SILICON EPICS AND BINARY BARDS: DETERMINING THE PROPER SCOPE OF COPYRIGHT PROTECTION FOR COMPUTER PROGRAMS

Anthony L. Clapes*
Patrick Lynch**
Mark R. Steinberg**

CONTENTS

* Senior Corporate Counsel, International Business Machines Corporation, Armonk, N.Y.
** Partner, O'Melveny and Myers, Los Angeles, CA

Preface

One of the most significant contributions that a master in a field of law can make is to apply his or her expertise in guiding the law as smoothly as possible through the often rough waters of social change. For copyright law, the roughest waters in recent years have been those in which sea changes in behavior have been caused by the availability of new technology. Reprography, audio and videotaping, and computer software have each presented severe challenges to the copyright law by facilitating—indeed, in some cases encouraging—novel kinds of activity inconsistent with the exclusive rights of the copyright owner.

Mel Nimmer on many occasions applied his singular expertise to help steer the development of American copyright law through such troubled waters. By serving as Vice Chairman of the National Commission on New Technological Uses of Copyrighted Works ("CONTU"), a body appointed by President Ford to consider what changes, if any, should be made in the Copyright Act in consequence of the broad availability of these new technologies, he played an important role in shaping the 1980 amendments to the Act. By updating his powerful treatise not just with summaries and syntheses of cases dealing with the new technological uses, but with his own perspectives on how (and which) copyright principles ought to be applied to those uses, he spread understanding to the broadest possible audience. By thrusting the hard questions posed by the new technologies before his students, along with traditional copyright teachings, he equipped a cadre of young lawyers to deal with novel issues

without doing violence to a legal scheme that has served our society so well for so long.

Shortly before he died, Professor Nimmer had an opportunity to deal first-hand with what was then and is still one of the most challenging questions resulting from the application of copyright law to the new technologies: What is the proper scope of copyright protection for computer programs? The question arose a few years ago, once it became clear that computer programs were protected by copyright. At that point the focus of both scholarly and commercial debate shifted to the question whether the exclusive rights of the author of a computer program are limited to the mechanical duplication of the literal text of the program only, or whether they extend to close paraphrases, translations and other non-literal forms of copying. In 1984, Professor Nimmer was engaged by one of the parties to a dispute over the scope of copyright protection for computer programs. In the course of that engagement, he prepared a declaration under seal concerning the deliberations on that subject that underlay the CONTU Final Report. The Nimmer declaration is an important addition to the body of scholarly opinion on computer programs and copyright; important because of its content and also important because it clarifies certain doubts which he had expressed in his concurring opinion to the CONTU Final Report. With the agreement of the parties to that dispute, the authors have been able to make the substantive text of that declaration public. It appears as an Appendix to this Article. This Article attempts to build on the wisdom which Mel Nimmer provided in that document.

I. INTRODUCTION

> *If we forget the scientific culture, then the rest of western intellectuals have never tried, wanted, or been able to understand the industrial revolution, much less accept it. Intellectuals, in particular literary intellectuals, are natural Luddites.*[1]

> *. . . The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures. . . .*
>
> *Programming then is fun because it gratifies creative longings built deep within us and delights sensibilities we have in common with all men.*[2]

> *In my Treatise I have analyzed in some depth the standards for determining when a copyrighted work has been infringed through the duplication of its fundamental plot, structure and arrangement (the pattern test). In my opinion, CONTU fully expected that these traditional principles be applied to computer programs.*[3]

For two centuries the American law of copyright has been evolving into a broad scheme of protection for literary works; protection that, consistent with its constitutional basis, guarantees to the author the exclusive right to copy, adapt, and distribute his or her work of authorship.[4] The

1. C.P. SNOW, THE TWO CULTURES AND THE SCIENTIFIC REVOLUTION 23 (1959).

2. F. BROOKS, THE MYTHICAL MAN-MONTH 7–8 (1975).

3. M. Nimmer, Declaration Regarding the National Commission on New Technological Uses of Copyrighted Works (CONTU) Final Report pt. 25 (Nov. 15, 1984), (*see infra* Appendix) [hereinafter Nimmer Declaration].

4. In enumerating the powers vested in the federal government, the Constitution provides that "[t]he Congress shall have power . . . To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries." U.S. CONST. art. I, § 8, cl. 8.

This clause of the Constitution was adopted by the framers without debate. James Madison stated in Federalist Paper No. 43 that "the utility of this power will scarcely be questioned." 1 M. NIMMER, NIMMER ON COPYRIGHT § 1.01[A] (1986). *See also* Goldstein v. California, 412 U.S. 546, 555–56 (1973). The following brief history of copyright legislation and the associated expansion of copyright protection through 1972 is found in Chief Justice Burger's opinion in *Goldstein*:

> The first congressional copyright statute, passed in 1790, governed only maps, charts, and books. Act of May 31, 1790, ch. 15, 1 Stat. 124. In 1802, the Act was amended in order to grant protection to any person "who shall invent and design, engrave, etch or work . . . any historical or other print or prints . . . ." Act of Apr. 29, 1802, ch. 36, 2 Stat. 171. Protection was extended to musical compositions

notion of protection against copying, from its early manifestations in English law as a mechanism to prevent transcription by persons other than the copyright owner,[5] has matured into a proscription against the unauthorized taking either of the literal text of a copyrighted work or of the es-

when the copyright laws were revised in 1831. Act of Feb. 3, 1831, ch. 16, 4 Stat. 436. In 1865, at the time when Mathew Brady's pictures of the Civil War were attaining fame, photographs and photographic negatives were expressly added to the list of protected works. Act of Mar. 3, 1865, ch. 126, 13 Stat. 540. Again in 1870, the list was augmented to cover paintings, drawings, chromos, statuettes, statuary, and models or designs of fine art. Act of July 8, 1870, ch. 230, 16 Stat. 198.

In 1909, Congress agreed to a major consolidation and amendment of all federal copyright statutes. A list of 11 categories of protected works was provided . . . . The House Report on the proposed bill specifically noted that amendment was required because "the reproduction of various things which are the subject of copyright has enormously increased," and that the President has specifically recommended revision, among other reasons, because the prior laws "omit[ted] provision for many articles which, under modern reproductive processes, are entitled to protection." H. R. REP. No. 2222, 60th Cong., 2d Sess. 1 (quoting Samuel J. Elder and President Theodore Roosevelt). Since 1909, two additional amendments have been added. In 1912, the list of categories in § 5 was expanded specifically to include motion pictures. . . . Finally, in 1971, § 5 was amended to include "sound recordings." Congress was spurred to action by the growth of record piracy, which was, in turn, due partly to technological advances. *See Hearings on S. 646 and H.R. 6927 Before Subcomm. No. 3 of the House Comm. on the Judiciary*, 92d Cong., 1st Sess. 4–5, 11 n.5 (1971).

Subsequently, the 1909 Act was supplanted by a general revision known as the Copyright Act of 1976, which became effective on January 1, 1978. Among other improvements, Congress eliminated the distinction between common law and statutory copyright, provided authors with a termination provision not tied to the renewal term of copyright, and explicitly recognized the application of copyright to "technological advances not dreamed of in 1909." 1 M. NIMMER, *supra*, at v–vi. Specifically, Congress recognized that computer programs were included among the "original works of authorship" protected by the Act. *See* H. R. REP. No. 1476, 94th Cong., 2d Sess. 51, *reprinted in* 1976 U.S. CODE CONG. & ADMIN. NEWS 5659, 5664; S. REP. No. 473, 94th Cong., 1st Sess. 50–51 (1975). This recognition was reaffirmed when Congress amended the Act in 1980. Act of Dec. 12, 1980, Pub. L. No. 96-517, § 10(a), 94 Stat. 3028 (1980) (codified at 17 U.S.C. §§ 101, 117 (1982)).

5. *See, e.g.*, 8 Anne ch. 19, 1710 *cited in* A. LATMAN, HOWELL'S COPYRIGHT LAW 2 (1962). The Statute of Anne was the first statute specifically to recognize the rights of authors and was "the foundation of all subsequent legislation on the subject of copyright both here and abroad." A. LATMAN, *supra*, at 3 (citing H. RANSOM, THE FIRST COPYRIGHT STATUTE (1956)); *see also* Donaldsons v. Becket, 4 Burrows 2303, 98 Eng. Rep. 257 (K.B. 1774).

sence of its expression.[6] That it is wrong to publish without a license a closely paraphrased or substantially similar version of a copyrighted literary work is a verity woven tightly into the fabric of our society;[7] by international convention and comparable legislation that verity has worked its way into the social fabric of responsible nations around the globe.[8]

And yet, as we celebrate the bicentennial anniversary of the U.S. Constitution and its charter to Congress to "promote Science and the useful Arts," that tradition is being challenged as it applies to an important new class of literary work—computer programs. Because computer programs as a form of expression are not well-understood by the makers of law and policy, they are presently at risk of being relegated to the backwaters of copyright, to an inferior status at law in which the authors of this class of literary work would be accorded less than full protection against the taking of their original works of authorship.[9] Only the steady hand of the federal judiciary on the tiller has thus far saved the copyrights in these new kinds of writing from foundering on the shoals of ignorance and suspicion.[10]

6. *See* Atari Inc. v. North American, 672 F.2d 607 (7th Cir.), *cert. denied*, 103 S. Ct. 176 (1982); Sid & Marty Krofft Television v. McDonald's Corp., 562 F.2d 1157 (9th Cir. 1977); Donald v. Zack Meyer's T.V. Sales and Service, 426 F.2d 1027 (5th Cir.), *cert. denied*, 400 U.S. 992 (1970); Nikanov v. Simon & Schuster, Inc., 246 F.2d 501 (2d Cir. 1957); Nichols v. Universal Pictures Corp., 45 F.2d 119 (2d Cir. 1930); Nutt v. National Inst. Inc., 31 F.2d 236 (2d Cir. 1929).

7. *Atari*, 672 F.2d at 618; Universal Pictures Co. v. Harold Lloyd Corp., 162 F.2d 354, 360 (9th Cir. 1947); *Nichols*, 45 F.2d at 121; 3 M. NIMMER, *supra* note 4, at § 13.03[A][1].

8. 4 M. NIMMER, *supra* note 4, at § 1709. A list of countries with which the United States has copyright relations as a result of treaty or international compact appears *id.* at App. 20.

9. Exemplary of the assault on the notion of full copyright protection for software are the following policy-oriented commentaries: OFFICE OF TECHNOLOGY ASSESSMENT, U.S. CONGRESS, *Intellectual Property Rights in an Age of Electronics and Information*, 78–85 (1986) [hereinafter OTA STUDY]; Goldstein, *Infringement of Copyright in Computer Programs*, 47 U. PITT. L. REV. 1119 (1986) [hereinafter *Infringement of Copyright*]; Kost, *Whelan v. Jaslow: Back to the Rough Ground*, 5 COMPUTER L. REP. 145 (1986) [hereinafter *Rough Ground*]; Liebman, Katsch & Leitch, *Back to Basics: A Critique of the Emerging Judicial Analysis of the Outer Limits of Programming "Expression,"* COMPUTER LAW., Dec.1, 1985 [hereinafter *Back to Basics*].

10. *See* Whelan Assoc. v. Jaslow Dental Laboratory, Inc., 797 F.2d 1222 (3d Cir. 1986), *cert. denied*, 107 S. Ct. 877 (1987); M. Kramer Mfg. Co. v. Andrews, 783 F.2d 421 (4th Cir. 1986); Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240 (3d Cir. 1983), *cert. dismissed*, 464 U.S. 1033 (1984); Stern Elec. Inc. v. Kaufman, 669 F.2d 852 (2d Cir. 1982); SAS Inst., Inc. v. S & H Computer Sys-

Forty years ago the computer program did not exist. Today a multibillion dollar industry has materialized to provide customers with the high-tech poetry that serves them in so many ways. According to published reports, in the United States alone there were over 14,000 enterprises writing computer programs for commercial distribution in 1984.[11] Since computer programs are software intended to be run through a hardware "player" (the computer), the demand for software stimulates the demand for hardware—the sales of which amount to many tens of billions of dollars in the United States.[12] The computer industry as a whole, hardware and software taken together, is a principal industrial contributor to the American balance of trade.[13] This industry has also been identified as critical to domestic economic performance in other major industrial nations.[14]

Like other technology-based segments of a country's economy, the computer industry flourishes in an environment in which the intellectual creations that fuel technological competition are sufficiently secure from misappropriation to guarantee the creator an opportunity for return on invested effort.[15] In the case of computer programs, the principal intellectual property is the program itself, the author's written expression that intrinsically describes the complete intellectual content of the product—much as the score of a symphony discloses the complete intellectual content of the composer's creation. For a product with such attributes, and for which the act of marketing can constitute publication,[16] copyright is the principal source of

tems, Inc., 605 F. Supp. 816 (M.D. Tenn. 1985); E.F. Johnson Co. v. Uniden Corp., 623 F. Supp. 1485 (D. Minn. 1985); Apple Computer, Inc. v. Formula Int'l, Inc., 562 F. Supp. 775 (C.D. Cal. 1983), *aff'd*, 725 F.2d 521 (9th Cir. 1984).

11. U.S. DEPT. OF COMMERCE, *A Competitive Assessment of the U.S. Software Industry*, 5 Washington, D.C., U.S.G.P.O. (December 1984) [hereinafter *Competitive Assessment*].

12. W.H. DAVIDSON, THE AMAZING RACE 113 (1984) [hereinafter DAVIDSON].

13. NATIONAL RESEARCH COUNCIL, *The Competitive Status of the U.S. Electronics Industry* 29, 58 (1984) [hereinafter *Competitive Status*].

14. *See* DAVIDSON, *supra* note 12, at 97. *See also* U. WEIL, INFORMATION SYSTEMS IN THE 80'S 307 (1982) [hereinafter U. WEIL].

15. *Competitive Status*, *supra* note 13, at 57.

16. Indeed, those firms that act as distributors of the works of program authors are referred to as "software publishers." *See, e.g.*, T. HARRIS, THE LEGAL GUIDE TO COMPUTER SOFTWARE PROTECTION 143–48 (1984) [hereinafter HARRIS].

intellectual property protection.[17] Trade secrecy, and in appropriate cases patent law, provide supplemental safeguards,[18] but the copyright grant and its constellation of exclusive rights, together with the ability to license customers and others to share in a limited way in those rights, form the basis for the software industry as we know it today.

The vitality of the software industry could be imperiled by a drastic limitation of the scope of copyright protection available to authors of computer programs. Yet, largely on the basis of misapprehensions and ignorance, movements to do exactly that are gathering adherents in this country and abroad.[19] This Article attempts to cast light on the status of computer programs as literary works and explain why traditional rules of copyright law adapt very comfortably to these novel forms of expression. It is an exercise that several federal courts have undertaken recently. In illuminating what is evidently a murky technological twilight for many commentators and policymakers, we hope to allay the fears they harbor and to provide a base of understanding similar to that which those courts have already acquired.

17. Braunstein, Fischer, Ordover & Baumol, *Economics of Property Rights as Applied to Computer Software and Data Bases*, in TECHNOLOGY AND COPYRIGHT 237–38 (G. Bush & R. Dreyfuss eds. 1979); J. Kasdan, The Economics of Copyright with Application to Licensing 1–2 (paper presented at the Conference on the International Legal Protection of Computer Software, Stanford Law School, July 24–26, 1986).

18. *See* T. HARRIS, *supra* note 16, at 131, 133–39; *see also* Diamond v. Diehr, 450 U.S. 175 (1981); Diamond v. Bradley, 450 U.S. 381 (1981); F. NEITZKE, A SOFTWARE LAW PRIMER 25 (1984) [hereinafter NEITZKE].

19. *See* authorities cited *supra* note 9; *see also* the following papers presented at the Conference on the International Legal Protection of Computer Software, Stanford Law School, July 24–26, 1986: W.R. Cornish, Legal Protection of Computer Programs in the United Kingdom and Parts of the British Commonwealth; A. Dietz, Copyright Protection for Computer Programs: Trojan Horse or Stimulus for the Future Copyright System? (Germany); Z. Kitagawa, Legal Protection of Computer Programs— One Aspect of Technology Law in Japan. A recent addition to the literature of this movement is Karjala, *Copyright, Computer Software, and the New Protectionism*, 28 JURIMETRICS 33 (1987) [hereinafter *Protectionism*]. *Protectionism* is an instructive exercise in the political use of words. By persistently referring to programs as "technology" and to program authors as "software engineers," *see id.* at 36-41, and by analogizing programs to typewriter keyboards, bridges, and other physical objects not protected by copyright, *see id.* at 39, 43, 45-46, 61, *Protectionism* gives the impression that the subject under discussion is something other than a work of authorship.

II. The Nature of the Debate

A. *The Opposing Viewpoints*

Since the mid-1970s, the federal courts have been dealing energetically with copyright infringement suits involving computer software. It has become fashionable to speak of this litigation in two generations, or waves, with each wave characterized by a different level of fundamental challenge to the copyrights asserted by authors of computer programs.[20]

For the most part, the "first wave" cases involved mechanical or other forms of slavish copying. They examined the question whether computer programs, or at least the kinds of programs claimed to be infringed, are protectable by copyright at all. This question was premised on the argument that programs are "functional" or "utilitarian" and therefore not within the ambit of the Copyright Act.[21] These cases firmly established that, despite the facial differences perceived by some to exist between computer programs and literary works of a more traditional nature, programs are indeed protected by copyright.[22]

The "second wave" cases have posed the far more subtle question of what copyright protection means in the context of computer programs.[23] The question has several forms:

What kinds of "taking" constitute infringement of a copyright in a computer program?

What kind of evidence is necessary to prove such an infringement?

20. *See, e.g.*, Laurie, *The Copyrightability of Microcode: Is It Software or Hardware . . . or Both*, COMPUTER LAW 1 (Mar. 1985). Even before the second wave has crested, a third wave of software copyright cases is discernable on the horizon. *See infra* note 29.

21. *See* Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1251 (3d Cir. 1983), *cert. dismissed*, 464 U.S. 1033 (1984); Stern Elec., Inc. v. Kaufman, 669 F.2d 852, 856–57 (2d Cir. 1982); Apple Computer, Inc. v. Formula Int'l, Inc., 562 F. Supp. 775, 780 (C.D. Cal. 1983), *aff'd*, 725 F.2d 521 (9th Cir. 1984). *See also* Williams Elec., Inc. v. Artic Int'l, Inc., 685 F.2d 870, 874–75 (3d Cir. 1982).

22. *Franklin Computer Corp.*, 714 F.2d at 1249; *Stern Elec., Inc.*, 669 F.2d at 857; *Formula Int'l, Inc.*, 562 F. Supp. at 779–81; *Williams Elec., Inc.*, 685 F.2d at 874–75.

23. Whelan Assoc., Inc. v. Jaslow Dental Laboratory, Inc., 797 F.2d 1222 (3d Cir. 1986), *cert. denied*, 107 S. Ct. 877 (1987); E.F. Johnson Co. v. Uniden Corp., 623 F. Supp. 1485 (D. Minn. 1985); SAS Inst., Inc. v. S & H Computer Sys., Inc., 605 F. Supp. 816 (M.D. Tenn. 1985).

What is the boundary between legitimate competition and infringement of a computer program?

The "second wave" cases have involved nonliteral forms of copying, such as translation from one programming language to another or from one computer environment to another,[24] or duplication of the outline, structure and flow of a program.[25] In general, the federal courts have held that substantial nonliteral copying of this sort also constitutes infringement of a program author's copyright.[26] Nevertheless, two opposing camps have formed around these "second wave" issues, and the debate over the validity of the recent cases is now sharply drawn.

On one side are the commentators who favor a narrow scope of copyright protection for software, one that would allow a reasonably free taking from an original program.[27] A principal concern expressed by these commentators is that "broad" copyright protection would prevent the use of ideas and concepts embodied in a program, thereby conferring a monopoly of ideas to the first author in a given field.[28] Another concern is that traditional copyright principles, if applied to computer programs, would result in granting "patent-like" protection to program authors, making it impossible for others to offer products in substitution for original programs.[29] Not surprisingly, accused infringers in the

24. *See Johnson*, 623 F. Supp. at 1497; *SAS*, 605 F. Supp. at 821; *see also Whelan*, 797 F.2d at 1226.

25. *Whelan*, 797 F.2d at 1228–29.

26. *Id.* at 1224–25; *Johnson*, 623 F. Supp. at 1497; *SAS*, 605 F. Supp. at 829–30. *But see* Synercom Technology, Inc. v. University Computing Co., 462 F. Supp. 1003, 1014 (N.D. Tex. 1978) (writing a program that conforms to input formats in a copyrighted manual not an infringement of copyright in the manual).

27. *See supra* note 19.

28. OTA STUDY, *supra* note 9, at 83; *see also* M. Mangan, The Software Copyright Lawsuit—The Alleged Infringer's Case, (unpublished manuscript distributed at Jan. 16, 1987 BNA Copyright Program) [hereinafter Infringer's Case]. A more extreme version of this concern holds that—at least for a significant number of programs—copyright, while it protects only *expression*, not the *idea*, of a program, is a cause of the market's failure to function properly. Menell, *Tailoring Legal Protection for Computer Software*, 39 STAN. L. REV. 1329, 1372 (1987) [hereinafter *Tailoring*].

29. *Back to Basics*, *supra* note 9, at 5; *Rough Ground*, *supra* note 9, at 148. The argument that copyright protection against nonliteral copying will stifle proliferation of substitute products has reached its shrill apogee in the computer trade press in a flurry of articles that have imported a new term into the debate: "look and feel," an industry term that refers to the aspects of a program that a user sees when a computer is operating under the control of that program. *See, e.g.*,

"second wave" cases have tended to embrace these arguments without question or analysis. In some cases, their positions may reflect a conscious strategy to sell programs knowingly "cribbed" from someone else's original.[30] In other cases, the defendants may have been arguing more from the expediency of a particular situation than on the basis of a reasoned view of their economic self-interest.[31]

On the other side of the debate are commentators who argue that software should be accorded the same copyright protection available for other works of imagination and who oppose the imposition of artificial restrictions on such protection.[32] Aligned with these commentators, in addition to the plaintiffs in the "second wave" cases, are the authors of computer programs that have achieved substantial popularity in the marketplace.[33] As already mentioned, the federal judiciary has, within broadly defined limits, adopted the view espoused by this group.[34] Nimmer's writings over the years, his concurring opinion in the CONTU Report, and his emphatic declaration, published as the Appendix to this Article, indicate that Mel Nimmer also believed that this view was the correct one.

Machrone, *Taking the Stand: The Look-and-Feel Issue Examined*, PC MAGAZINE 255 (May 26, 1987); *The Software Inquisition*, PC WORLD 15 (May 1987) [hereinafter *Inquisition*]. In part, the "look-and-feel" controversy results from the fact that programmers are just now learning what copyright is and in the absence of that knowledge have been operating in a state of innocence that is aptly reflected in the comments of Dan Bricklin, co-author of the first spreadsheet program, *Visicalc*: "Most of the developers I talked to . . . would like to be free to borrow from others as they see fit, in terms of the user interface . . . . You never copy exactly; you always embellish because of your ego." *Inquisition*, *supra* at 26. The legal aspects of the substitution arguments and the look-and-feel debate are considered in Part III, *infra* notes 49–291 and accompanying text.

30. *See* Petition for Certiorari at 12–13, Jaslow Dental Laboratory, Inc. v. Whelan Assoc., Inc. (No. 86-675), *cert. denied*, 107 S. Ct. 877 (1987) [hereinafter *Jaslow* Petition]; *Back to Basics*, *supra* note 9, at 8 (quoting declaration of Arthur J. Levine in *Whelan*).

31. *See* E.F. Johnson Co. v. Uniden Corp., 623 F. Supp. 1485, 1503–04 (D. Minn. 1985).

32. *See, e.g.*, Davidson, *Protecting Computer Software: A Comprehensive Analysis*, 1983 ARIZ. ST. L.J. 611, 653; Note, *Copyright Protection of Computer Object Code*, 96 HARV. L. REV. 1723, 1733–35 (1983); *see also* Note, *Defining the Scope of Copyright Protection for Computer Software*, 38 STAN. L. REV. 497 (1986).

33. Brief Amicus Curiae, Association of Data Processing Service Organizations, Inc., Whelan Assoc., Inc. v. Jaslow Dental Laboratory, Inc., No. 85-1358 (filed October 23, 1986).

34. *See infra* notes 183–268 and accompanying text.

In the balance between these opposing camps lies the future of the international computer industry.

B. *The Stakes: The Business That Derives From the Art of Programming*

1. The Role of Software in the Computer Industry

Without software, there could be no computer industry. The hardware provides heat and a little light on its own, but no information. Software controls the operation of the hardware and adapts the hardware to the performance of a wide variety of tasks. The critical importance of software to the computer industry as a whole is illustrated by the experience of Japanese suppliers of computer systems, who are said to have failed to gain broad acceptance of their hardware products on the world market because of inadequate software.[35]

Measures of the amounts of revenue generated from the commercial distribution of computer programs are inexact because the population of software suppliers is so diverse and diffuse, but it appears that for 1985 the figure is on the order of $8 billion.[36] Software revenues have grown quite rapidly, at a rate exceeding approximately 30 percent per annum in recent years.[37] These figures demonstrate that the art of writing software has fostered as vital and valuable a market as the art of writing novels or plays or popular music; perhaps more valuable, in fact, because the social benefits and economic efficiencies deriving from high-quality computer programs, e.g., to the banking system or the air traffic control and airline reservation systems or the auto industry or the public schools, may exceed the cost of the programs in question by several orders of magnitude. Because software plays the pivotal role in expanding the demand for computer systems, realization of the social and economic benefits of electronic computers is dependent to a large measure on the means that society uses to encourage the production of those original works of authorship that adapt

35. *Competitive Status*, *supra* note 13, at 63. *See also* S. MCCLELLAN, THE COMING COMPUTER INDUSTRY SHAKEOUT 119–20 (1984); U. WEIL, *supra* note 14, at 311.

36. Software Industry Report, June 27, 1986, at 10. *See Competitive Assessment*, *supra* note 11, at 20.

37. Software Industry Report, June 27, 1986, at 10.

the computer to the conquest of ever more challenging tasks.

2. The Dynamics of the Programming Business

It is important to keep in mind, as we consider the role of copyright in this industrial rather than aesthetic context, that the products in question derive entirely from the creative energies of authors practicing a craft. The literary nature of computer programming dictates the characteristics of the software market under consideration here. Programming as a business is characterized by the following economic attributes:

1. no significant capital barriers to entry;
2. a high degree of reliance on skilled intellectual labor;
3. a high degree of variability in the skill level of people active in the profession; and
4. a shortage of people who know how to program at all, and a severe shortage of good programmers.[38]

Because the cost of entry into the business of writing programs is low, and because skilled individuals can have access to a large market with few inputs other than their own time and energy, one finds a large number of enterprises engaged in the business of programming. One also finds that programs written by a single author or a small group of authors can become extraordinarily successful,[39] and garage-shop operators can suddenly find themselves major forces in the industry.[40]

Another characteristic of software is that its inherent benefits may be realizable on many different brands of hardware. Programs written for one type of computer can be adapted or translated (or "ported," to use an industry term) to a different type of computer, particularly if they are written with this possibility in mind.[41] For the author, this "portability" can increase the demand for her programs, in the same way that translating a novel from its original language into another language can increase demand for copies

38. *Competitive Assessment*, *supra* note 11, at 7, 11, 27.

39. *See* R. LEVERING, M. KATZ & M. MOSKOWITZ, THE COMPUTER ENTREPRENEURS, 115–33, 145–53 (1984).

40. *Id.* at 155–163, 189–94, 205–12.

41. *See, e.g.*, B. MACLENNAN, PRINCIPLES OF PROGRAMMING LANGUAGES: DESIGN, EVALUATION, AND IMPLEMENTATION 157–59 (1983); J. SAMMET, PROGRAMMING LANGUAGES: HISTORY AND FUNDAMENTALS, 36–48 (1969).

of the novel. For the user, portability can provide additio flexibility in configuring computer installations and choo ing suppliers.

In short, the business of programming has much in common with other branches of the business of writing. The products of the programming business are the products of the human activity known as authorship. That such authorship takes place in the context of a programming language does not change its essential nature, as we will shortly demonstrate.[42] It does, however, add a dimension of competitive urgency to the author's motivation, a dimension that is important to understand.

3. Competition Through Innovation

The dynamics of competition in the computer industry are familiar to students of contemporary commerce. Since its inception in the 1950s, the computer industry has been characterized by competition on the basis of the price/performance and functional capability of suppliers' products.[43] This form of competition, sometimes called "innovational competition,"[44] has produced consumer benefit on a scale unequaled in American industry.[45]

The process of innovational competition requires that firms develop new products. Those developments produce features that give their products advantages in price/performance or function. Those advantages, in turn, attract customers. Competing firms must offer similar (or greater) advantages or risk losing customers. The development efforts of the competing firms in the industry cause the

42. *See infra* notes 49–172 and accompanying text.

43. "Price/performance," for purposes of this Article, can be thought of as a measure of a product's primary capability—e.g., processing speed, storage capacity, or print speed—divided by the product's price. "Functional capability" is simply the range of different uses a product offers.

44. *See* A. THOMPSON, ECONOMICS OF THE FIRM 464, 469–79 (1973).

45. F. FISHER, J. MCKIE & R. MANCKE, IBM AND THE U.S. DATA PROCESSING INDUSTRY 353–55 (1983); M. PHISTER, DATA PROCESSING TECHNOLOGY AND ECONOMICS 58–73 (2d ed. 1979). In 1952, it cost about $300 to do a million processor operations and consumed ten minutes' time; in 1980 the cost was $0.001 and the time was 0.1 seconds. Computation speed in 1980 was 1,000,000,000 times greater in 1980 than it was in 1950 and by the end of the 1980s is expected to be 1,000,000,000,000,000 times greater. In 1952, computer memory capacity was on the order of 40,000 characters. In 1975, the memory capacity was as much as 15,000,000 characters. OTA STUDY, *supra* note 9, at 4.

successive waves of innovation that, in turn, produce dramatic improvements in price/performance and functional capability. In the case of software, these development efforts consist largely of programmer time, whether put forward by large teams of programmers working for established companies or by individuals or small groups investing their "sweat equity" to produce software for which they feel there will be a demand.[46]

Reaction by competitors to instances of innovational competition has generally taken one of two forms beyond the temporal expedient of price-cutting:

— response by offering products based on independent, alternative designs, but offering similar or even greater improvements; or
— response by copying the design of the innovative product.[47]

Both forms of competitive reaction have been important stimuli to technological advance in the industry, but innovation is obviously a more important factor than imitation. Given that these reactions are predictable, there is a vital interest in preserving that which stimulates the innovator to innovate in the first place. The question is, what moves the innovator to expend her efforts to develop a new product?

The innovator calculates, either explicitly or implicitly, that, despite the competitive reactions, she will be able to earn an attractive return on her investment in development of the product in question. This return will be promising if:

— the product has adequate value to potential customers,
— and the innovator will be able to sell her product to enough customers to justify her risk and investment.

46. "In 1981, it appears that farming is capitalized more heavily than programming." C. JONES, PROGRAMMING PRODUCTIVITY: ISSUES FOR THE EIGHTIES 3 (1981). "It cannot be emphasized too strongly that bringing a software system into existence is not analogous to the production of a car or even a computer mainframe. It is rather the equivalent of the initial design and prototype construction of the car or the computer . . . ." J. BUCKLE, MANAGING SOFTWARE PROJECTS 4 (1977).

47. *See, e.g.* F. FISHER, J. MCKIE & R. MANCKE, *supra* note 45, at 143–49, 286–303, 409–48; *see also* C. FREEMAN, THE ECONOMICS OF INDUSTRIAL INNOVATION 179–82 (2d ed. 1982); A. THOMPSON, *supra* note 44, at 471. The ability lawfully to copy a product's design is of course constrained by intellectual and industrial property laws. Only the constraints imposed by copyright law are considered in this Article.

Because imitation is clearly foreseeable in all events, and can be technically effortless in the case of software, the innovator will find the economic calculus attractive only if she is assured sufficient "lead time" before competitive products begin displacing her product in the market.

4. Copyright as a Precondition to Competition Through Innovation

The existence of lead time is indispensable to competition through innovation. Lead time is assured by several mechanisms:

— keeping new developments secret before they are announced;
— keeping product details secret after announcement, and insofar as possible, even after shipment;
— patenting inventions; and
— copyrighting expressions.

Hardware products produce lead time because reverse-engineering hardware entails a substantial expenditure of time. Would-be imitators must first determine how the new product works and what manufacturing processes were used to produce it, then adapt or build the requisite manufacturing facilities, and then build and test their own products.

Software is different. There are typically no manufacturing processes to analyze, and no special factories to set up. Software is written and tested; it is then published, like books, records, or videotapes. It is possible to copy a computer program in seconds and readily reproduce that copy by the hundreds or thousands. It is more difficult, but nonetheless relatively easy, to adapt, translate, or "port" a program, and thereby appropriate much of the value inherent in the original author's creation. Software, by its nature, lends itself to quick and unexpected duplication and even translation. Thus, if a programmer's only lead time "is the time it takes a technician to copy or adapt his work," rarely, if ever, will the first author enjoy the exclusive use of his work long enough to justify the effort and risk of its creation.[48]

48. Braunstein, Fischer, Ordover & Baumol, *supra* note 17, at 237–38; *Tailoring*, *supra* note 28, at 1337. The fact that computer programs, like other literary works, require legislation to deter copying—i.e., that it is difficult to prevent those who don't pay for the product from taking the benefit of it—is said to be one of two factors that render software a "public good," similar to a garden or military

In this environment, what produces the lead time? The answer that has worked well with other forms of authored products is the copyright law and its proscription against copying, adaptation, translation, or preparation of derivative works by others. The proscription ensures that rivals must invest in authorship in order to compete, while guaranteeing that such rivals have the free use of ideas embodied in the original expression. Moreover, the scope of the copyright protection available to programmers will be a determining factor in the level of investment in software development, the degree of competition by innovation, and the resulting rate of technological progress in the computer industry.

III. Resolving the Controversy

In order to decide what scope of protection the copyright law properly affords to computer programs, it is necessary to understand what comprises expression in computer programs, and then to consider how the copyright law applies to this expression. Developing that understanding, at least on a general level, simply requires an examination of the way programmers express themselves in programs, and of the process of writing programs. It is to that examination that we next turn.

A. *How Programmers Express Themselves*

1. Works of Authorship vs. Results

There are two fundamental modes of thinking about what computer programs are. One mode represents the perception of the typical customer. A typical customer perceives computer programs as things that cause computers to act in a particular way. In a sense, this perspective is similar to the concertgoer's impression of a symphony. To the concertgoer, the symphony is the set of sounds which he hears,

defense (the other factor being that additional consumers do not deplete the supply of goods available to others). *Tailoring*, *supra* note 28, at 1337. While in a technical economic sense software may be likened to public gardens or military defense for some purposes, the analogy is not particularly instructive. The vigor of the American software industry (*see supra* text accompanying notes 35–47) belies the notion that the government needs to be *a* supplier—or *the* supplier—of software to industrial or individual customers or to subsidize the supply of software to such customers. That intellectual property requires legal protection against misappropriation does not mean that property "belongs in some sense to the public." *See infra* note 179.

an aesthetic experience. Yet the concertgoer knows that the symphony, as created by its composer, is in fact a sequence of instructions that must be processed by an instrumentality, the orchestra, to produce the "symphony" heard by the listener.

The other way of thinking about computer programs, or symphonies, is to consider them as writings. Programs, like other literary works, consist of lines of text composed of symbolic characters. That is the programmer's perception, for that is the way she or he writes, or reads, a program. Returning to our concert analogy, the programmer's view would be analogous to that of the composer.

The expression in a program may be discerned from either perspective, but more of it is discerned when programs are considered as writings.[49] In this Section, we invite the reader to experience the programmer's view of her craft. We will first examine whether the constraints on program authorship imposed by the computer itself are quite as limiting as some commentators think. Then we will have a quick course in the elements of programming style, after which we will consider what the experts have said about the range of programming expression. We conclude our excursion with an overview of the process of writing programs and a consideration of what that process suggests about the scope of programmers' expression.

49. There are, of course, programs (such as computer games and other highly interactive programs) in which a great degree of expression also appears in the audio-visual display which the program instructs the computer to present. This Article does not deal with questions involving expression in such audio-visual displays, as to which application of traditional copyright principles is more easily understood. Although there has been a degree of controversy in the computer industry recently concerning the liability of those who copy the "look and feel" of the audio-visual displays of screen-oriented computer programs, *see, e.g.*, Complaint, Lotus Development Corp. v. Mosaic Software, Inc., No. 87-0074k (D. Mass 1987); Complaint, Lotus Development Corp. v. Paperback Software Int'l, No. 87-0076K (D. Mass 1987) ; *see also* Sanger, *A Divisive Lotus "Clone War"*, N.Y. Times, Feb. 5, 1987, at D1, col. 3, the scope of protection afforded to such audio-visual displays is quite well articulated and would appear not to differ depending on whether the source of those displays is computer program instructions, motion picture film, videotape, or any other tangible medium of expression. *See* Broderbund Software, Inc. v. Unison World, Inc., 648 F. Supp. 1127 (N.D. Cal. 1986).

2. The Context in Which Programs Are Written

It is sometimes said that programs differ from other kinds of literary works because they are not intended to be read by people. That assertion is false. Programs written for commercial distribution are almost always written with the idea that they will be read by people. We are not talking about bedtime reading, of course (except perhaps for hard-core "byteniks"), but about the fact that for programs to be modified, enhanced, or corrected, they must be "human-readable."[50]

Eventually, of course, a program is reduced to a form that can be read by a computer. Even in "machine-readable" form, however, a program can be read and understood only by humans, although that form is the most difficult of all versions of a program for humans to read and use.[51]

For programs to be "machine-readable" as well as "human-readable," the programmer must adhere to special kinds of syntactic, semantic, and constructional conventions. These conventions affect the form of expression but, as we shall see, not the range of expression for computer programs.

a. *What Computers "Understand"*

In the present context, "computers" are no more than machines fabricated of steel, silicon, plastic, and other physical components. Conceptually, these machines are only arrays of elements capable of being in a "turned on" or a "turned off" state like so many light bulbs. They are brought to life by the requisite switches either automatically in response to a change in some other element or at the direction of some intelligent operator.

50. R. LINGER, H. MILLS & B. WITT, STRUCTURED PROGRAMMING: THEORY AND PRACTICE 147–48 (1979). *See also* J. SAMMET, *supra* note 41, at 14–17.

51. Ironically, most "narrow protectionists" readily concede that the strict machine-readable sequences of a program are protected by copyright—and are all that is protected—whereas the elements of the program that survive translation into a higher-level, more human-intelligible expression are said to be not protectable. In other words, copyright protection should, the critics say, be severely truncated because programs are not human-readable; yet, it is precisely the most human-readable elements of the program's expression that would be denied protection by the truncation they propose. The paradox signals the fallacy of the argument.

Such a machine cannot "understand" anything. It simply reacts in accordance with the principles of physics to the actions of operators on its various circuits. The "lights" turning "on" and "off" have significance only to human beings who have designed a code that can be expressed in sequences of on's and off's, a code in which known electrical reactions will correspond exactly to large numbers of elementary instructions.

In order for this machine to do anything useful, someone must work the switches. This could be done by hand, switch by switch; and in the earliest days of electronic computing, it was.[52] Over the years, however, people have invented ways to work switches by "remote control." One way, exemplified by the Jacquard loom of post-revolutionary France, is to use cards with holes in them. In the Jacquard loom, selected pegs would fall into holes in the inserted cards, thereby switching the loom's pattern. A much more sophisticated approach is to use electronic signals—"on's" and "off's;" *i.e.*, to instruct the hardware to turn the switches on or off according to a predetermined plan and, as it turns out, at electronic speed.

Use of this latter technique is the hardware underpinning of the art of programming and the basis of modern computer systems. The power of this technique is so great that no general purpose computer is made up of hardware alone. *All* computers consist of (1) hardware that stores and manipulates "on's" and "off's" representing information or data, (2) hardware that controls the critical switches of the computer in specified ways in response to a set of predefined patterns of "on's" and "off's" called the "instruction set" of the computer, and (3) elaborate sets of instructions—programs—that will cause the computer to perform operations that human beings can translate into intelligible and useful results.[53]

These three constituents of a computer or, more properly, a computer system, are organized functionally according to the following taxonomy:

i. *Processing*: The hardware device that "reads" the programs is the processor. The job of the processor is to act on

52. *See infra* notes 82–84 and accompanying text.

53. C. GEAR, INTRODUCTION TO COMPUTER SCIENCE 49–50 (1973); M. MANDL, FUNDAMENTALS OF ELECTRONIC COMPUTERS: DIGITAL AND ANALOG, 6–8 (1967).

data in accordance with the program's instructions.[54] Important elements of the processor are:

(i) a pointer to the address of the instruction next to be processed,
(ii) circuitry to interpret instructions,
(iii) storage locations, or registers, to hold the data to be acted on, and
(iv) circuitry to perform the arithmetic or logical operation on the data which is called for by the instruction.[55]

With modest exception, today's processor hardware can process, albeit very rapidly, only one elemental instruction at a time, such as adding two numbers or comparing two numbers.[56] It can perform only those elemental operations that are built into its circuitry.[57] For any given computer, the set of such operations corresponds to a series of electronic signals that we have already referred to as that computer's "instruction set." One rule of expression for computers, therefore, is that instructions must be expressed to the processor in terms of combinations of the elemental operations in its circuitry.[58]

Another rule of expression for programmers is that the processor can only read "on's" and "off's."[59] That limitation forced programmers in the early days of the industry to use a "binary" notation system, i.e., a system in which all characters are expressed as combinations of the symbols "O" and "1."[60] Binary numbers work very much like Arabic numbers, although they are much less compact. For example, the decimal number 37 is written in binary as 00100110. Binary notation may also be used to represent letters.

In many computers, binary information is separated into blocks—each of which can represent a single character in some other language, such as a letter of the alphabet or

54. THE MCGRAW-HILL COMPUTER HANDBOOK §§ 2–8 (H. Helms ed. 1983).

55. *Id.* at §§ 2–8, 6–16; *see also* A. RALSTON, ENCYCLOPEDIA OF COMPUTER SCIENCE AND ENGINEERING 102–06 (2nd ed. 1983).

56. THE MCGRAW-HILL COMPUTER HANDBOOK, *supra* note 54, §§ 6–16. The exception referenced in the text is the so-called "parallel processor," which is not yet an important commercial factor but which will, in time, add yet another new dimension of complexity and flexibility to the task of writing programs. *See* PARALLEL PROCESSING SYSTEMS (D. Evans ed. 1982).

57. THE MCGRAW-HILL COMPUTER HANDBOOK, *supra* note 54, §§ 6–16.

58. *Id.*

59. *Id.* at §§ 1-7.

60. A. GILLIE, BINARY ARITHMETIC AND BOOLEAN ALGEBRA 1–14 (1965).

the symbol for an Arabic numeral. Thus, many computers use eight "bits" (*b*inary dig*its*, i.e., zeros or ones) to represent characters. With eight bits, we can represent 256 characters, which can include the English alphabet in both upper and lower case, the numerals 0–9, and many other symbols as well. This eight-character block is called a "byte" of information, and is a common way of organizing binary information in computers.[61]

Binary arithmetic has much in common with formal logic.[62] The two-state format, 1 and 0, is analogous to the true-false format of formal logic. As a result, the binary system lends itself not just to the computation of mathematical solutions, but also to the parsing of logical problems. For example, a computer may be instructed to do the following: If it is true that a case mentions the word "copyright," and true that it mentions either "computer program" or "software," and true that it was decided after 1982, then the name of that case should be printed.

As already suggested, there are electronic circuit elements that function in exactly the same way as binary arithmetic.[63] That is, they are only capable of being in one of two states, on or off. One circuit that can be built from such elements is an "And" circuit. A simple "And" circuit has two input lines and one output line. In this circuit, current will flow out (signifying a binary 1) only if current is flowing in on both inputs. Another such circuit is an "Or" circuit, out of which current will flow only if current is flowing in on either input. From such elementary circuits, a simple adder may be constructed in which the inputs 1 and 0 result in the output 01, the inputs 1 and 1 result in the output 10 and the inputs 0 and 0 result in the output 00.

By interconnecting large numbers of these elementary circuits, powerful computers capable of performing millions of instructions per second are created.[64] By switching the connections between these circuits through software, the

61. M. WEIK, STANDARD DICTIONARY OF COMPUTERS AND INFORMATION PROCESSING 49 (2d ed. 1977).

62. S. ADELFIO & C. NOLAN, PRINCIPLES AND APPLICATIONS OF BOOLEAN ALGEBRA 51 (1964).

63. THE MCGRAW-HILL COMPUTER HANDBOOK, *supra* note 54, § 5.2; S. ADELFIO & C. NOLAN, *supra* note 62, at 206–15.

64. A. RALSTON, *supra* note 55, at 103, 1127–31.

range of problems that computers can be directed to solve is, in fact, infinite.

ii. *Storage*: Just as there are circuits that can perform logic operations, there are also circuits that function to store a bit of information (a 1 or a 0), to report on what is stored when queried, and to change the contents of storage on command.[65] Large numbers of such circuits can be created on a semiconductor chip. In 1981, the storage chips in widest manufacture could hold sixteen thousand bits of information.[66] Today, the million-bit or megabit chip, is widely available and the 16-megabit chip is being discussed in the trade press.[67]

For our purposes, storage in a processor can be conceptualized as a collection of mailboxes or cells, each of which can hold one bit of information.[68] That information may be data or it may be part of an instruction. Each cell has an associated address and the processor sends information to or retrieves information from a cell by referencing its address.

Processor storage is temporary storage for data or instructions immediately needed by the processor.[69] Long-term permanent storage is generally provided by disk drives, tape drives, or other devices that store the information magnetically or, to refer to an exciting new technology, optically as points formed on a surface by light from a laser.[70] These devices generally offer much larger storage capacity than processor storage but retrieve stored data at a much slower speed than the processor storage. One job of the program

65. THE MCGRAW-HILL COMPUTER HANDBOOK, *supra* note 54, § 7.3.

66. W.H. DAVIDSON, *supra* note 12, at 107.

67. *See 16 Meg—A Test Vehicle for Now*, ELECTRONIC NEWS, Mar. 2, 1987, at 36.

68. M. MANDL, *supra* note 53, at 164. In actual practice, the information in cells may only be retrievable in a string, *e.g.*, a byte, rather than bit-by-bit. A. RALSTON, *supra* note 55, at 944–45.

69. In other words, the data and instructions in processor storage may constantly be changing as the processing proceeds. *See* THE MCGRAW-HILL COMPUTER HANDBOOK, *supra* note 54, at 2–4.

Like all generalities, this one has its infirmities. For example, many computers contain some permanent storage in the form of read-only memories, in which certain programming steps are permanently or semi-permanently recorded. A. RALSTON, *supra* note 55, at 1264–65.

70. A. RALSTON, *supra* note 55, at 955–67. *See also* J. HECHT & D. TERISI, LASER, SUPERTOOL OF THE 1980S 207–08 (1982).

author is solving problems caused by such differences in speed.[71]

iii. *Input and Output*: The information processed by a computer system must originally come from somewhere else and ultimately must be delivered somewhere else.

The input information may be in one or more of several forms. Data, text, image, voice, and graphics are five common types of information "input" into a computer.[72] The component of the computer that performs the input task may also take any number of forms: card readers, tape, disk or diskette drives, punched paper tape readers, keyboards on video display terminals, optical or magnetic character readers, or speech recognition devices.[73] Increasingly, computers themselves are serving as input devices to other computers.[74] The results obtained from processing may be stored on tape or disk storage. They may also be printed out on a printer or displayed on a screen, spoken over the telephone by a voice response unit, punched as holes in cards or paper tape, graphed on a plotter, or sent directly from one computer to another.[75]

Each type of input and output device has its particular way of addressing, or being addressed by, the processor. The ability to communicate with such devices is generally provided to the processor by means of software. One of the tasks typically performed by the authors of operating system software is to facilitate the input and output of information between the processor and the profusion of input and output devices with their various operating requirements.[76]

iv. *Control*: Computer systems are composed of a multiplicity of elements. The interactions of these elements must be controlled. For the most part, controlling the ele-

71. *See* S. KURZBAN, T. HEINES & A. SAYERS, OPERATING SYSTEMS PRINCIPLES 93–107 (1975).

72. A. RALSTON, *supra* note 55, at 736–38.

73. *Id.* at 739–66.

74. *Id.* at 647 (front end processors). *See also* D. CHAROFAS, PERSONAL COMPUTERS AND DATA COMMUNICATIONS 299 (1986) ("The successful implementation of intelligent, multifunctional, interactive workstations is tantamount to their online connection to mainframe resources, particularly to large databases.").

75. A. RALSTON, *supra* note 55, at 739–66.

76. S. KURZBAN, T. HEINES & A. SAYERS, *supra* note 71, at 93–107.

ments of the computer is the task of software writers.[77] Their creative energies transform the hardware world of elementary operations on ones and zeros into a coordinated set of products that can store and retrieve vast libraries of information, transmit messages among thousands of terminals, and allow users at those thousands of terminals to share the resources of the same central processor at the same time. "Control," in this sense, denotes the imaginative elaboration of instructions that take as given a computer's instruction set and compose on that limited base a complex, finely-articulated environment that, because of the speed of processor circuitry, responds in almost miraculously varied ways to customer prompting.

v. *Communication*: Computer systems can communicate with one another and parts of computer systems may communicate with one another over telecommunications lines.[78] The personal computer behind a lawyer's desk can access case law data bases in a large computer hundreds of miles away; an international enterprise's computers around the globe can transfer information to one another via satellite.[79] The various teller terminals in a bank may communicate with a processor over telephone lines or other wiring systems, forming a "local area network" or "data communications network."[80] For such networks to succeed, there must be programs that instruct the computers to behave much like a post office or central telephone exchange.[81]

77. *See* M. MANDL, *supra* note 53, at 241 ("Thus, a program tells the machine *what* to do, *how* to do it, and *where* to find the data to be acted on during execution of the program.").

78. COMPUTER NETWORK ARCHITECTURES AND PROTOCOLS 3 (P. Green ed. 1982).

79. *See* West Publishing Co. v. Mead Data Cent., Inc., 616 F. Supp. 1571 (D. Minn. 1985), *cert. denied*, 107 S. Ct. 962 (1987); W. STALLINGS, DATA AND COMPUTER COMMUNICATIONS 299–319 (1985).

80. "Local area networks" are communications systems that interconnect devices installed proximate to one another (*i.e.*, a few kilometers or less apart). They may utilize privately-owned wiring systems. *See* COMPUTER NETWORK ARCHITECTURES AND PROTOCOLS *supra* note 78, at 148 ("Data communications networks" are communications systems that interconnect devices that are installed remote from one another (*i.e.*, up to thousands of miles apart). They usually utilize shared switched networks or broadcast networks, most prominently the public switched network.). *See* W. STALLINGS, *supra* note 79, at 189–92, 240–43.

81. *See* U. BLACK, DATA COMMUNICATIONS, NETWORKS, AND DISTRIBUTED PROCESSING 121–35 (1983).

b. *The Language(s) of Computers*

The foregoing discussion indicates that computer hardware imposes certain requirements on the way programs are written. Yet those requirements provide a view of programming at its most primitive level, a level like that of an Ur-language that is no longer employed in actual discourse. Rather, programmers have created languages more convenient to human expression, which computers then translate into elemental patterns of 1's and 0's. We will now take a brief look at the development of such languages during the history of computer usage.

One of the first programmable computers was the ENIAC, which was designed and built at the University of Pennsylvania in 1946.[82] It weighed 30 tons, contained 17,000 vacuum tubes, and occupied 1800 square feet.[83]

The ENIAC was designed to calculate trajectory tables for artillery shells but it was capable of being adapted to solving other problems as well. In order to adapt the ENIAC, the operators had to reset some of the 6,000 switches on the machine and replug some of the hundreds of cables, like telephone operators at an old-fashioned switchboard. In effect, they were rewiring the connections among the ENIAC's circuits. The technicians who did this work no doubt used written instructions from the machine's designers that indicated in human-readable diagrams or language which switches should be set and how cables should be plugged in order to cause the machine to produce the desired result.

Replugging the computer was obviously a time-consuming, error-prone process, and if it had not been mechanized, there would be no commercial computer industry today. Fortunately, one of the people familiar with the ENIAC pro-

82. The development of the ENIAC is described in S. AUGARTEN, BIT BY BIT: AN ILLUSTRATED HISTORY OF COMPUTERS AND THEIR INVENTORS 107–31 (1984). A certain amount of controversy attends the history of this pivotal development project and that of its successor, the EDVAC, with substantial discord among the participants as to who was responsible for what aspects of the development. Three personal views of what transpired are available in H. GOLDSTINE, THE COMPUTER FROM PASCAL TO VON NEUMANN 149–84 (1972); H. LUKOFF, FROM DITS TO BITS: A PERSONAL HISTORY OF THE ELECTRONIC COMPUTER 29–41 (1979); and N. STERN, FROM ENIAC TO UNIVAC 7–86 (1981). The textual description of the ENIAC computer and the central concept of the stored program to which the difficulties of programming the ENIAC gave rise is taken from S. AUGARTEN, *supra*, at 121–42 and N. STERN, *supra*, at 50–51.

83. Today's hand-held programmable calculators are more powerful.

ject was the great mathematician John Von Neumann. Von Neumann championed the idea that the instructions for replugging the ENIAC were a kind of information in themselves and that they could be stored in the computer just like the data on which the computer operated. By running the instructions, which we now call a "program," through the computer along with the data and thereby indicating to the computer through which of its circuits the data should flow and in what sequence, the task of rewiring the computer could be eliminated and the computer would become a truly general-purpose, problem-solving tool.

This concept, the "stored program," is the key to the industry and to understanding the computer. By storing—for example, on disk or tape—a vast variety of programs and calling them into the processor's storage as needed, the same computer can be made to process a payroll, look up a customer history, book an airline reservation, send an item of electronic mail, play chess, simulate a hurricane, or keep track of a manufacturer's inventory.

The instructions that comprise the program must be written in a language that the processor reads. Strictly speaking, processors read only their own "machine language." ("Read" in this context means "can execute.") The "instruction set" of a computer is a specific set of strings of binary numbers that have particular "meaning" to the processor; when they are presented as input to the processor, they cause the processor's circuitry to perform specified actions.[84]

Machine-language programs, then, are programs expressed as strings of 1's and 0's. Each machine-language program consists of sequences of instructions. The instructions, generally speaking, take the form of (1) a command to the processor to perform some operation, (2) the identification of the parameters on which the operation is to be performed, and (3) the locations in processor storage where certain actions are to take place.[85] One such instruction,

84. M. WEIK, *supra* note 61, at 80, 296, 315 (defining "instruction set" as the set of operators of the "instruction repertory" or "operating codes" which the computer is capable of executing).

85. M. BOHL, INFORMATION PROCESSING 192–93 (4th ed. 1984). The discussion in the following text paragraph of the evolution from machine languages to more natural programming languages is taken from *id.* at 365–68.

translated from its binary form, might be "move parameter A to memory location 1103."

Machine-language programs contain large numbers of such instructions all written as sequences of ones and zeros. In the very early days of the industry, almost everyone who wrote computer programs wrote them directly in machine language because it was the only way to instruct the machine. This was a long, tedious, error-prone process. Because of the problems of writing in machine language, people began to write programs that would make the job of writing other programs easier, by serving as translators between symbols that humans could more readily understand and machine language. These translation programs perform tasks somewhat akin to those performed by translators at the United Nations. Programmers write their programs in languages other than machine language, languages that are more natural for them to write in, and the translation programs translate into machine language the programs so written. The translations are not necessarily simultaneous; except in the case of programs called "interpreters," translation programs generally prepare complete translations of other programs and then present the completed translations to the processor for processing. The languages that these translation programs accommodate may be categorized as follows:

Assembler language is simply machine language written in symbols more comprehensible to humans than binary notation.[86] In other words, the commands are the same as in machine language, but they are expressed as alphabetic characters or recognizable words (such as ADD) instead of as 1's and 0's. Parameters may be expressed as decimal or hexadecimal numbers instead of as binary numbers.[87] Locations are given names as well and those names can be chosen to provide some idea of what is stored in each location, e.g., "BALANCE" or "SUM" or "TOTAL."

A further aid to the readability of assembler-language programs is the ability of the programmer to write com-

86. *Id.*

87. A hexadecimal number is a number expressed in base-16 notation rather than base-10 (decimal) notation. Any group of four binary numbers may be represented as one hexadecimal number. *Id.* at 77–79. Hexadecimal 11, for example, is the same as decimal 17. Hexadecimal 1A is the same as decimal 26.

ments as to the reason for each instruction.[88] The comments are written for purposes of later reference by the author or any other reader; they are not processed by the computer during execution of the program.[89]

Assembler language cannot be read directly by the computer. A program called an "assembler" translates programs written in assembler language into the 1's and 0's of machine language.[90]

Both machine language and assembler language are based on the way computers are designed, rather than the way people think. They specify each elementary action that the computer must take, a very tedious thing to do.[91] Although a significant improvement over writing in machine language, assembler-language programming is still burdensome because, generally speaking, each instruction that the processor is to perform has to be written by the programmer. However, in the 1950s people began to realize that programs could be written that would translate single, high-level instructions into strings of assembler or machine-language instructions.

High-level languages are based on the notion that the job of translating a program written the way people think (high-level) into a program written the way computers are designed (low-level) could be mechanized if the high-level language was sufficiently formalized.[92]

The first formalized, high-level language available commercially was FORTRAN (FORmula TRANslator), created at IBM in the late 1950s. Since then, many others have been created, with names like COBOL, BASIC, PASCAL, and LISP.[93]

Just as assembler language programs are translated into machine language by "assemblers," high-level language programs are translated into assembler or machine language by

88. THE MCGRAW-HILL COMPUTER HANDBOOK, *supra* note 54, at § 11.3.

89. S. ALAGIC & M. ACBIB, THE DESIGN OF WELL-STRUCTURED AND CORRECT PROGRAMS 11, 17, 254 (1978).

90. M. BOHL, *supra* note 85, at 368.

91. *Id.* at 365–68. *See also* T. PRATT, PROGRAMMING LANGUAGES: DESIGN AND IMPLEMENTATION 21 (2d ed. 1984).

92. M. BOHL, *supra* note 85, at 370–71.

93. *See* HISTORY OF PROGRAMMING LANGUAGES CONFERENCE 25–41 (R. Wexelblatt ed. 1981).

programs called "compilers."[94] By expanding a single, high-level instruction into several "low-level" instructions, compilers remove from the programmer the burden of knowing how a specific processor actually works (*i.e.*, what its "instruction set" is).[95] As a result, the programmer can express the program in a language that is both more economical of her time and more comprehensible to a human reader. Additionally, the existence of compilers for a popular language like COBOL allows the same programs to run on processors with very different instruction sets. In other words, compilers written for each processor can translate the same high-level language program into different machine languages.[96]

In high-level languages, the command-parameter-location format of instructions can be substantially softened. The direct imperatives of low-level languages (*e.g.*, "Add A, B," "Move X, Y") are replaced by more refined directions (*e.g.*, "If A exceeds 1000, then print 'continue;' else print 'stop' "); or by descriptions of relationships among "objects" and query verbs that allow inferences to be drawn from those relationships; or by graphic icons and screen pointers.[97]

The point here is twofold. First, computers can be programmed in languages that are increasingly "natural" for people to use. Therefore, any argument that programs should be treated differently from other literary works because programming languages are different from human languages proceeds from a fundamentally erroneous premise. Second, there is no linguistic basis for distinguishing between the protection to be afforded programs expressed in high-level languages and that to be afforded programs expressed in machine or assembler languages, since a machine language or assembler language version of a high-level language program is nothing but a translation (in many cases, a mechanical translation) of the expression from one language into another.

94. T. PRATT, *supra* note 91, at 21. *See also* M. BOHL, *supra* note 85, at 372–74.

95. J. SAMMET, *supra* note 41, at 9.

96. *Id.* at 9–10, 36–43.

97. A concise comparison of solutions to the same problem written in different high-level programming languages is provided in Tesler, *Programming Languages*, SCI. AM., Sep. 1984, at 70–78.

c. The Categories of Programming Works

For purposes of this Article, we can think of two broad categories of programs:

Application programs are programs that customize the computer so that it can solve a particular problem for a customer.[98] Examples are word processing, airline reservations, or income tax computations.

Operating systems are programs that enable the other programs to run.[99] They act as a buffer between the application program and the hardware.[100] They also coordinate the many things that are happening in the computer system.[101]

B. *Elements of Programming Style*

A program is a literary work. It is a particular kind of literary work, to be sure, as is a musical composition or a "shooting script" for a movie, but one that has attributes in common with more familiar kinds of literary works. Those attributes are structure, flow, logic, design, naming conventions, commentary, and resultant style.

1. Structure of a Program

Programs have structure. They are organized into recognizable sections and subsections. Programs of any size have both a "coarse" structure and a "fine" structure. The coarse structure is generally called modularity.

a. Modularity

Large programs are usually organized into sections that are called modules. Typically, each module will be devoted to one of the major capabilities of the program.[102] Modules

98. A. RALSTON, *supra* note 55, at 92.

99. *Id.* at 1053.

100. *See* D. BARRON, COMPUTER OPERATING SYSTEMS 5–6 (1971); M. BOHL, *supra* note 85, at 401–02. A brief but comprehensive history of operating systems evolution appears *id.* at 404–31.

101. *See* A. RALSTON, *supra* note 55, at 1055–58.

102. *Id.* at 996. *See* D. VAN TASSEL, PROGRAMMING STYLE, DESIGN, EFFICIENCY, DEBUGGING AND TESTING 67–73 (1978). It seems to be a commonly accepted tenet of good programming style that modules be self-contained in terms of purpose and accessible to one another through a minimum number of connections. *See, e.g.*, J. ARON, THE PROGRAM DEVELOPMENT PROCESS 99 (1974); A. RALSTON, *supra* note 55, at 996; D. VAN TASSEL, *supra*, at 68–69. However, the design of a program may be divided into modules in a number of different ways. The division

are roughly equivalent to chapters in a book. A module may be broken down into recognizable submodules,[103] just as a chapter may have subchapters or paragraphs.

b. Blocks of Code

The fine or "detailed" structure of a program includes two types of structural elements: blocks of code and data areas.

Blocks of code may be linear or reusable. A linear block of code is simply a series of contiguous lines of programming text. A reusable block of code is a series of contiguous lines of programming text that is susceptible of incorporation by reference where desired when writing a program.[104] Reusing the same set of instructions at different places within the flow of a program is a fundamental technique of program authorship.[105]

Programmers have developed various approaches for "marking" segments of code for reuse, for "pointing" the program to the location of a desired segment, and for returning the program to the next proper instruction once the referenced code segment has been executed.[106] The principal distinctions among the various approaches used for marking and using reusable segments of program text lie in the way the computer is told to identify the reusable unit in question. Some of the principal types of reusable units of code are "macros," "routines," "subroutines," and "entry points" within routines.

The term "macro" (short for macro-instruction) refers to a reusable block of code that can be incorporated by reference using a convenient procedure provided in the assembler program.[107] This procedure permits a desired block of reusable code to be assigned a symbolic name (*e.g.*, COM-

may be based on the *functions* to be provided by the program, on the *sequence* of the program's contemplated execution, on the anticipated *relationships* among parts of the program, or on some combination of the foregoing. *See* J. ARON, *supra*, at 99–100.

103. J. ARON, *supra* note 102, at 100–01.

104. *See* P. SHERMAN, TECHNIQUES IN COMPUTER PROGRAMMING 100–02 (1970).

105. *Id.*

106. *Compare* P. SHERMAN, *supra* note 104, *with* J. ULLMAN, FUNDAMENTAL CONCEPTS OF PROGRAMMING SYSTEMS 133–40, 152–53 (1976).

107. J. ULLMAN, *supra* note 106, at 133–34. Macros are commonly associated with assembly language programs. *Id.*

POUNDINT). The assembler will automatically be able to locate that sequence of instructions when another program refers to it by its symbolic name. Thereafter, a programmer need only specify the appropriate macro name. The assembler or compiler will include the instructions from the macro routine of that name in the programmer's program as if they had been written out in sequence at that place in the flow of the program.

Generally, a macro will be set up to allow the using programmer to customize it by "inserting" one or more specific values.[108] In a sense, such a macro is like a standard letter that is customized by filling in the blanks. The "blanks" that allow these insertions are the "formal parameters" of the macro and the values inserted by the using programmer are called "actual parameters."[109] Definition of the formal parameters that a particular macro will support is an important part of the design process and is reflective of the creativity and style of the particular programmer or group of programmers who write a given program.[110]

Macros have many uses in computer programming. Most importantly, they allow programmers to avoid writing detailed series of instructions over and over.[111] They can also be used to simplify the connection between two programs, *e.g.*, an operating system and an applications program.[112] Communication between two programs must be effected more carefully than communication between two people. If one person asks another for her address and phone number and the other person gives her phone number first and then her address, her response will still have been understood. Computer programs do not generally have that sort of flexibility. One of the purposes for writing macros can be to save a program author from having to remember cumbersome rules for program-to-program communication that have to be embodied in the program. (A telephonic analogy to calling a macro for this purpose would be pushing the speed-dial button on a programmable phone.)

108. *Id.*
109. *Id.*
110. *See* E. YOURDON & L. CONSTANTINE, STRUCTURED DESIGN 285–86 (1976).
111. A. RALSTON, *supra* note 55, at 904.
112. *See, e.g.*, A. SIMPSON, UNDERSTANDING DBASE III 170–73 (1985).

Definition of a macro is but one of the ways to mark a reusable block of code. The procedure described above is, however, for our purposes adequately descriptive of the alternative methods of "blocking" sets of instructions for reference or reuse. These alternative methods are:

Nonreturning Routine, a self-contained block of code or set of instructions that may be "callable" by reference to a symbolic label but which has no provision within it for returning control to the "calling" part of the program.[113]

Subroutine, a block of code that can be "called" from elsewhere in the program and that will return control to the next instruction following the calling instruction (or to another specified part of the program).[114] From the programmer's standpoint, "calling" a subroutine only requires the use of a symbolic name; the assembler does the rest.

Entry point, a label to which the program can branch if the calling programmer wants to perform some, but not all, of a routine.[115]

c. *Data Structures*

We now turn from the notion of reusable blocks of code to the other major distinguishing feature of a program's detailed structure: the way the program organizes and refers to the data that it uses. The facilities for accomplishing this are known as the program's *data areas* or *data structures*.

Among the features provided by almost all programming languages is the ability to refer to an item of data by assigning it a name or identifier. Some of the quantities so named may be constants, which have the same value throughout. For example, *pi* might be assigned the value 3.14159. Other named quantities are variables, which can be assigned new values by statements within the program, so that their values cannot be known until the program is run. The variables DIAMETER and CIRCUMFERENCE, for example, might take on new values each time a calculation is done.

113. *See* M. WEIK, *supra* note 61, at 302.
114. P. SHERMAN, *supra* note 104, at 110–15.
115. *See* M. WEIK, *supra* note 61, at 261.

A data structure is an organized collection of data.[116] Volumes have been written on the subject of data structures for computer programs.[117] For economy of exposition, we will treat here only the data structures known as "control blocks," which are data structures used for control purposes by many types of programs. The following discussion may be generalized to other types of data structures, however.

A control block is an area of memory reserved in accordance with instructions from a program and used as a "file card" or "notepad" to store information that the program will need from time to time to carry out given tasks.[118] A loose analogy might be a set of reference tables for the identities of characters in a lengthy novel. A control block is made up of a series of fields in which discrete items of information are to be filed.[119] The fields, and sometimes even individual bits within the fields, are assigned symbolic names.[120] The control block is defined by the variables that it includes, the order of the variables, and their respective sizes.[121]

The definition of individual control blocks, the selection of information to be contained in those control blocks, and the reservation of fields of certain sizes to hold presently-needed information or information that may be needed in the future are matters within the discretion of the program

116. *See* A. RALSTON, *supra* note 55, at 497–501. Commonly occurring types of data structures are arrays (indexed groupings of data), sets (groupings of data in which the order within the set is irrelevant), lists (sequentially organized and accessible groupings of data), trees (lists in which one entry is the root and all the rest have unique predecessors), stacks (lists in which entries are created or deleted on a last-in-first-out basis) and queues (lists in which entries are created or deleted on a first-in-first-out basis). *Id.*

117. *See, e.g.*, A. BERZTISS, DATA STRUCTURES (2d ed. 1975); R. ELLZEY, DATA STRUCTURES FOR COMPUTER INFORMATION SYSTEMS (1982); I. FLORES, DATA STRUCTURE AND MANAGEMENT (1970); C. GOTLIEB & L. GOTLIEB, DATA TYPES AND STRUCTURES (1978).

118. D. HSIAO, SYSTEMS PROGRAMMING: CONCEPTS OF OPERATING DATA BASE SYSTEMS 79–81, 218 (1975); M. WEIK, *supra* note 61, at 44.

119. *See, e.g.*, D. HSIAO, *supra* note 118, at 79–81.

120. *See, e.g.*, INTERNATIONAL BUSINESS MACHINES CORPORATION, SYSTEMS NETWORK ARCHITECTURE FORMAT AND PROTOCOL REFERENCE MANUAL: ARCHITECTURAL LOGIC App. A at A-23 (3d ed. 1980) [hereinafter SNA MANUAL]; A. LISTER, FUNDAMENTALS OF OPERATING SYSTEMS 95 (3d ed. 1984).

121. *See, e.g.*, H. LEVY & R. ECKHOUSE, COMPUTER PROGRAMMING AND ARCHITECTURE—THE VAX-77 192–93 (1980); H. LORIN & H. DEITEL, OPERATING SYSTEMS 172–78 (1981).

author.[122] The choices made by the program author can substantially affect the quality and efficiency of the resulting program.[123] More importantly, such choices are generally selected from a large field of alternatives based on a program author's training, experience, and writing style.[124]

2. Flow of a Program

Programs have *flow* as well as structure. The flow of a program is roughly akin to the flow of a plot in literature. It answers the question "what happens next?" There are two types of flows that can profitably be considered herein. One is the flow of control; the other is the flow of data.

The *flow of control* answers the question "what instruction is presented to the processor next?" Control of the processor rests in the program and is passed from instruction to instruction in various fashions.[125] The simplest fashion is *seriatim*. Unless the programmer decides differently, instructions will be presented to the processor in the order in which they appear in the author's text, known as the "source code."

As discussed above, however, the programmer can decide that control of the computer should be passed from one part of the program to a remote part rather than to a contiguous part. This can be done by calling a macro, for example, or branching to a subroutine. Programs can also loop through the same string of instructions a number of times before moving on.[126]

122. *See* D. VAN TASSEL, *supra* note 102, at 52.

123. H. LEVY & R. ECKHOUSE, *supra* note 121.

124. *Id. See also* B. SCHNEIDERMAN, SOFTWARE PSYCHOLOGY 161–71 (1980); G. WEINBERG, THE PSYCHOLOGY OF COMPUTER PROGRAMMING 29 (1971); E. YOURDON & L. CONSTANTINE, *supra* note 110, at 223–27.

125. M. HECHT, FLOW ANALYSIS OF COMPUTER PROGRAMS 4 (1977); M. MANDL, *supra* note 53, at 241; P. SHERMAN, *supra* note 104, at 48.

126. The purpose of looping is not simply to cause the computer to repeat the same instruction sequence with the same data, like a phonograph needle stuck in a single segment of the groove of a scratched record, but to repeat the same instruction sequence on data that differs with each pass through the loop. I. FLORES, *supra* note 117, at 139–71.

For example, the calculation of compound interest may be achieved by repeating the simple-interest computation for as many times as the interest is to be compounded, each time using as principal the result of the previous calculation.

Many types of literary works have a flow of control that is other than seriatim (*e.g.*, "If you already have an understanding of contract law, skip to chapter 3," or "For the recipe for Bernaise Sauce, turn back to page 42"). There are even inter-

The flow of control is dependent on two factors: the extent to which the program uses linear or reusable blocks of code and the extent to which parts of separate modules interact to accomplish a single purpose.

The *flow of data*, on the other hand, answers the question "where is what data sent and when?"[127] Items of data in computer programs are roughly analogous to characters in a novel. They have names or "labels," and each has a role to play in the program.[128] Some of these characters are constant and unchanging, while others are transformed by the flow of events in the computer as controlled by the program.

As already noted, items of data are stored in data areas; a common type of data area is the "control block," so called because it contains information relating to control of the system, rather than data required by applications programs. In the control block the data are stored in named fields.[129] An item of data flows (or, more accurately, is copied) out of the control block when its name is called and then is either compared to some other data or modified and replaced or otherwise acted upon.[130] One of the distinguishing features of a program is the pattern described by the movement of data.[131] This pattern consists of several elements: when an item of data is called, where it is called from, for what purpose it is called, where else in the program it is passed, if anywhere, and what its ultimate disposition is. A tracing of the flow of data in a program produces a road map to all of the activities of the "characters" in the literary work that the program represents.

active novels today, wherein the choices made by the reader will dictate what portion of the work should be read next. *See, e.g.*, E. PACKARD, SUGARCANE ISLAND (1976). Provision for non-seriatim playing of musical pieces has long been a feature of musical notation. Finally, non-seriatim flow is, of course, essential to the enjoyment of computerized interactive novels or adventure games. *See, e.g.*, C. CRAWFORD, BALANCE OF POWER (1985); ICOM SIMULATIONS, INC., DEJA VU (1985).

127. *See*, M. HECHT, *supra* note 125, at 4–5 (suggesting that analysis of a program's data flow can be similar to observing "the way gossip is propagated").

128. J. HARTLING, L. DRUFFEL & F. HILBING, INTRODUCTION TO COMPUTER PROGRAMMING: A PROBLEM-SOLVING APPROACH 38–44 (1983).

129. *See, e.g.*, SNA MANUAL, *supra* note 120, at 123–24.

130. *See* J. HARTLING, L. DRUFFEL & F. HILBING, *supra* note 128, at 38–39.

131. M. HECHT, *supra* note 125, at 14–15.

3. Logic of a Program

Another principal characteristic of programs is their logic. Logic[132] is of course a characteristic of all literary works. Works of persuasion are generally recognized as proceeding in a logical fashion, using various kinds of rigorous reasoning to impel the reader to follow and accept the author's argument. Fictional works, too, require logical development to maintain tension, keep the reader engaged, and sustain believability; the logical relationship among words, sentences, paragraphs, or chapters in a novel is an important element of a novel's expression.[133] Computer programs exhibit analogous logical relationships.

As in the case of a program's structure, there is a "coarse" logic and a "fine" or "detailed" logic in every program. The coarse logic is simply the set of major capabilities that the program provides and the relationship among those capabilities.[134] The detailed logic of a computer program

132. The term "logic" has many meanings and some of those meanings (*e.g.*, "principles of reasoning") describe levels of abstraction presumably referred to by those commentators, including the Copyright Office of the United States, who contend that program "logic" is not protected by copyright. *See* COPYRIGHT OFFICE, COMPENDIUM II: COMPENDIUM OF COPYRIGHT OFFICE PRACTICES, § 325.02(C) (1984). It will be clear from what follows that program logic includes much more than such high levels of abstraction, however. As Professor Nimmer reminds us:

> The crucial question is whether a particular word describes only an idea or whether it refers to the concrete expression of an idea in a program. Where such words as "logic," "flow," or "structure" refer to expression (as, e.g., under the pattern test as explained in my *Treatise* . . .) they refer to copyrightable subject matter.

Nimmer Declaration, *supra* note 3, at pt. 28 (*see infra* Appendix).

133. Perhaps the logical first step in the fictional process is the writer's conscious or intuitive recognition of the nature of narrative, and his acceptance of the shackles imposed by his decision to tell a story . . . . By definition—and of aesthetic necessity—a story contains *profluence*, a requirement best satisfied by a sequence of causally related events, a sequence that can end in only one of two ways: resolution, when no further event can take place . . ., or in logical exhaustion, our recognition that we've reached the stage of infinite repetition . . . .

J. GARDNER, THE ART OF FICTION 53 (1983). And again:

> As for fiction, . . . it seems fair to say that, since no narrative beyond a certain length can hold interest without some such profluence as a causal relation of events (by either real-world logic, comic mock-logic, or poetic logic), no narrative except a very short one can escape real-world relevance. . . . Fiction seeks out truth.

Id. at 79.

134. For example, the coarse logic of a payroll program might be described as follows:

comprises the sequences of individual steps that make up the program.[135] As in the case of different human languages, not all elements of expression exist in all programming languages,[136] but the following principal elements are relatively common.

One common element of logic expression is IF-THEN-ELSE: IF a certain condition exists, THEN take a prescribed action, ELSE take a prescribed alternative action. IF-THEN-ELSE is one way for the computer to test the status of data and take an action depending on the outcome of the test.

Programmers often use IF and THEN statements without ELSE statements in order to accomplish a conditional result without specifying an alternative action. ("If your shoes are muddy, then take them off before you come in.") IF will usually test a variable to see if it is equal to, less than, or greater than a specified value, as in "IF counter = 0." THEN may be followed by an action or it may be followed by one or more IF statements. (IF Johnson's birthday is March 29, THEN IF today's date is March 27 or before but not earlier than March 1, THEN send a birthday card. ELSE call.)

The actions that can follow a THEN are the same kinds of actions that may be taken without IF-THEN tests preceding them. They are other forms of logic expression, *e.g.*:

- Set Variable A equal to some value, either constant or variable, *e.g.*, VARA = R15 (where R15 is the contents of register 15).
- CALL a macro or subroutine.
- GO TO some other part of the program (in other words, transfer control to another part of the program with no return).
- DO a series of steps. The steps referenced by the DO command may be presented repeatedly if desired, using DO WHILE or DO UNTIL, where WHILE and UNTIL insert tests of the conditions under which the steps are to be done.

Taken individually, of course, no one of the described logic elements can be said to be highly expressive in itself. The combination and sequence of these logic elements, however, can represent creative expression. Out of these elementary logic expressions and many others, computer programs of great elegance and complexity can be written. The choice of logic elements, their pattern, sequence, and significance are as fundamental to programmers' expression as the choice of words, their sequence, and significance are to poets' expression. To make the sweeping statement, as some do, that "program logic" is not protectable by copyright, is to display a profound ignorance of the nature not just of programming languages but of languages in general.[137]

4. Design of a Program

Another attribute which programs have in common with other literary works is *design*. Design is the qualitative result of combining structural, flow, and logic elements in the particular fashion chosen by the author.[138] As with structure, flow, and logic, design is an attribute that may be considered at a high level of abstraction[139] or at a low level of abstrac-

This program instructs the computer to read time-card information and update the employee time record for all employees. That updated record is then used to compute compensation for each employee for the pay period just ended. Straight-time and overtime are computed separately and then summed. The summation is used to compute taxes and social security deductions. Those deductions as well as any other deductions for the pay period are summed. The deduction sum is offset from the compensation sum for each employee and a check is printed.

135. These sequences define in detail the specific way in which, through a series of elementary statements conforming to grammatical rules, a program author has instructed a computer to provide each of the capabilities specified in his program. *See* J. CLEAVELAND & R. UZGADIS, GRAMMARS FOR PROGRAMMING LANGUAGES (1977); G. MYERS, SOFTWARE RELIABILITY: PRINCIPLES AND PRACTICES 89–90 (1976).

136. There are literally hundreds of programming languages and language characteristics can differ substantially. *See* M. BOHL, *supra* note 85, at 391–93. The exposition of program logic elements that follows in the text is based on the PL/1 programming language described in S. POLLACK & T. STERLING, A GUIDE TO PL/1 39–55, 313–27, 365 (1969) and M. AUGENSTEIN & A. TENENBAUM, DATA STRUCTURES AND PL/1 PROGRAMMING 47–81, 97 (1979).

137. One could as well say that taken individually neither the notation of an A Minor chord or a three-quarter note middle C or a whole-note rest would be highly expressive or arbitrary in itself. Taken in a combination and in context, however, the aggregations of elemental music notational elements express a vast body of highly creative effort.

138. *See* D. VAN TASSEL, *supra* note 102, at 41–112.

139. *E.g.*, "This program is divided into twenty modules and makes heavy use of macros."

tion.[140] The low level of abstraction, the program's detailed design, is a complex web of structure, sequence, pattern, and organization. The resulting combination is a tapestry of decisions and actions that is the essence of the author's expression.[141]

5. Naming Conventions

The writer of a program, unless he or she is writing in machine language, must invent names for numerous elements of his or her program. Reusable blocks of code, data items, and variables are generally referred to by name. These names are entirely arbitrary since they are converted into memory addresses when the program is translated into machine language. The computer does not in fact "see" the names that are chosen. Thus, within very modest constraints (word length, reserved names, etc.), the program author's range of choice for naming conventions is very large indeed. As one would expect, therefore, the choice of names tends to reflect the personality and experiences of the individual programmer.[142] Since most programs are intended to be read by other programmers, it is preferable for names to convey some sense of the purpose of the thing being named,[143] but the reference may be explicit or cryptic, abbreviated or full, tied to the name of another program element or not, or totally arbitrary, at the whim of the author.

6. Comments

It is common for programmers to insert comments in their programs to assist themselves and others in understanding the programs.[144] Like names, the comments are

140. *E.g.*, "At this point, we test to see if the number of years exceeds ten. If it does, we branch to a subroutine to calculate the pension amount and store the result in the field called 'Retmnt' of the control block titled 'PersBen'.

141. *See* P. BRUCE & S. PEDERSON, THE SOFTWARE DEVELOPMENT PROJECT: PLANNING AND MANAGEMENT 85–86 (1982).

142. *See* G. WEINBERG, *supra* note 124, at 223–24.

143. B. KERNIGHAN & P. PLAUGER, ELEMENTS OF PROGRAMMING STYLE 144–45 (2d ed. 1978); D. VAN TASSEL, *supra* note 102, at 11–18.

144. D. VAN TASSEL, *supra* note 102, at 4–7, identifies three types of comments: *Prologue Comments*, which may appear at the beginning of each significant structural element of the program and explain the purpose for that element; *Directory Comments*, which may appear at the start of a lengthy program and provide, in effect, a table of contents; and *Explanatory Comments*, which explain individual logic elements of the program.

not processed by the computer when the program is executed and therefore within broad constraints can be anything the author wishes.[145] They can be numerous or sparse, expansive or concise, abbreviated or full text. The context of the comments and the aspects of the program on which she chooses to comment reflect the arbitrary choices of the author.

7. Programming Style

Not surprisingly, each programmer develops her own style of expression.[146] To some extent, the stylistic characteristics are reflective of the education or training of the programmer, her imagination or intellect, or lack thereof.[147] To some extent, they may result from conventions adopted within her department.[148] In some cases, style is collective rather than individual.[149] The style in which a program is written may also reflect the environment in which the program was written.[150]

Style is reflected in the logic elements that the programmer tends to favor and in the way those elements are employed.[151] It is reflected in the modularity of the program and the extent to which modules call on one another or are self-contained.[152] It is reflected in the data structures chosen by the author.[153] Some programs are elegant; others are

145. *See id.* at 3–9; B. KERNIGHAN & P. PLAUGER, *supra* note 143, at 141–44; B. SCHNEIDERMAN, SOFTWARE PSYCHOLOGY: HUMAN FACTORS IN COMPUTER AND INFORMATION SYSTEMS 66–70 (1980).

146. G. WEINBERG, *supra* note 124 *passim*.

147. *Id.* at 161–99 (discussing the role of intellect, education, and experience in the development of programming styles).

148. *See id.* at 47–139.

149. *See* ADVANCES IN COMPUTER PROGRAMMING MANAGEMENT 135–51 (T. Rullo ed. 1980); P. METZGER, MANAGING A PROGRAMMING PROJECT 86–91 (2d ed. 1981).

150. *See* G. WEINBERG, *supra* note 124, at 67–93. (In particular, note that differences in composition of programming teams may engender explicit differences in structure for programs having the same function. The contrast drawn by Weinberg is that between a team composed of one experienced programmer and four trainees, which could be expected to produce a program consisting of one "main program" component and several relatively small subroutines, and a team of three experienced programmers, which could be expected to produce a program of three modules or "phases.").

151. B. KERNIGHAN & P. PLAUGER, *supra* note 143, at 9–26.

152. *Id.* at 59–78.

153. D. VAN TASSEL, *supra* note 102, at 52.

clumsy.[154] Some are written to execute very rapidly in a processor; others are written to take up little memory.[155] Some are written using techniques that reduce the programmer's writing time, irrespective of the resulting execution time or memory space utilization.[156] There are other indicia of individual style, oriented toward the readability of the program to other programmers: the use of comments, for example, or the choice of labels.[157] All these elements combine to give a program the same individuality that makes one novelist's work different from another.

C. *Potential for Alternative Expression*

Commentators and copyright defendants have argued that the potential for alternative expression in computer programs is limited; therefore, they say, the scope of protection for such works should be narrow, akin to that provided for accounting forms or contest rules.[158] Neither the experts nor the cases nor common sense bear out this premise or conclusion.

1. Expert Opinion

Consider first the following description of the craft of writing computer programs, a description that will doubtless surprise many non-programmers. It was written by a gentleman with broad and deep experience in computer science, both in industry and in academia.

> Why is programming fun? What delights may its practitioner expect as his reward?
>
> First is the sheer joy of making things. As the child delights in his mud pie, so the adult enjoys building things, especially things of his own design. I think this delight must be an image of God's delight in making things, a delight shown in the distinctness of newness of each leaf and each snowflake.
>
> Second is the pleasure of making things that are useful to other people. Deep within, we want others to use our work and to find it helpful. . . .
>
> Third is the fascination of fashioning complex puzzle-like objects of interlocking moving parts and watching them work in subtle cycles, playing out the consequences of principles built in from the beginning.
>
> Fourth is the joy of always learning, which springs from the nonrepeating nature of the task. In one way or another the problem is ever new, and its solver learns something: sometimes practical, sometimes theoretical, and sometimes both.
>
> Finally, there is the delight of working in such a tractable medium. The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures. . . .
>
> Programming then is fun because it gratifies creative longings built deep within us and delights sensibilities we have in common with all men.[159]

It seems exceedingly unlikely that a form of authorship in which the range of expression is substantially limited could be described in such terms. Consider, too, the confessions of Data General microcode author Carl Alsing:[160]

> Writing microcode is like nothing else in my life. For days there's nothing coming out. The empty yellow pad sits there in front of me, reminding me of my inadequacy. Finally, it starts to come. I feel good. That feeds it, and finally I get into a mental state where I'm a microcode-writing machine. . . .
>
> You have to understand the problem thoroughly and you have to have thought of all the myriad ways in which you can put your microverbs together. You have a hundred L-shaped blocks to build a building. You take all the pieces, put them together, pull them apart, put them together. After a while, you're like a kid on a jungle gym. There are all these constructs in your mind and you can swing from one to the other with ease.

154. B. KERNIGHAN & P. PLAUGER, *supra* note 143 *passim*, provide a large number of examples of clumsy programs, together with suggested rewritings and general stylistic maxims applicable to each example.

155. *See* G. WEINBERG, *supra* note 124, at 22–25.

156. *Id.* at 19–20.

157. *See supra* text accompanying notes 142–145.

158. *See, e.g.*, *Back to Basics*, *supra* note 9, at 4–6; *Jaslow* Petition, *supra* note 30, at 19–20. Professor Goldstein has suggested that for operating systems, as opposed to applications programs, the limitations of expression and reasons for narrow protection are particularly acute. *See Infringement of Copyright*, *supra* note 9, at 1126–27. That such a view is predicated on factual inaccuracy is eloquently demonstrated by the fact that both the Brooks and Alsing quotes reproduced in this section relate to the writing of operating system software.

159. BROOKS, THE MYTHICAL MAN-MONTH 7–8 (1975).

160. T. KIDDER, THE SOUL OF A NEW MACHINE 101–02 (1981). Microcode, or microprogramming, is roughly analogous to an assembly-language program. *See* M. MANDL, *supra* note 53, at 263.

> I've done this in short intervals for a short period each year. There's low intensity before it and a letdown at the end. There's a big section where you come down off it, and sometimes you do it awkwardly and feel a little strange, wobbly and tired, and you want to say to your friends, "Hey, I'm back."

Alsing's description affirms that computer programming is a kind of creative writing. Indeed, as much as the employers of computer programmers would love to "structure" their work[161] and think of it as "software engineering,"[162] it is generally accepted in the industry that programming is art as much as science.[163]

In the United States, the National Commission on New Technological Uses of Copyrighted Works (CONTU), whose Final Report was the basis for the 1980 amendments to the Copyright Act, considered the question of the range of programming expression explicitly and concluded that in program authorship the "availability of alternative noninfringing language is the rule rather than the exception."[164]

2. Judicial Authority

Those who have actually written computer programs feel that the medium is a very creative one. Nonetheless, as noted at the outset of this section, commentators and copyright defendants often argue that there is a limited range of expression for software. Such arguments have been resoundingly rejected by the courts that have considered the question. As the court recognized in *SAS Institute, Inc. v. S&H Computer Systems, Inc.*,

161. *See, e.g.*, C. McGOWAN & J. KELLY, TOP-DOWN STRUCTURED PROGRAMMING TECHNIQUE (1975).

162. *See, e.g.*, R. JENSEN & C. TONIES, SOFTWARE ENGINEERING (1979); *see also Protectionism, supra* note 19, at 39.

163. The decision to write a program "is very much like the decision to write a novel or to write a poem. In both software and the literary analog we are dealing with a highly creative activity." J. LEATHRUM, FOUNDATIONS OF SOFTWARE DESIGN 11 (1983). Ben Schneider, Jr., opines that the concepts underlying "tops-down" programming and "structured" programming have been known to writers of fiction "at least since Homer." Schneider, *Programs as Essays*, DATAMATION, May, 1984, at 162.

164. FINAL REPORT OF THE NATIONAL COMMISSION ON NEW TECHNOLOGICAL USES OF COPYRIGHTED WORKS 20 n.106 (July 31, 1978) [hereinafter CONTU Report]. In support of that conclusion, CONTU quoted the following colloquy from its tenth meeting:

> Throughout the preparation of a complicated computer program such as SAS, the author is faced with a virtually endless series of decisions as to how to carry out the assigned task. . . . The author must decide how to break the assigned task into smaller tasks, each of which must in turn be broken down into successively smaller and more detailed tasks. . . . At every level, the process is characterized by choice, often made arbitrarily, and only occasionally dictated by necessity. Even in the case of simple statistical calculations, there is room for variation, such as the order in which arithmetic operations are performed. . . . As the sophistication of the calculation increases, so does the opportunity for variation of expression.[165]

Similarly, the Federal Court of Canada, Trial Division, had the following to say about computer programs:

> There is no doubt that computer programs are highly individualistic in nature and contain a form of expression personal to the individual programmer. No two programmers would ever write a program in exactly the same way (except perhaps in the case of the most simple program). Even the same programmer, after writing a program and leaving it for some time, would not write the program the same way on a second occasion. The sequence of instructions would most certainly be different. The possibility of two programmers creating identical programs without copying was compared by the

> Commissioner Miller: How many different ways are there to produce a program . . . ?
>
> Dan McCracken [Vice-President of the Association for Computing Machinery]: An infinite number in principle, and in practice dozens, hundreds.
>
> Miller: So it is comparable to the theoretically infinite number of ways of writing Hamlet?
>
> McCracken: I believe so. It is not really true that there is a very restrictive way to write a program which might make it not copyrightable. I don't believe that at all.
>
> Miller: When you say 'infinite,' I assume that along that scale there are increases and decreases in the efficiency with which the machine will operate?
>
> McCracken: Perhaps.
>
> Miller: In all of the programs that we have been talking about this morning, with particular reference to . . . compiler programs, does it continue to be true that there are an infinite number of ways of writing particular programs to do particular jobs?
>
> McCracken: Yes . . . There are hundreds of different compiler programs for going from FORTRAN to some machines. . . .

Id.

165. SAS Inst. v. S & H Computer Sys., 605 F. Supp. 816, 825 (M.D. Tenn. 1985).

> defendants' expert witness to the likelihood of a monkey sitting at a typewriter producing Shakespeare.[166]

The U.S. Court of Appeals for the Fourth Circuit has reached a similar conclusion:

> [I]n the computer field "[t]here exists a virtually unlimited number of instruction sequences that would enable a programmer to construct a program which performs even the more basic algorithmic or mathematical procedures." . . . It follows, therefore, that normally in the computer field, courts are concerned with expression and not idea, as those terms are defined in copyright law.[167]

Most recently, in the *Whelan* case,[168] the Court of Appeals for the Third Circuit, after carefully considering how computer programs are written, took pains to reject the argument that the process of development and progress in computer programming is significantly different from that in other areas of science or the arts.

The argument that limited possibilities for alternative expression exist often arises in connection with one firm's business strategy to offer programs that are, in one sense or another, "compatible replacements" for the programs of another firm.[169] The replacer claims that its compatibility strategy dictates close conformance, if not identity, with the expression in the target program; in other words, that the "idea" and the "expression" in the target program have merged, and since copyright does not protect ideas, it does not, as a result of the merger, protect the expression either.[170] Even in that context, however, the courts so far have not been convinced that there were few or no other ways in which the replacer could have expressed the replacing program. In the first place, as the Court of Appeals for the Third Circuit pointed out in *Apple v. Franklin*, the fact

166. Apple Computer, Inc. v. Mackintosh Computers, Ltd., No. T-1232-84, No. T-1235-84, slip op. at 3 (F. C. Can., Apr. 29, 1986).

167. M. Kramer Mfg. Co., Inc. v. Andrews, 783 F.2d 421, 436 (4th Cir. 1986).

168. Whelan Assoc. v. Jaslow Dental Laboratory, Inc., 797 F.2d 1222, 1238 (3d Cir. 1986).

169. *See* Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1253 (3d Cir. 1983); E.F. Johnson Co. v. Uniden Corp., 623 F. Supp. 1485, 1501 (D. Minn. 1985); SAS Inst., Inc. v. S & H Computer Sys., Inc., 605 F. Supp. 816, 825 (M.D. Tenn. 1985); Apple Computer, Inc. v. Formula Int'l, Inc., 562 F. Supp. 775, 782 (C.D. Cal. 1983); *see also Protectionism*, *supra* note 19, at 63-72.

170. *See* Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1253 (3rd Cir. 1983); Infringer's Case, *supra* note 28, at 24.

that one firm establishes a compatibility strategy cannot *ipso facto* cause a merger of idea and expression in the program of another firm.[171] Secondly, the courts that have focused on this question have found that even if the idea of compatibility is adopted, the range of expression is not limited.[172]

In short, the courts that have actually considered the question have concluded that idea and expression do not as a rule merge in computer programs.

3. Commentators

How does one account, then, for the contrary views expressed by some commentators? How can it be that a body such as the Office of Technology Assessment of the United States Congress can conclude that if the copyrightable ex-

171. Franklin claims that whether or not the programs can be rewritten, there are a limited "number of ways to arrange operating systems to enable a computer to run the vast body of Apple-compatible software". . . . This claim has no pertinence to either the idea/expression dichotomy or merger. The idea which may merge with the expression, thus making the copyright unavailable, is the idea which is the subject of the expression. The idea of one of the operating system programs is, for example, how to translate source code into object code. If other methods of expressing that idea are not foreclosed as a practical matter, then there is no merger. . . .

> Franklin may wish to achieve total compatibility with independently developed application programs written for the Apple II, but that is a commercial and competitive objective which does not enter into the somewhat metaphysical issue of whether particular ideas and expressions have merged.

Apple v. Franklin, 714 F.2d at 1253.

172. "Many different computer programs can produce the same 'results,' whether those results are an analysis of financial records or a sequence of images and sounds. . . . Obviously, writing a new program to replicate the play of 'Scramble' requires a sophisticated effort, but it is a manageable task." Stern Elec., Inc. v. Kaufman, 669 F.2d 852, 855 (2d Cir. 1982).

> Defendant will still be free to produce programs which result in the machine performing the same calculations, setups, memory loading, etc., as Plaintiff's software does and thus compete with Plaintiff in the software market. As the CONTU Report aptly put it, "One is always free to make the machine do the same thing as it would if it had copyrighted work placed in it, but only by one's own creative effort rather than by piracy."

Apple Computer, Inc. v Formula Int'l, Inc., 562 F. Supp. 775 (C.D. Cal. 1983), *aff'd*, 725 F.2d 521 (9th Cir. 1984). "Thus, the mere fact that defendant set out with the objective of creating an LTR-compatible radio does not, without more, excuse its copying of plaintiff's code. The Court finds that copying plaintiff's code was not the only and essential means of creating an LTR-compatible software program." E.F. Johnson Co. v. Uniden Corp., 623 F. Supp. 1485, 1502 (D. Minn. 1985).

pression in computer programs is deemed to include the programs' "design," "logic," or "structure," the result will be the extension of copyright protection to procedures, processes, systems, or methods of operation?[173] The explanation lies in the somewhat abstruse nature of programming works that puts intellectual distance between, on the one hand, the legal commentators who have not studied the craft of programming and, on the other hand, the programming experts who know the territory and the judges who have learned the territory. Put another way, the judges who are charged with deciding what constitutes programming expression, in the concrete context of two programs whose attributes must be first identified and then compared, have learned more about what programming actually is than have commentators who deal in a more abstract way with matters of theory or policy.

Consider the following example. Federated Computers, Inc. writes a new program for its line of computers. The new program is an instant success and materially advances sales of FCI computers. The Consolidated Computer Company offers a competitive line of computers. Consolidated translates the new Federated program into a language more appropriate for Consolidated computers and offers the translation commercially, increasing the competitiveness of its line of computers against Federated's. Because of differences between the language in which the Federated program was originally written and the Consolidated language, the translation bears little immediate visual resemblance to the original. Has Consolidated infringed Federated's copyright?

The instinctive answer is: obviously. However, some commentators would disagree or at least equivocate.[174] They might argue that unless each line of the original program has been translated individually, there is no infringement. The rationale is apparently that at any level of specificity higher than that of individual lines of code, the program consists of ideas, not expression. It should be evident to anyone who has read this far that the rationale is fallacious. To make the point absolutely clear, it will be helpful to consider, as, for example, the *Whelan* court found it necessary to consider,[175] how programs are written.

173. OTA STUDY, *supra* note 9, at 81.

174. *See, e.g.*, *Back to Basics*, *supra* note 9, at 7; *Infringement of Copyright*, *supra* note 9; *Protectionism*, *supra* note 19, at 79; Note, *Copyright Infringement of Computer Programs: A Modification of the Substantial Similarity Test*, 68 MINN. L. REV. 1264, 1294–99 [hereinafter *A Modification*].

4. The Process of Writing a Program

Without trying to compartmentalize too rigidly the steps or their order, the process is likely to encompass in some sense or other the following phases:[176]

Product definition, in which a general idea of what the product should be able to do to meet customer requirements and to be competitive is developed.

Generalized design, in which the general idea for performance objectives is translated into a plan as to how the program will be written. At this stage, the overall job the program is to perform may be divided into discrete components to make the job more manageable.

Intermediate design, in which the sections defined in the high-level design are elaborated in more detail, resulting in definition of the major structures of the program; *i.e.*, identification of distinct modules, identification of control block structures, and identification of the specific information that will be passed among components of the program.

Detailed design, in which the design is broken down into codeable units (subroutines, control blocks, macros, and the like). By this point, the logic or plan of the program should be so detailed that the starting point for every unit can be defined with such particularity that, if the size of the project warranted it, each could be assigned in parallel to individual programmers with reasonable confidence that the pieces would fit together as a whole once written. The detailed design may take the form of pseudocode or flow charts that fully define what the code will accomplish in human-reada-

175. Whelan Assoc. v. Jaslow Dental Laboratory, Inc., 797 F.2d 1222, 1229–31 (3d Cir. 1986).

176. In actual practice, the process of writing programs is both less rigid and less linear than a serial description of the stages of development may suggest. Two or more of the described phases may be collapsed into one, for example, or earlier phases may be revisited as the writing occurs. The text description following this footnote is adapted from P. METZGER, *supra* note 149 and G. HICE, W. TURNER & L. CASHWELL, SYSTEM DEVELOPMENT METHODOLOGY (rev. ed. 1978). Discussion of the extensive testing and documentation activity which accompanies the described development phases has been omitted for purposes of clarity.

ble words or symbols. The detailed design documents constitute a complete articulation of the program in English words or graphic symbols.

Coding, in which the English or graphic descriptions contained in the detailed design documents are translated into programming language. This part of the process of program development can often be routinely entrusted to a beginning programmer.[177] From the foregoing it can be seen that the detailed design of the program, being the equivalent of a paragraph-level outline of a novel, is the complete expression of the program of which the ultimate, coded text is a translation. In any other work of authorship, copying the elements of the ultimate text that constitute its detailed design would constitute infringement. According to Professor Nimmer, CONTU concluded that there is no reason for a different treatment of computer programs.[178]

There are practical, evidentiary considerations that compel this conclusion as well. Among them, in our Federated/Consolidated example, is the problem of proving that the Consolidated program is a translation of the Federated program. Remember that the source code listings of the two programs look about as much alike, and are about as decipherable to the untutored lay observer, as a Farsi translation set beside a Mandarin Chinese translation of Hamlet's soliloquy. Absent an admission of copying by Consolidated, proof of substantial similarity can be made to a non-expert trier of fact unfamiliar with either language only by showing that though the programs look dissimilar there are actually substantial similarities in some or all of the following: detailed structure, flow, logic, design, naming conventions, and comments.

5. Policy Considerations

It is sometimes argued that the copyright law is for the benefit of the public, not the author, and that a more modest scope of protection would redound to the public good by encouraging the rapid proliferation of equivalent pro-

177. R. GUNTHER, MANAGEMENT METHODOLOGY FOR SOFTWARE PRODUCT ENGINEERING 49 (1978); *see also Whelan*, 797 F.2d at 1231.

178. Nimmer Declaration, *supra* note 3, at pt. 25 (*see infra* Appendix). The *Whelan* court reached the same conclusion. 797 F.2d at 1238.

grams.[179] This argument overlooks three important factors. First, the Founding Fathers concluded that the public would be best served by granting exclusive rights to copyright owners.[180] Second, the public has in fact benefited immensely from the incentive that the copyright laws provide. Permitting free translation would eliminate the competitive lead time that is a major incentive for investment in new pro-

179. *Jaslow* Petition, *supra* note 30, at 13; *A Modification*, *supra* note 174, at 1291; Note, *Defining the Scope of Copyright Protection for Computer Software*, 38 STAN. L. REV. 497, 498, 518 (1986); *cf.* Note, *Copyright Protection of Computer Program Object Code*, 96 HARV. L. REV. 1723, 1739–42 (1983) (citing, *inter alia*, Twentieth Century Music Corp. v. Aiken, 422 U.S. 151, 156 (1975)). One such line of reasoning proceeds from the premise that the purpose of the copyright grant is to promote the dissemination of ideas and that the texts of computer programs, being unintelligible to most people, do not comport with that purpose. *See* Samuelson, *CONTU Revisited: The Case Against Copyright Protection for Computer Programs in Machine-Readable Form*, 1984 DUKE L.J. 663, 705–53 (1984). That line of reasoning relies on an unwarranted leap of faith. First, it is simply false that the text of a computer program is unintelligible. The fact that it requires a level of competence to read a program with appreciation merely puts programs in the company of foreign language texts and other specialized literature. It does not negate a dissemination of ideas. Second, the argument confuses the end with the means. The purpose of the Copyright Act is to "promote the progress of science and the useful arts." U.S. CONST. art. I, § 8. One means to serve that purpose is indeed by promoting the free dissemination of ideas. That was not, however, the means Congress adopted pursuant to the constitutional mandate.

The framers decided that the constitutional purpose will be promoted by allowing authors to control, in appropriate cases, public access to the expressions in which ideas are embodied. This reflects a judgment that dissemination of ideas, as well as other elements of "progress" in science and the useful arts, will be advanced by enabling authors, as a class, to pursue their self-interest. The law simply does not compel dissemination in every case. This is reflected, for example, in the 1976 Act's statement that registration is *not* a prerequisite to copyright protection. Copyright attaches as soon as a work is fixed in a tangible medium of expression, whether or not the work is ever registered or ever published. Thus, for example, the author J.D. Salinger is able to assert copyright protection in his private letters. *See* Salinger v. Random House, Inc., 811 F.2d 90 (2d Cir. 1987). Even before the 1976 Act, the requirement of deposit of works to be registered was not designed primarily to insure a complete collection of all copyrighted works open to the public. Deposited copies could be, and were, returned, distributed, or destroyed under the direction of the Library of Congress. *See* Washingtonian Publishing Co. v. Pearson, 306 U.S. 30, 38–39 (1939). Moreover, the registration process itself does not require that the registered work be available to the public. Copyright Office procedures permit deposit of less than all (the first 25 pages and the last 25 pages) of a program sought to be registered, Copyright Office Circular R61, May 1983, at 1, and also provide for relief from the deposit requirement altogether, so that after registration such works are not available for inspection by the public. Copyright Office Circular R7d, August, 1982, at 2. In sum, views based on the assumption that the purpose of the copyright grant can only be satisfied by free public availability of the copyrighted work, although they may be creditable at first blush, do not have a firm foundation.

180. *See* U.S. CONST. art. I, § 8, cl. 8, *supra* note 4.

grams.[181] Finally, allowing infringers to profit from translations of the copyrighted work will encourage authors to write non-translatable programs.[182] These factors assure that the public good would be disadvantaged by depriving program authors of exclusive rights to translate or adapt their works.

Thus, in software infringement cases, application of traditional rules of copyright law rather than invention of special rules for computer programs is the most socially desirable approach. We now turn to the question whether application of traditional copyright rules is what the law requires.