practical choice but to describe and claim their programs in terms of their logical structure. The use of the means-plus-function claim format has been described as a "choice" that is made available to the claim drafter as a drafting "convenience,"[213] so that the drafter may simply claim an element in terms of what it does rather

---

[212] *See* Section III.D.
[213] *See supra* n.205.

than in terms of its physical structure.[214] A modern-day Morse, for example, would have the

legal and factual option of describing and claiming the telegraph either in terms of its physical

structure or in terms of the functions it performs. Although the law may offer software inventors

the same *formal* choice, as a practical matter the only option available to software inventors is to

describe their inventions in logical terms because it is not possible to describe them in physical

terms.

Furthermore, the historical skepticism towards functionally-worded claims is based in

part on the suspicion that patent applicants use such claims in a conscious attempt to secure a

broader range of protection than is warranted.[215] Although software patent claim drafters may be

guilty of such a sin in particular cases, the mere fact that a software patent claim is written in

purely functional language does not imply any malicious intent on the part of the claim drafter.

Rather, it may merely reflect an honest attempt to claim a software invention in the only terms in

which it is reasonably susceptible to being claimed. Patent law has long allowed product

inventions that are not susceptible to being claimed in terms of their physical structure to be

claimed in other terms, such as in terms of the process by which the invention is made.[216] Such

an exception to the general rule of structural claiming is based at least in part on the theory that

patent protection should not be denied to an invention merely because it is not susceptible to

description in conventional terms.[217]

---

[214] *See, e.g.,* O.I. Corp. v. Tekmar, 115 F.3d 1576, 1583 (Fed. Cir. 1997); *In re* Alappat, 33 F.3d at 1583 (Plager, J. concurring).

[215] *See supra* Section IV.E.2.

[216] *See* Atlantic Thermoplastics, 947 Co. v. Faytex Corp., 974 F.2d 1279, 1283-4 (Fed. Cir. 1992) (Newman, J., dissenting) (recognizing the long-standing acceptance of product-by-process claims). "The premise of such claims has been called the Rule of Necessity, for it provides a way of patenting inventions or discoveries whose structure is not sufficiently known or knowable to be described objectively." *Id.* at 1279.

[217] *See, e.g., In* re Painter, 1891 C.D. 200, 57 O.G. 999 (Comm'r of Pats. 1891) (holding that the right to a patent should not be denied, and that the use of the product-by-process claim format is appropriate, in the cases of an

The terms that typically are used to describe computer programs – such as "data," "record," "field," "array," and "list" – may be interpreted as referring either to logical entities or to the particular physical entities (e.g., electrical signals) in which such logical entities are embodied. Whether a term such as "record" refers in a particular case to a logical (abstract) data record or to a physical record stored in the memory of a computer cannot be ascertained merely by engaging in the formal process of analyzing the meaning of the term "record" in the abstract. Rather, it is necessary to analyze the meaning of the term in context to determine whether the patent applicant has enabled the public to make and use a physical record in the manner claimed.[218] Even terms that refer solely to numbers may refer to physical entities if their context indicates that they refer to physical entities (such as electrical signals or switch settings) that instantiate such numbers. These observations indicate that there is no useful distinction to be drawn between "physical terms" and "functional terms" or "logical terms" in the context of software patent claims, because so-called "logical terms" may actually refer to physical entities when they are used to describe executable software programs.

The mere absence of "physical" terms in the description and claims of a software patent, therefore, is not a reliable indicator that the claims are not directed to statutory subject matter.[219] Applying the formal physicality requirement to pure software claims[220] may therefore result in

invention which "cannot be properly defined and discriminated from prior art otherwise than by reference to the process of producing it"); *accord* Funk Bros. Seed Co. v. Kalo Inoculant Co., 333 U.S. 127, 138 (1948) (Burton, J., dissenting); *In re* Bernhart, 417 F.2d 1395, 1399 (C.C.P.A. 1969). Furthermore, "[w]hen mathematical formulae are the standard way of expressing certain functions or apparatus, it is appropriate that mathematical terms be used" to claim such apparatus. Arrhythmia Research Tech., Inc. v. Corazonix Corp., 958 F.2d 1053, 1060 (Fed. Cir. 1992).

[218] *See infra* Section V.(2)b.

[219] *See* Goodman et al., *supra* note 3, at 357 ("It is no wonder that in the patent applications of *Trovato*, '*no computer architecture is provided, no circuit diagram is revealed, and no hardware at all receives more than brief mention.*' These are completely unnecessary for programming a computer, and programming a computer inherently makes the circuitry – this is how a computer operates.") (emphasis in original).

[220] By a "pure" software invention I mean one which the inventor has conceived of solely in terms of its logical structure and which may be enabled by a description couched solely in terms of the invention's logical structure.

false negatives, i.e., a finding that particular computer programs are not patentable subject matter, despite the fact that the programs themselves are embodied in a physical form and perform useful functions.[221]

Using formal physicality as the *sole* criterion for the statutory subject matter determination may also result in false positives, i.e., findings that particular computer programs are statutory subject matter merely because they are embodied in a physical form and described in physical terms, even though they perform no useful function. Consider, for example, a phonograph record whose novelty lies solely in the originality of the song recorded on it.[222] The phonograph record has a particular physical structure and may even be described and illustrated in terms of that physical structure, although perhaps with some practical difficulty. Such a phonograph record should not, however, qualify as statutory subject matter because it does not have "practical utility."[223] The same is true of software lacking practical utility. If formal physicality were the sole test for subject-matter patentability, however, a claim to such a phonograph record (or software) written in physical terms would incorrectly qualify as statutory subject matter.

---

[221] Electrical signals may qualify as physical structure for purposes of satisfying the subject-matter requirements of 35 U.S.C. § 101. *See In re* Sherwood, 613 F.2d. 809, 819 (C.C.P.A. 1980) ("seismic traces [recited in the claims] are electrical signals from geophones, i.e., *physical* apparitions, or particular patterns of magnetization on magnetic tape, i.e., the pattern of the magnetization being a *physical* manifestation, or a *physical* line on a paper chart"); *Arrhythmia Research*, 958 F.2d at 1053 (holding that a process for manipulating and analyzing electrocardiograph signals constituted statutory subject matter).

[222] *See Examination Guidelines for Computer-Related Inventions*, *supra* note 158, at 7478-7492 for analysis of similar examples in the software context.

[223] State St. Bank & Trust Co. v. Signature Fin. Group, 149 F.3d 1368, 1375 (Fed. Cir. 1998).

3. Current Jurisprudence Results in Overbroad and Underbroad Software Patent Claims

The extent of software patent claim scope has received little, if any, attention from the Federal Circuit.[224] The question of software patent claim scope is particularly important, not only because of the technological significance of software and the potential economic impact of overbroad or underbroad software patent claims,[225] but also because software patent claims tend to be written using functional language which, as noted above,[226] may require the scope-limiting mechanisms of 35 U.S.C. § 112 ¶ 6 to be held in check.[227]

For example, in some cases patent protection may not be available or be available only narrowly to otherwise worthy software inventions which do not clearly fit within a particular category of statutory subject matter. Although algorithms tend to fit squarely within the statutory "process" mold, object-oriented software programs often do not. Each object defined by an object-oriented software program may include both procedures (referred to as "methods") and data (referred to as "properties").[228] An object-oriented software program is not best explained either as a pure process or as a pure product, but rather as a complex mixture of both process and data.

Attempts to claim an object-oriented program as a pure process or a pure product will likely result in a claim that emphasizes the program's process-like features over its product-like features, or *vice versa*. Writing separate claims to the program in product and process forms

---

[224] The question of claim scope in general has received much less attention than the question of patentability. *See generally* Cohen & Lemley, *supra* note 143, at 1 (arguing for narrow interpretation of software patent claims); Merges & Nelson, *supra* note 143, at 840 (noting the relative dearth of scholarly writing on patent claim scope).
[225] Because the proper scope of a claim is one that is commensurate with the scope of enablement, a claim is overbroad if its scope is broader than the scope of enablement and underbroad if its scope is narrower than the scope of enablement.
[226] See *supra* Section IV.C.
[227] *See, e.g.,* Stern, *supra* note 25, at 185.

might not solve the problem if the program's utility, novelty, and nonobviousness results from a synergy of the program's product-like and process-like features. Attempts to claim an object-oriented program using pure product and/or process claims, therefore, is likely to result in less protection than is warranted, either because the claim is held unpatentable or because the resulting claim scope is narrower than the scope of enablement.

In other cases, software patent claim scope may be too broad. Consider, for example, that the scope of a patent claim is limited primarily by two factors: the relevant prior art[229] and the physical structure recited in the claims and/or the specification.[230] There is a limited amount of software prior art available due to the relative novelty of the field of computer science and its rapid rate of progress. Although patent law often absorbs new technological fields without difficulty, computer programming is the first branch of engineering in which working physical embodiments may be enabled by descriptions expressed solely in logical terms. The universe of software prior art, therefore, consists primarily[231] of processes and other logical entities that may be enabled by descriptions that lack any reference to physical structure. The universe of this

---

[228] For more information about object-oriented software, see *supra* Section III.D.3.

[229] For example, a claim that characterizes an otherwise novel design so broadly that it encompasses designs within the prior art is not novel and therefore not patentable. *See, e.g.*, Kalman v. Kimberly-Clark Corp. 713 F.2d 760, 770-71 (Fed. Cir. 1983). Furthermore, an issued patent claim is presumed valid (35 U.S.C. § 282 (2003)) and should be construed to avoid encompassing prior art. Modine Mfg. Co. v. United States Int'l Trade Comm'n, 75 F.3d 1545, 1557 (Fed. Cir. 1996); Texas Instruments Inc. v. United States Int'l Trade Comm'n, 871 F.2d 1054, 1065 (Fed. Cir. 1989). The prior art therefore limits both patentability and claim scope.

[230] *See* Section IV.C, *supra*. Software patents may also be susceptible to particularly broad interpretations under the doctrine of equivalents. *See* Cohen & Lemley, *supra* note 143, at 39-50.

[231] I say "primarily" because a process that applies to and is defined in terms of a particular physical structure (such as a material on which the process operates) may serve as prior art against a subsequent software invention that is defined without reference to physical structure. Furthermore, a physical product may serve as prior art against a software program in the (rare) case that the product embodies the same logical structure as the software program.

class of prior art is very small, because it has only been the advent of computer programming that

has made it possible to populate that universe in quantity.[232]

Furthermore, and perhaps more significantly, the typical absence of physical structural

terms from software patent claims means that such terms are not present to limit the scope of the

claims, as they do in the case of electromechanical claims. The facial scope of a software claim

written in purely functional terms encompasses *all* physical entities that perform the claimed

function. In the context of software patents, therefore, patentability seems to be more a game of

all-or-nothing than one in which claim scope is commensurate with the scope of invention.[233]

Although one might think that § 112 ¶ 6 would limit the scope of such claims, the Federal Circuit

has said little about the application of the claim-limiting effect of § 112 ¶ 6 to software product

claims[234] and even less about the application of § 112 ¶ 6 to process claims in general.[235]

V. How Should Intellectual Property Law Protect Software in Light of its Unique Qualities?

The fourth, and final, question in the current analysis, now reformulated in light of the

discussion above, is: "How should patent law treat software in light of the differences between

---

[232] This lack of relevant prior art may cause claims for incremental software inventions to have the relatively broad scope typically affording only to "pioneer" inventions. *See, e.g.,* Symposium, *Internet Business Model Patents: Obvious by Analogy*, 7 MICH. TELECOMM. & TECH. L. REV. 253 (2001). For a definition of "pioneer patent," *see* Westinghouse v. Boyden Power Brake Co., 170 U.S. 537, 561-62 (1898).

[233] The same may be true of business method patents. *See, e.g.,* Symposium, *Scope-of-Protection Problems with Patents and Copyrights on Methods of Doing Business*, 10 FORDHAM INTELL. PROP. MEDIA & ENT. L.J. 105, 111-112 (1999).

[234] For example, although the Federal Circuit held that the means-plus-function claim in *Alappat* was directed to statutory subject matter, the court did not elaborate on the range of physical structures that are enabled by a specification which merely indicates that the invention may be practiced on a "general-purpose computer." Query, for example, whether such a claim would be infringed by a device performing the claimed functions but having a radically different physical structure, such as a biocomputer. *See, e.g.*, Texas Instruments v. United States Int'l Trade Comm'n, 805 F.2d 1558, 1572 (Fed. Cir. 1986) (holding accused device to be noninfringing which performed the functions claimed in a means-plus-function claim using subsequently developed means which differed significantly from the means disclosed in the patent specification).

[235] *See generally* Brad A. Schepers, Note, *Interpretation of Patent Process Claims in Light of the Narrowing Effect of 35 U.S.C. § 112(6),* 31 IND. L. REV. 1133 (1998).

software and other electromechanical devices?" Although the law currently allows software to

be patented,[236] the current contours of patent law fit software poorly for the reasons described

above.

Although it is necessary to embody a computer program in a physical form having a

particular physical structure, it is not necessary to describe or claim the program in terms of its

physical structure to enable the program to be made and used. In particular, the novel,

nonobvious, and useful features of a software program may be described clearly and with

particularity without using physical terms. As a result, software patent specifications and claims,

unlike their conventional electromechanical counterparts, should not need to recite physical

structure to satisfy the requirements for patentability or to provide a basis for limiting the scope

of software claims to the scope of invention.

My proposal is to modify the rules of patentability and claim scope by expressly allowing

software programs to be patented solely in terms of their logical structure. In particular, I

propose that:

(1) a software program be claimable solely in terms of its logical structure;

(2) a software program be patentable if:

    a. the inventor provides a written description of the claimed logical structure;

    b. the inventor provides a description that enables one of ordinary skill in the art

       to make and use the claimed program without undue experimentation;

---

[236] Currently, software programs may constitute statutory subject matter either as machines, articles of manufacture, or processes, depending upon the form in which they are claimed. For examples of each, *see, e.g., In re* Alappat, 33 F.3d 1526 (Fed. Cir. 1994) (claiming software as a machine); *In re* Beauregard, 53 F.3d 1583 (Fed. Cir. 1995), *described in* GREGORY A. STOBBS, SOFTWARE PATENTS, 172-77 (2d ed. 2000) (claiming software as electromagnetic

c. the claimed logical structure has a practical utility;

d. the inventor conceives of the claimed logical structure;

e. the claimed logical structure is novel and nonobvious; and

(3) the scope of a software patent claim be limited to products and processes that embody the claimed logical structure.

In effect, I propose that the traditional rules of patentability and claim scope be updated to apply to the logical structure of software programs in the same way that they currently apply to the physical structure of conventional electromechanical product inventions.[237] I will address each of these sub-proposals in turn.

A. Allow Software Programs to be Claimed in Terms of their Logical Structure

I propose a new category of statutory subject matter: a computer program, which would be claimable directly and solely in terms of its logical structure, without reference to the physical structure of the program's embodiments or the physical structure of the hardware on which the program may execute. In the case of a conventional algorithm, such a claim would take much the same form as a traditional process claim because algorithms are a kind of process, which are in

---

signals tangibly stored in a computer-readable medium such as a floppy diskette); Arrhythmia Research Technology, Inc. v. Corazonix Corp., 958 F.2d 1053, 1058 (Fed. Cir. 1992) (claiming software as a process).

[237] One reason to base software patent protection on the legal rules that have developed for product inventions, rather than process inventions, is that the law of product patents is more well-developed and well-settled than the law of process patents. The courts historically "have had more conceptual problems with process claims than with product claims." PATENTS, *supra* note 4, at § 1.03. The law of product inventions therefore serves as a more stable foundation from which to build a framework of protection for software. More generally, to the extent that the process that is used to invent product inventions is analogous to the process used to design the logical structure of software programs, and to the extent that the traditional rules of patentability and claim scope may be translated to protect logical entities in a manner that is analogous to the way in which patent law traditionally has protected physical entities (products), such a translation is justified. Rules derived in this way to apply to logical entities are equally applicable to processes because a process is a special case of a logical entity.

turn a kind of logical entity. For example, a claim to a process for receiving two numbers as input and adding them might be written as follows:[238]

    1.   A computer program comprising:

    receiving a first number;

    receiving a second number; and

    adding the first number to the second number.

A claim to a data structure for storing information about an email account might be written as follows:

    2.   A computer program comprising:

    a username for use with an email account of a person;

    a password corresponding to the username;

    an address of an outgoing email server accessible using the username and password; and

    an address of an incoming email server accessible using the username and password.

---

[238] The claims provided in this section are provided merely to illustrate the form that claims would be allowed to take according to the rules proposed herein, and are not intended to satisfy all of the requirements for patentability, such as novelty, nonobviousness, and utility.

A claim to a software object representing a used automobile according to the object-oriented programming paradigm might be written as follows:[239]

   3.   A computer program representing an automobile, the computer program comprising:

      properties including:

         a manufacturer of the automobile;

         a model of the automobile;

         a date of manufacture of the automobile;

         a price for which the automobile was purchased;

         a condition of the automobile; and

      methods including:

         a method for calculating a current value of the automobile based on the manufacturer, model, date of manufacture; price, and condition; and

         a method for estimating a current mileage of the automobile based on the date of manufacture.

---

[239] For an argument that object-oriented software should be considered patentable subject matter, *see* Keith Stephens & John P. Sumner, *Software Objects: A New Trend in Programming and Software Patents*, 12 SANTA CLARA COMPUTER & HIGH TECH. L.J. 1 (1996).

B. Requirements for Patentability

1. Statutory Subject Matter

I propose that the formal physicality requirement (i.e., the requirement that a claim

include physical terms) be replaced by a substantive physicality requirement in the context of

software claims. The formal physicality requirement places an unnecessary and unreasonable

burden on software inventors[240] and should be eliminated in the context of software claims

because of the defects described above in Section IV.E.

The substantive physicality requirement I propose would require that subject-matter

patentability be limited to physical software products and to software processes that work a

physical transformation on physical entities. Such a requirement would ensure that only software

claims that are directed to machine, manufactures, compositions of matter, articles of

manufacture, and processes that produce a concrete and tangible result qualify as statutory

subject matter.

In particular, the substantive physicality requirement would require that the patent

applicant enable those of ordinary skill in the art to implement the claimed invention in a

physical product (such as an executable software program) or in a process that works a physical

---

[240] At least one early court decision placed such a burden on patent applicants. *In re* Prater, 415 F.2d 1393
(C.C.P.A. 1969). The Federal Circuit, however, gradually began to realize that patent protection should not be
denied to certain inventions merely because they "operate according to," and are claimed in terms of, an algorithm.
See, e.g., In re Iwahashi, 888 F.2d 1370, 1375 (Fed. Cir. 1989). In *In re* Alappat, 33 F.3d at 1540, the Federal
Circuit noted that the lack of structure in the specification in cases such as *Abele*, *Pardo*, and *Walter* justified
interpreting the apparatus claims in those cases as if they were method claims, and pointed to the presence of
structure in Alappat's specification as a reason to interpret Alappat's apparatus claims as apparatus claims. *Id.* at
1541. Furthermore, the court held Alappat's claims to be directed to statutory subject matter because of the structure
that Alappat recited in the specification to support the claims. Id. at 1543-44. The court's reasoning in *Alappat*
implicitly placed a burden on the patent applicant to recite structure at a fairly detailed level in the patent application
to support the elements of a means-plus-function claim, perhaps even if the specification would have been enabling
absent the recitation of such structure.

transformation upon physical entities.[241] In other words, the enablement requirement would serve as a proxy for the substantive physicality prong of the statutory subject-matter requirement.

The statutory subject matter requirement should also retain the requirement of practical utility, as is required by developing Federal Circuit jurisprudence.[242] This approach, however, begs the question: what is *practical* utility?

For purposes of this discussion, however, it is not necessary to determine precisely the outer bounds of the practical utility requirement, because the focus of this paper is on clear cases in which physically-embodied software programs work physical transformations on physical entities and perform useful functions. One such class of clear cases is the class of software programs that perform the same functions as pre-existing electromechanical machines having practical utility, such as software for controlling robot arms, regulating automobile brakes, and monitoring room temperature. Such programs should be considered to have practical utility for the same reasons as their electromechanical counterparts.[243] Although cases such as *State Street Bank* raise extremely difficult and important questions about the scope of the "useful arts," the solutions proposed herein may be applied independently of the answers to those questions.

---

[241] This is consistent with current Patent Office practice, according to which the fact that a program performs a physical transformation within a computer "alone does not distinguish a statutory computer process from a non-statutory computer process. What is determinative is not how the computer performs the process, but what the computer does to achieve a practical application." *Examination Guidelines for Computer-Related Inventions*, *supra* note 158, at 7484.

[242] *See State St. Bank & Trust Co.*, 149 F.3d. at 1375 (the statutory subject matter determination does "not focus on which of the four categories of subject matter a claim is directed to . . . – process, machine, manufacture, or composition of matter – but rather on the essential characteristics of the subject matter, in particular, its practical utility.").

[243] *See* Durham, *supra* note 92, at 1423 ("My own proposal is to recognize that the programmer's art is the art of converting an often non-technological plan (such as a particular scheme for managing a family of mutual funds) into the kind of logical structure executed by a computer. Fashioning a logical structure is, like fashioning a physical structure, a 'technological' endeavor, at least when the purpose is to produce a useful computer program.").

2. Written Description

The written description requirement should require only that a software patent specification contain a written description of the logical structure of the claimed software. In cases in which a program is claimed in terms of the functions it performs rather than in terms of its particular logical structure, the specification would need to provide a description of the logical structure of the claimed program to avoid invalidity under 35 U.S.C. § 112 ¶ 6.[244]

This proposed written description standard is consistent with current Patent Office procedure. Although the general rule is that product inventions must be described in terms of their physical structure or other identifying physical properties, a product invention may be described in terms of "functional characteristics when coupled with a known or disclosed correlation between function and structure."[245] The disclosure of the logical structure of a computer program in such detail that one of ordinary skill in the art may implement the program without undue experimentation should qualify as the disclosure of "functional characteristics . . . coupled with a known . . . correlation between function and structure," because the program may be implemented in executable software using techniques that are well-known to those of ordinary skill in the art.

3. Enablement

I propose that current standard of enablement be applied without modification to computer program claims in the regime proposed herein. Under current law, a software patent claim is enabled if the specification enables a programmer in the appropriate field to make and

---

[244] For a detailed discussion of the relationship of 35 U.S.C. § 112 ¶ 6 to the proposals made herein, see *infra* Section V.C.

[245] *Guidelines for the Examination of Patent Applications Under the 35 U.S.C. § 112, ¶ 1 'Written Description' Requirement*, **§** II.A.3.a, U.S. Patent & Trade Office (2001).

use the invention by writing source code that satisfies the elements of the claim.[246] The Federal

Circuit has held that the enablement requirement does not necessarily require the specification to

include source code, but rather that flow charts, functional block diagrams, and natural-language

descriptions of the software may satisfy the enablement requirement in particular

circumstances.[247] Such descriptions of a software program in terms of its logical structure best

express the true scope of what the programmer has enabled others to make and use and should

therefore satisfy the enablement requirement.

4. Conception

The conception requirement should require only that the patent applicant conceive of the

logical structure of the claimed program. There should be no requirement that the inventor

conceive of any particular physical structure for implementing the program, because such a

requirement would be inconsistent with the nature of the process by which software is

invented.[248] Proving conception of an algorithm, for example, should require the same kind of

proof as that required for a conventional process invention, except that it should not require proof

that the inventor had conceived of any particular physical structure, perhaps other than a general-

purpose computer, for implementing the algorithm.

---

[246] *See*, *e.g.*, N. Telecom, Inc. v. Datapoint Corp., 908 F.2d 931, 941 (Fed. Cir. 1990), *cert. denied*, 498 U.S. 920 (1990).
[247] *See*, *e.g.*, Fonar Corp. v. GE, 107 F.3d 1543, 1549 (Fed. Cir. 1997); *In re* Hayes Microcomputer Prods. Inc. Patent Litigation, 982 F.2d 1527, 1534-38 (Fed. Cir. 1992); *In re* Donohue, 550 F.2d 1269, 1271 (C.C.P.A. 1977).
[248] *See supra* Section III.

5. Novelty and Nonobviousness

Patent law's *novelty* requirement requires that a product or process be new to merit patent protection.[249] A prior art product that satisfies all of the elements of a claim is said to "anticipate" the claim.[250] A product is novel only if it is not anticipated by any single prior art product.[251] Therefore, in a case in which the elements of a product claim define the product in terms of its physical structure, the claim only satisfies the novelty requirement if no single prior art source discloses all of the physical structure recited in the claim.[252]

I propose that software claims be interpreted to be limited to encompass only physical embodiments of the claimed invention for purposes of sections 102 (novelty), 103 (nonobviousness), and 271 (infringement) of the patent statute. Such a rule should apply both during patent prosecution and litigation. This would avoid the problem, addressed at length in several process patent cases,[253] of how to determine whether a claim that reads both on statutory subject matter and non-statutory subject matter satisfies the requirements of 35 U.S.C. § 101.

Computer program claims should be deemed novel if the claimed logical structure is novel.[254] Computer program claims should be interpreted to encompass any product or process that embodies the claimed program.[255] Consider, for example, the three kinds of computer program claims described above: (1) pure process-type claims, such as claim 1; (2) pure data

---

[249] *See* 35 U.S.C. § 102 (1994). The comments made herein with respect to the novelty requirement are equally applicable to patent law's "nonobviousness" requirement. *See id*. § 103.

[250] *See, e.g.,* Hybritech Inc. v. Monoclonal Antibodies, Inc., 802 F.2d 1367, 1379 (Fed. Cir. 1986).

[251] *See In re* Donohue, 766 F.2d 531, 533 (Fed. Cir. 1985).

[252] *See, e.g.,* Del Mar Eng'g Laboratories v. Physio-Tronics, Inc., 642 F.2d 1167, 1172 (9th Cir. 1981); Schroeder v. Owens-Corning Fiberglas Corp., 514 F.2d 901, 903-04 (9th Cir. 1975).

[253] *See, e.g., In re* Schrader, 22 F.3d 290 (Fed. Cir. 1994); *In re* Prater, 415 F.2d 1393 (C.C.P.A. 1969*)*.

[254] *See* Durham, *supra* note 92, at 1524.

[255] Whether a particular claim should be interpreted broadly to encompass *any* device or process that implements the claimed logical structure, or more narrowly to encompass only general-purpose computers that implement the claimed logical structure, is a difficult question that would need to be decided on a case-by-case basis. For a detailed description of claim scope according to the proposal put forth in this section, *see infra* Section V.C.

structure-type claims, such as claim 2; and (3) hybrid claims, such as claim 3. A pure process-type computer program claim should be interpreted as if it were *both* a conventional product claim drafted in means-plus-function format *and* a conventional process claim reciting a sequence of steps for purposes of novelty, nonobviousness, and infringement.[256] For purposes of determining novelty, for example, the claim should: (1) be interpreted as if the terms "means for" appeared in front of each element and then compared to the product prior art; and (2) be interpreted as if the terms "a step of" appeared in front of each element and then compared to the process prior art. The claim would be novel only if it were novel in light of both the product and process prior art. Similarly, the claim would be nonobvious only if it were nonobvious in light of both the product and process prior art. A process-type computer program claim would therefore be infringed both by a product that embodied the claimed process (such as a floppy diskette storing a computer program for performing the claimed process) and by the act of performing the process itself.[257]

A product claim written in means-plus-function format that does not include any physical limitations other than those inherent in any digital computer should be interpreted as a process claim in accordance with the rules just described.[258] A product claim that includes physical limitations beyond those inherent in any digital computer should be limited by the recited physical limitations.[259]

---

[256] The legal standard for infringement is the same as that for novelty. *See, e.g.,* PATENTS, *supra* note 4, § 3.02[1]; Knapp v. Morss, 150 U.S. 221, 228 (1893) ("[T]hat which infringes, if later, would anticipate if earlier.")

[257] A similar methodology for making patentability determinations for floppy disk claims is proposed in Richard H. Stern, *An Attempt to Rationalize Floppy Disk Claims*, 17 J. MARSHALL J. COMPUTER & INFO. L. 183, at 198-99 (1998).

[258] This is consistent with current Patent Office procedure. *See* Examination Guidelines for Computer-Related Inventions, U.S. Patent & Trademark Office, 61 Fed. Reg. 7478 (Feb. 28, 1996).

[259] *See id.*

A pure data structure-type claim should be interpreted as written and then compared to the product prior art, including prior art data structures.[260] The claim would be novel only if it were novel in light of the product prior art and be nonobvious only if it were nonobvious in light of the product prior art. Furthermore, the claim should be deemed obvious in light of any prior art *process* for *generating* the claimed data structure, even if the prior art reference (e.g., a patent or printed publication) did not recite the data structure itself.

Finally, in the case of a hybrid (e.g., object-oriented) computer program claim, each of the process-type elements of the claim should be interpreted in the same manner as the elements of pure process-type claims, and each of the data-type elements of the claim would be interpreted in the same manner as the elements of pure data-type claims. This would result in two different interpretations of such a claim for purposes of novelty, nonobviousness, and infringement, in which: (1) the claim is interpreted as written, so that process elements are interpreted as process elements and data elements interpreted as product elements; and (2) process elements are interpreted as product elements (by inserting "means for" in front of each process element) and data elements are interpreted as product elements.

C. Claim Scope

I propose that the scope of a computer program claim be limited to the logical structure of the claimed program. In cases, such as claims 1-3 above, in which the claim directly recites the program's logical structure, the scope of the claim should be commensurate with the claimed logical structure. Consider, for example, claim 1, which recites three steps that define the logical structure of the claimed program. Such a claim should be interpreted to encompass any computer

---

[260] See *In re* Lowry, 32 F.3d 1579, 1584 (Fed. Cir. 1994), for an example of a nonobviousness analysis comparing a claimed data structure to a prior art data structure.

program, whether stored on a computer-readable medium or in execution, which performs the

recited steps.

1. Software Claim Scope Should be Limited to the Scope of Enablement

Whether computer program claims should be limited in scope to embodiments

implemented using general-purpose computers is a difficult question. For example, should a

claim to an algorithm be interpreted narrowly to encompass only implementations of the

algorithm using a general-purpose computer, or more broadly to encompass non-computer

embodiments, such as a purely mechanical device whose physical structure has been designed

specifically to perform the algorithm? Such questions of claim scope should be answered in

particular cases by reference to the rule that the scope of a patent claim should be commensurate

with the scope of enablement provided by the patent specification.[261] In some cases, the

specification may enable both computer-implemented embodiments and non-computer

implemented (e.g., purely mechanical) embodiments, while in other cases the specification may

enable only computer-implemented embodiments.

The disclosure of a particular algorithm, for example, typically enables programmers of

ordinary skill to implement the algorithm in a variety of programming languages to produce

either executable software or firmware.[262] Whether such a disclosure, however, would enable the

design, construction, and use of a special-purpose machine that performs the algorithm and

which does not have the architecture of a general-purpose computer would need to be decided

based on the facts of the particular case. Whether such a machine would infringe a claim

---

[261] *See, e.g., In re* Fisher, 427 F.2d 833, 839 (C.C.P.A. 1970) (35 U.S.C. § 112 ¶ 1 requires that "the scope of the claims must bear a reasonable correlation to the scope of enablement provided by the specification to persons of ordinary skill in the art."); *accord In re* Geerdes, 491 F.2d 1260 (C.C.P.A. 1974).
[262] *See supra* note 247.

directed to the algorithm solely in terms of its constituent steps (i.e., without reference to any particular physical structure for performing such steps) would therefore also need to be determined based on the facts of the particular case.[263]

2. Software Claim Scope Should be Limited by 35 U.S.C. § 112 ¶ 6

A corollary to the proposed rule that the scope of a computer program claim should be limited to the logical structure of the claimed program is that accused products and processes that perform the same function or achieve the same result as the claimed program should not fall within the literal scope of the claim if such products or processes do not embody the claimed logical structure. Consider, for example, a hypothetical program which performs only the following two steps: (1) receiving a first number; and (2) adding the first number to itself. Such a program would not literally infringe claim 1 above because it does not perform the recited step of "receiving a second number." Although the claimed program and the hypothetical program perform the same function and achieve the same result, they do so using different logical structures (in this case, using different process steps).

One might imagine that patent applicants would attempt to work around this limitation on software claim scope by drafting computer program claims more broadly, perhaps by claiming programs in terms of the functions they perform or the results they achieve, rather than in terms of their logical structures. The inventor of claim 1, for example, might rewrite the claim as "a computer program comprising obtaining a sum based on at least one input number." A more realistic example would be a programmer who devises an algorithm for sorting a list of numbers

---

[263] *See supra* note 234 and accompanying text for related discussion on this topic.

using a complex sequence of steps, and who claims the algorithm as "a computer program comprising sorting a list of numbers."[264]

I propose that the scope of such claims be limited by operation of 35 U.S.C. § 112 ¶ 6 to the scope of the logical structure disclosed in the specification for performing the claimed functions. After all, the purpose of § 112 ¶ 6 is to "cut back" on the facial breadth of functionally-worded claims and thereby prevent patentees from expanding the scope of their claims to encompass any device or process for performing the claimed function.[265]

Although the courts have applied 35 U.S.C. § 112 ¶ 6 almost exclusively to product claims, the plain language of the paragraph applies equally to process claims.[266] Just as product claim elements that are expressed as a "means" for performing a specified function are to be construed to cover the corresponding structure or material described in the specification, so too are process claim elements that are expressed as a "step" for performing a function to be construed to cover the corresponding acts described in the specification.[267] The limiting effect of 35 U.S.C. § 112 ¶ 6 is triggered in the context of a process claim element when the element is "recited as a step for performing a specified function without the recital of acts in support of the function."[268] In such a case, the literal scope of the claim is limited to the particular procedural acts recited in the specification for performing the function recited in the claim.

---

[264] Although such a claim would be unpatentable for several reasons, in the present discussion I focus only on the claim's use of functional language for purposes of simplicity.

[265] See supra note 205.

[266] For an excellent overview of the history and current state of the step-plus-function doctrine, as well as evidence for the applicability of 35 USC § 112 ¶ 6 to process claims, see Schepers, supra note 235.

[267] See O.I. Corp. v. Tekmar Co., Inc., 115 F.3d 1576, 1582-3 (Fed. Cir. 1997) (interpreting the single sentence of 35 U.S.C. § 112 ¶ 6 to correlate "means" with "structure" and "material" and to correlate "step" with "acts").

[268] Id. at 1583; Caterpillar Inc. v. Detroit Diesel Corp., 961 F. Supp. 1249, 1255 (N.D. Ind. 1996), aff'd, 194 F.3d 1336 (Fed. Cir. 1999) (unpublished) (35 USC § 112 ¶ 6 "applies to functional methods [sic] claims where the element at issue sets forth a step for reaching a particular result, but not the specific technique or procedure used to achieve the result.").

The courts have done little to develop standards for determining whether a particular process claim element constitutes a functional "step" (which triggers § 112 ¶ 6) or a procedural "act" (which does not trigger § 112 ¶ 6).[269] The problem of distinguishing functional "steps" from procedural "acts" is complicated by the fact that English verbs in their gerund form – such as "adding," "converting," "applying," "determining," and "comparing" – may refer either to functional "steps" or to procedural "acts," depending on the context in which they are used.[270] The problem, therefore, may not be solved simply by developing a catalog of words that signify functional "steps" and another catalog of words that signify procedural "acts." Nor does it suffice to define as "functional" those terms which merely recite a "result,"[271] particularly in the context of computer programs, in which it is common to use the result of one procedure as an act in a higher-level procedure. A single term may therefore refer either to a step or an act, depending on the perspective from which it is viewed. Furthermore, any attempt to draw a dividing line between claimed processes that are sufficiently "specific" or "narrow" to avoid invocation of § 112 ¶ 6 on one hand and "broad" or "abstract" algorithms or "program flows" that invoke § 112 ¶ 6 on the other hand are destined to fail due to the inherently arbitrary and

---

[269] *See* Schepers, *supra* note 235, at 1162 ("It is nothing less than remarkable that four and one-half decades have passed without the emergence of judicial guidance concerning how (or if) paragraph six applies to process or method claims.").

[270] *See, e.g.,* Seal-Flex, Inc. v. Athletic Track & Court Constr.uction, 172 F.3d 836, 849 (Fed. Cir. 1999) ("Both acts and functions are often stated using verbs ending in 'ing.' For instance, if the method claim element at issue in this case had merely recited the 'step of' 'spreading an adhesive tack coating,' it would not have been clear solely from this hypothetical claim language whether 'spreading' was a function or an act. In such circumstances, claim interpretation requires careful analysis of the limitation in the context of the overall claim and the specification."). Furthermore, the Federal Circuit has stated that "[i]f we were to construe every process claim containing steps described by an 'ing' verb, such as passing, heating, reacting, transferring, etc. into a step-plus-function limitation, we would be limiting process claims in a manner never intended by Congress." *O.I. Corp.*, 115 F.3d at 1583.

[271] The decision of the Court of Customs and Patent Appeals in *In re* Cohn may be interpreted to stand for the proposition "that only those process steps which are truly 'result-oriented' and adequately defined within the specification will be subject to the effects of paragraph six." Schepers, *supra* note 235, at 1155-56.

subjective nature of the task of drawing that line absent some objective reference point.[272]

Rather, a more nuanced approach is required.

I propose an approach based on the Federal Circuit's reasoning in *Greenberg v. Ethicon Endo-Surgery, Inc.*,[273] and as further elaborated in *Personalized Media Communications v. ITC.*[274] At issue in *Greenberg* was whether the claim term "detent mechanism" was a purely functional term for purposes of § 112 ¶ 6. The court found that "detent" is a device, like a filter, brake, clamp, screwdriver, or lock, that takes its name from the function it performs.[275] The court held that "the fact that a particular mechanism . . . is defined in functional terms is not sufficient" to trigger § 112 ¶ 6 so long as "the term, as the name for the structure, has a reasonably well understood meaning in the art."[276]

The Federal Circuit further held in *Personalized Media Communications* that the claim term "digital detector" did not invoke § 112 ¶ 6 because the term "detector" had a "well-known meaning to those of skill in the electrical arts connotative of structure."[277] The term "detector" connoted, for example, a rectifier or demodulator, both of which were structures well known in the art. Even though the term "detector" did not "specifically evoke a particular structure, it [did] convey to one knowledgeable in the art a variety of structures known as 'detectors.'"[278]

---

[272] Similar arguments are often made in attempts to draw a dividing line between statutory and non-statutory subject matter. *See, e.g.,* Whitmeyer, *supra* note 92, at 1103, 1138 (arguing that only "narrow algorithms" should be considered statutory subject matter, while "program flows" should not be considered statutory subject matter).
[273] 91 F.3d 1580 (Fed. Cir. 1996).
[274] 161 F.3d. 696 (Fed. Cir. 1998).
[275] *Greenberg*, 91 F.3d. at 1583.
[276] *See id.*
[277] Personalized Media Communications, 161 F.3d. at 704.-705.
[278] *Id.* at 705; *followed by* Katz v. AT&T Corp., 63 F. Supp. 2d 583, 592 (E.D. Pa. 1999) ("a structural term need not connote a precise physical structure to those of ordinary skill in the art to avoid a means-plus-function analysis, as long as it conveys a variety of structures that are referred to by that term.").

The standard that apparently emerges from *Greenberg* and *Personalized Media Communications* is that a claim term does not invoke § 112 ¶ 6 if the term has a reasonably well understood meaning to those of ordinary skill in the relevant art that connotes either a particular *physical* structure or a variety of *physical* structures. I propose that the limiting effect of § 112 ¶ 6 be implemented in the context of computer program claims using an analogous standard, whether such claims be written in product, process, or hybrid form.

In particular, I propose that § 112 ¶ 6 be invoked by a term in a computer program claim if the term does not have a reasonably well understood meaning to those of ordinary skill in the relevant art that connotes either a particular logical structure or a variety of logical structures. As a result, an element in a computer program claim that is expressed as a means or step for performing a function without reciting a logical structure for performing that function, and which does not connote a particular logical structure or variety of logical structures for performing that function from the point of view of those of ordinary skill in the art, would be construed to cover only the corresponding logical structures described in the specification and equivalents thereof. Failure to describe any such logical structure in the specification would invalidate the claim. This would simply take the extension of patent claims to computer programs to its logical (no pun intended) conclusion.

Some examples may help to clarify the intended application of this proposed standard. Consider, for example, a computer program claim which includes as an element a step of "sorting" a list of numbers. Various sorting algorithms, consisting of particular sequences of steps that may be implemented in software, are well-known to programmers of ordinary skill.

Each such algorithm is an example of a logical entity having a known logical structure. The "sorting" claim element, therefore, would not invoke § 112 ¶ 6.

As an example of a computer program claim element that would invoke § 112 ¶ 6, consider an encryption algorithm whose novelty and nonobviousness stems from its particular novel and nonobvious combination of existing procedural acts that enable it to perform encryption using a single encryption key with a degree of security that is as high as that achieved by dual-key encryption algorithms. A claim to such an algorithm that failed to recite this particular novel and nonobvious combination of acts, but rather which recited only the function performed by the algorithm (such as "encrypting a message") or the algorithm's beneficial result (such as "encrypting a message using a single encryption key with as high a degree of security as that achieved by dual-key encryption algorithms") would invoke § 112 ¶ 6.

Invocation of § 112 ¶ 6 is appropriate in such cases because the inventor has invented and taught the public, through the specification, how to implement a particular encryption algorithm consisting of a particular combination of procedural acts, not how to implement *any* encryption algorithm that performs the same function or achieves the same result. Therefore, the facial scope of the claim is broader than the scope of enablement. This is analogous to the problem raised by Morse's functionally-worded claim 8 to the telegraph.[279]

The standard proposed above would correctly indicate that § 112 ¶ 6 should be invoked to limit the scope of such a claim, because the function recited by the claim does not connote a particular logical structure from the point of view of those of ordinary skill in the art. In fact, the recited function defines precisely the novel feature of the invention, and therefore by definition

---

[279] *See supra* Section IV.C.

does not correspond to any well-known logical structure. Applying the step-plus-function

standard proposed above, such a claim would be limited in scope to the particular logical

structure – the particular encryption algorithm – recited in the specification. Although this is a

trivial example, the same standard could be applied more generally to individual claim elements

which are expressed in terms of a function to be performed rather than in terms of a particular

logical structure or class of logical structures.[280]

D. Benefits of the Proposed Solution

1. Allows Software to be Described and Claimed in the Language of Computer Science

Allowing software programs to be conceived of, described, and claimed in terms of their

logical structure would provide several benefits. It would allow software to be described in

claimed in the same terms in which computer scientists design, describe, and think about

software, making software patents easier for computer scientists – and perhaps even lawyers,

patent examiners, and judges – to understand.[281] Computer scientists typically think about

software in terms of its logical structure, not in terms of the physical structures that are used to

implement it. Requiring software to be described and claimed in physical terms is an artifice that

results from requirements which developed from the processes that were historically used to

---

[280] By "class of logical structures" I refer to logical structures having one or more shared structural properties, at least one example of which is known to those of ordinary skill in the art. In the context of a product claim, for example, a "chair leg" refers to a class of structures, at least one example of which is known to those of ordinary skill in the art, and therefore does not invoke § 112 ¶ 6. The claim term "chair leg" need not, however, be interpreted narrowly to any particular chair leg known to those of ordinary skill in the art at the time of the patent filing or issuance, but rather may be interpreted broadly to apply to any structure that qualifies as a "chair leg" because it has the structural properties that characterize the class of chair legs.

[281] *See* Durham, *supra* note 92, at 1528 ("[T]he patent claim should reflect the art of programming; it should reflect the substantive details that belong to the programming art and that enable the technological implementation of the non-technological plan.").

develop previous kinds of electromechanical components. Such requirements simply do not fit software.[282]

2. Re-Interprets the Physicality Requirement in Light of the Nature of Software

As described above,[283] formal physicality neither provides a good test for subject-matter patentability of software programs nor provides a good mechanism for determining the scope of software patent claims. Replacing the formal physicality requirement with a substantive physicality requirement would provide a basis for re-interpreting the rules of patentability and claim scope in light of the unique ways in which software is invented and understood. These re-interpreted rules are designed to perform the same function as the old rules – to provide patent protection for novel, nonobvious, and useful inventions within the technological arts – except that the re-interpreted rules apply to inventions that are designed and conceived solely in terms of their logical structure.

Eliminating the need to recite physical structure in software patents would not eliminate the important role that physical structure plays in limiting patentability and claim scope. Rather, the approach proposed herein remains true to the role played by physical structure by retaining the requirement that the specification enable one of ordinary skill in the art to implement the claimed software program in a physical embodiment. This requirement would limit patentability to software programs that the inventor has actually taught the public to make and use, thereby retaining a substantive physicality requirement that performs the same function the formal physicality requirement has played in the context of conventional electromechanical inventions.

---

[282] *See, e.g.,* Chiappetta, *supra* note 6, at 94 (requiring that software be described in terms of the hardware on which it is implemented forces inventors to mischaracterize software innovations, thereby inhibiting the ability to describe and protect "the true nature" of software inventions under the patent laws).

[283] *See supra* Section IV.E.2.

3. Retains the Benefits of Patent Law

Another benefit of the approach proposed herein is that by retaining all of the traditional patentability requirements and rules of claim scope, but applying them to logical rather than physical structure, the benefits of the rich analytical framework that patent law has developed over the centuries would be retained. This framework may be applied to logical structure in the manner described above because of the strength of the analogy between the process of designing physical structures and the process of designing logical structures.[284] As a result, computer programs would only be patentable if their logical structure satisfied all of the traditional requirements for patentability. Furthermore, the scope of computer program claims would be defined by their logical structure and therefore commensurate with the scope of enablement.

Furthermore, such an approach will be equally applicable to new technological fields, such as biocomputing, in which physical entities are designed and described in terms of their logical structure. Even traditional forms of electromechanical engineering, such as integrated circuit design, are beginning to use techniques that resemble computer programming. Electrical engineers may now design certain kinds of circuits in purely functional terms using hardware description languages (HDLs) such as Verilog and VHDL.[285] Such a description may be provided to a "compiler," which generates a description of the physical structure of a circuit that performs the specified functions. This description may be provided directly to a foundry to fabricate the physical circuit itself. In such a case, the electrical engineer has designed a physical

---

[284] See Section [284] *See Supra* part III.G.III.G.

[285] *See generally* http://www.verilog.com/; http://www.vhdl.org. Verilog became IEEE Standard 1364 in 1995. *See also* Goodman et al., *supra* note 3, at 357-360 (describing commercially-available tools that allow electrical engineers to design circuits in terms of the functions they perform).

circuit having a novel physical structure without designing the physical structure.[286] The

principles discussed herein are applicable in this and other contexts that are likely to become

increasingly common in the future.

> 4. Recognizes Dual Nature of Computer Programs as Both Product and Process

Benefits of adding a new category of "computer program" statutory subject matter include

eliminating the need to couch software claims in either product or process language and

eliminating the need to include duplicative claims in software patent applications which

essentially claim the same subject matter using both product and process claim formats.[287] Any

requirement that a computer program be characterized as either a product or process is artificial;

in the context of computer programs, product and process are flip sides of the same coin.

Furthermore, distinctions among program (or process) and data are illusory in the context of

computer programming.[288]

---

[286] Even the process of designing analog circuits is being automated. *See, e.g.*, John G. Spooner, *Shaking up the Art of Chip Design,* CNET News.com(Apr. 19, 2002), *available at* http://news.com.com/2100-1001-886899.html (Apr. 19, 2002) ("Bucking tradition, start-up Barcelona Design is introducing software that automates aspects of analog-circuit design – a process long considered an art form by some engineers."). *See also* Vinod Kathail et al., *PICO: Automatically Designing Custom Computers,* IEEE COMPUTER MAGAZINE, Sept. 2002, at 39-47 (describing the "program in, chip out" automated system for using techniques similar to computer programming to automate the design of optimized, application-specific computer systems).

[287] *See, e.g.,* John R. Thomas, *Of Text, Technique, and the Tangible: Drafting Patent Claims Around Patent Rules*, 17 J. MARSHALL J. COMPUTER & INFO. L. 219, 265 (1998) (suggesting that "[o]ne way out of this morass of confused jurisprudence would simply be to abolish the historical distinctions between artifact [product] and technique [process] in their entirety"). A patentee would still have the option, under my approach, of using the conventional product or process claim formats if desired in particular cases.

[288] *See, e.g.,* DAVIS, MARTIN, *The Universal* Computer, WW Norton & Co., New York 2000, *supra* note 74, at 164-165 ("Turing's universal machine showed that the distinctness of these three categories [(machine, program, and data)] is an illusion."). The final theoretical and technological breakthrough that enabled modern computers to be designed and constructed was the "stored program" concept, according to which both program instructions and data are stored in the same memory. *See* IFRAH, *supra* note 37, at 222-232.

5.   Recommendations made herein are consistent with adoption of other reforms

Many changes to patent law have been proposed over the years to accommodate software.[289] These include, for example, increasing the USPTO's competence to examine software patents,[290] shortening the term of software patents (to, perhaps, two to seven years),[291] creating a limited right to reverse-engineer patented software,[292] applying the doctrine of equivalents narrowly in software patent infringement cases,[293] requiring European-style central claiming for software inventions,[294] limiting patent protection to those software inventions that are actually embodied in commercial products,[295] eliminating the presumption of validity for Internet business model patents,[296] eliminating or modifying the examination process to enable software patents to be issued more quickly,[297] and establishing a compulsory licensing scheme for software patents.[298] The USPTO has already adopted some measures in response to criticisms of business method patents which also address at least some of the concerns about

---

[289] *See generally*, Tim O'Reilly, T*he Internet Patent Land Grab*, 43 COMMUNICATIONS OF THE ACM 29-31 (2000).
[290] *See, e.g.*, *Oracle Corporation Patent Policy*, *Patent & Trademark Office Software Patent Hearings* (Jan. 26-27, 1994)*, available at*
http://www.jameshuggins.com/n/tek1/software_patent_oracle.htmlpf.ai.mit.edu/Patents/testimony/statements/oracle.s tatement.html (recommending that the "[p]rior art capabilities of PTO records . . . be vastly improved to confirm effectively the novelty and non-obviousness of software . . . that is the subject of [patent] applications"); *Prepared Testimony and Statement for the Record of Jim Warren Before the Patent and Trademark Office*, *supra* note 36 (recommending that a nationally-accessible collection of software prior art be made available, that the USPTO exercise much greater due diligence with regard to software patents, and that a large public advisory body be created to assist the Patent Office with its duties).
[291] *See, e.g.*, Gleick, *supra* note 36 (quoting a recommendation that Internet-related patents should have a term not exceeding two years); *see Prepared Testimony and Statement for the Record of Jim Warren Before the Patent and Trademark Office*, *supra* note 36 (recommending that the term of patent protection for software be limited to two years); Paley, *supra* note 22, at 306; Randall Davis et al., *supra* note 7, at 28.
[292] *See, e.g.,* Cohen & Lemley, *supra* note 143, at 1; Paley, *supra* note 22, at 343-346.
[293] Cohen & Lemley, *supra* note 143.
[294] Paley, *supra* note 22, at 332.
[295] *See id.* at 334-5.
[296] Symposium: *Internet Business Model Patents: Obvious by Analogy*, 7 MICH. TELECOMM. & TECH. L. REV. 253, 278 (2001).
[297] *See, e.g., Oracle Corporation Patent Policy, intra supra* note 359 (recommending a six-month patent examination process); Stern, *supra* note 25, at 167183 (advocating for a petite patent system for algorithms with negligible examination).

software patents.[299] The approach proposed herein is consistent with the adoption of any, all, or none of these reforms. In particular, the conception of software proposed herein may be integrated with intellectual property systems that protect software in a wide variety of ways, including *sui generis* systems of protection.[300]

VI.   Conclusions

A. Software development is a "useful art" and software should receive IP protection

Computer science is one of the "useful arts" and the invention and public disclosure of novel software designs should be encouraged. Our difficulty fitting software within the mold of patent law stems in part from problems within the law itself and in part from our own attachment to outmoded ideas about what it means for something to be a machine:

---

[298] *See, e.g., Prepared Testimony and Statement for the Record of Jim Warren Before the Patent and Trademark Office*, *supra* note 36; Simson L. Garfinkel, *Patently Absurd*, WIRED (July 1994); available at http://www.wired.com/wired/archive/207/patents.html Paley, *supra* note 22, at 306.

[299] *See, e.g.*, The O'Reilly Network,. *Internet Society Panel on Business Method Patents*, (Oct. 23, 2000)), *available at* http://www.oreillynet.com/pub/a/policy/2000/10/23/isoc.html (explaining the USPTO's "Business Method Patent Initiative," which included: (1) improvements to patent examiners' prior art searching capabilities; (2) the "Second Look," according to which a primary level patent examiner takes a second look at business method patents before they issue; and (3) imposing a requirement that patent examiners use automated search tools to search for business method prior art.) (September 14, 2003). *See also Business Methods Patent Initiative: An Action Plan* (September 14, 2003), *available at* http://www.uspto.gov/web/offices/com/sol/actionplan.html (September 14, 2003).http://www.uspto.gov/web/offices/com/sol/actionplan.html. Furthermore, the American Inventors Protection Act, Pub. L. 106-113, modified U.S. patent law to require that most patent applications be published 18 months after they are filed (35 USC 154(d)), providing the public with earlier notice of the subject matter potentially covered by a patent and making it easier for the public to challenge the patentability of an invention claimed in a patent application. *See* 37 CFR 1.902–1.997.

[300] Those who have stressed *sui generis* protection for software have recommended the need for a clearly-defined conceptual model of software to serve as the basis for such a system. *See, e.g.,* Samuelson, *supra* note 9, at 1148 ("The first step in creating a sui generis statute for the protection of computer program innovations would be the development of an appropriate conceptual model about both the nature of the subject matter to be protected and the purposes to be served by a protection statute."). For discussions of potential *sui generis* systems of intellectual property protection for software, *see generally* Phillips, *supra* note 44, at 997; Griem, *supra* note 8, at 145; Paley, *supra* note 22, at 306; Symposium: *Toward a Third Intellectual Property Paradigm: Article: A Manifesto Concerning the Legal Protection of Computer Programs*, *supra* note 3, at 2308.

What a road has been traveled since the French mathematician
Gaspard Monge (1746-1818), teaching at the École Polytechnique
in 1808, gave this definition of machines that today is considered
classic to the case in point: '[Machines] are engines destined to
bend the forces of nature, of weight, wind, water, and horsepower
to the work that man desires" [citation omitted].[301]

Although executable software programs "bend the forces of nature," they do so largely

invisibly and in a way that most likely is beyond our understanding in direct physical terms.

Rather, we understand and explain software through the intermediaries of its logical structure, the

processes it carries out, and the non-physical functions it performs. Software programs,

nonetheless, are components of machines and intellectual property protection should be available

for software designs for the same reasons as it is available for conventional electromechanical

designs. The development of new and useful software requires ingenuity, skill acquired through

training and practice, and investment which should be rewarded to provide software developers

with sufficient incentive to continue developing software and to disclose such software to the

public.

B. Patent law provides a good vehicle for providing such protection

Any intellectual property regime for protecting software must begin with a clear

understanding of what software *is*, including a recognition that software is defined in part by the

process by which it is invented. Patent law is the form of existing protection that is most

amenable to modification to provide appropriate protection for software that performs useful

functions. Patent law already includes a variety of mechanisms which would be useful in any

intellectual property system that protects software. These include mechanisms for: (1)

determining whether a design is new, useful, and nonobvious; (2) distinguishing among

---

[301] IFRAH, *supra* note 37, at 227 (quoting Gaspard Monge, cited in L. Couffignal, LA CYBERNÉTIQUE (Presses Universitaires De France 1963).

embodiments of a design, technical descriptions of such embodiments, and descriptions of the novel and nonobvious features of the design; (3) providing a scope of protection that is tailored to the scope of invention in each case; and (4) notifying the public of the scope of protection in each case.

C. The Challenge: Retraining the Legal System

Even if the doctrinal recommendations proposed herein were adopted in full, implementation of such reforms would pose significant practical problems. Patent law, and the lawyers, judges, and USPTO employees who implement it, have developed sophisticated conceptual tools for analyzing and describing inventions in terms of their physical structure, building in large part on the work of scientists and engineers over the course of hundreds of years. Patent law's mechanisms for analyzing and describing inventions in terms of their logical structure are much less developed. Even though patent law has allowed process claims for well over a century, it is only as a result of the advent of software that patent law has been presented with an entire technological field in which inventions are conceived of and described in terms that lack any reference to the physical properties of the inventions.

Although computer scientists have developed an extremely powerful and nuanced set of analytical tools for designing software and a variety of languages for embodying and describing software programs solely in terms of their logical structure, such knowledge and skills have yet to be transferred to members of the legal profession or imported into legal doctrine. Whatever the cause, most patent examiners, lawyers, and judges have yet to make the mental leap from physical to logical structure.

Computer scientists need to help the law and legal professionals to make this leap. Such a

project would necessarily involve computer scientists educating lawyers, patent examiners, and

judges about the basic features of computer technology, the thought processes that computer

scientists use to develop software, the languages that computer scientists use to convey their

designs to computers and to other computer scientists, and the kinds of problem-solving skills

shared by computer scientists. Such education would help to refocus the law on the logical

entities that software developers invent and away from the mere presence or absence physical

claim limitations. Furthermore, a better understanding of the kinds of techniques that computer

scientists typically use to solve problems may enable patent examiners and judges to bring the

law of obviousness more into line with actual software development practice than is currently the

case.[302]

D. Final Words

Patent law's emphasis on physical structure made sense, and continues to make sense, in

the context of electromechanical devices that must be designed and described in terms of their

physical structure to enable the public to make and use them. Software programs, however, need

not be designed nor described in terms of their physical structure to enable the public to make

and use them. The traditional requirements for patentability (such as utility, novelty, and

nonobviousness) should therefore be applied to the claimed logical structure of software

programs.

The recommendations provide herein, if adopted, would result in intellectual property

protection for software that both is consistent with the unique process by which software is

---

[302] For excellent recommendations along these lines, see Jeffrey D. Ullman's article *Ordinary Skill in the Art*, based upon the 2000 Knuth-Prize Lecture.(Nov. 16, 2000, updated May 16, 2001) Jeffrey D. Ullman, *Ordinary Skill in the*

invented and commensurate with the scope of invention in each case. Such a result would bring

the intellectual property law's treatment of software fully in line with its traditional treatment of

conventional electromechanical devices and thereby provide a fitting way to launch intellectual

property law into the twenty-first century.

Art, available at http://www-db.stanford.edu/~ullman/pub/focs00.html (Nov. 16, 2000, updated May 16, 2001).